

Selenium c# (Visual Studio, Nunit, TFS and MTM) dll:

1. Selenium.WebDriver.3.0.1
2. Selenium.Support.3.0.1
3. Selenium.WebDriver.IEDriver.3.0.0.1
4. Selenium.Firefox.WebDriver.0.13.0
5. Selenium.WebDriver.ChromeDriver.2.27.0
6. Excel – Interop dll – Microsoft Excel 15.0 Object Library – built in
7. Excel : Microsoft.Csharp
8. Excel – NPOI dll
9. XML – System.XML
10. API – RestSharp
11. Service – Add Service Reference
12. Nunit – nuit.framework (nunit3, unit3 test adapter – NuGet Package manger)

Reference:

Frame: http://jqueryui.com/draggable/

Alert: http://irctc.co.in

Window: http://www.crystalcruises.com/

Nunit : http://www.c-sharpcorner.com/UploadFile/093731/selenium-webdriver-nunit-with-visual-studio/

Excel Compare:

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.Runtime.InteropServices;

using Excel = Microsoft.Office.Interop.Excel; //microsoft Excel 14 object in references-> COM tab

namespace SeleniumCsharpPractice.Utilities

{

[TestClass]

public class ExcelCompareTest

{

[TestMethod]

public void excelCompareMethod()

{

//Create COM Objects. Create a COM object for everything that is referenced

Excel.Application xlApp1 = new Excel.Application();

Excel.Workbook xlWorkbook1 = xlApp1.Workbooks.Open(@"D:\Pioneer\epen\ProjectDefinitionXL.xlsx");

Excel.Application xlApp2 = new Excel.Application();

Excel.Workbook xlWorkbook2 = xlApp2.Workbooks.Open(@"D:\Pioneer\epen\ProjectDefinitionCSV.xlsx");

Excel.\_Worksheet xlWorksheet1 = xlWorkbook1.Sheets[1];

Excel.\_Worksheet xlWorksheet2 = xlWorkbook2.Sheets[1];

Excel.Range xlRange1 = xlWorksheet1.UsedRange;

Excel.Range xlRange2 = xlWorksheet2.UsedRange;

int rowCount = xlRange1.Rows.Count;

int colCount = xlRange1.Columns.Count;

for (int i = 1; i <= rowCount; i++)

{

for (int j = 1; j <= colCount; j++)

{

//new line

// if (j == 1)

// Debug.WriteLine("\r\n");

//write the value to the console

// if (xlRange1.Cells[i, j] != null && xlRange1.Cells[i, j].Value2 != null)

// Console.Write(xlRange1.Cells[i, j].Value2.ToString() + "\t");

// String val = xlRange1.Cells[i, j].Value2;

// String val1 = xlRange2.Cells[i, j].Value2;

if (xlRange1.Cells[i, j] != null && xlRange1.Cells[i, j].Value2 != null && xlRange2.Cells[i, j] != null && xlRange2.Cells[i, j].Value2 != null)

{

if (xlRange1.Cells[i, j].Value2.ToString() != xlRange2.Cells[i, j].Value2.ToString())

{

xlRange1.Cells[i, j].Interior.ColorIndex = 3;

// Debug.WriteLine("val :" + "\t" + val);

}

}

}

}

//cleanup

GC.Collect();

GC.WaitForPendingFinalizers();

//rule of thumb for releasing com objects:

// never use two dots, all COM objects must be referenced and released individually

// ex: [somthing].[something].[something] is bad

//release com objects to fully kill excel process from running in the background

Marshal.ReleaseComObject(xlRange1);

Marshal.ReleaseComObject(xlWorksheet1);

Marshal.ReleaseComObject(xlRange2);

Marshal.ReleaseComObject(xlWorksheet2);

// save the workbook

xlWorkbook1.Save();

xlWorkbook2.Save();

//close and release

xlWorkbook1.Close();

Marshal.ReleaseComObject(xlWorkbook1);

xlWorkbook2.Close();

Marshal.ReleaseComObject(xlWorkbook2);

//quit and release

xlApp1.Quit();

Marshal.ReleaseComObject(xlApp1);

xlApp2.Quit();

Marshal.ReleaseComObject(xlApp2);

}

}

}

ExcelCompareRandomCheck:

// this program will read each and every row form source workbook and search it in destination workbook,

// if the match found it will highlight the row in green color, if not highlight in red color.

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.Runtime.InteropServices;

using Excel = Microsoft.Office.Interop.Excel; //microsoft Excel 14 object in references-> COM tab

using System.Diagnostics;

namespace SeleniumCsharpPractice.Utilities

{

[TestClass]

public class ExcelCompareRandomCheck

{

string sheet1cell1 = null;

string sheet1cell2 = null;

string sheet1cell3 = null;

string sheet1cell4 = null;

string sheet1cell5 = null;

string sheet1cell6 = null;

string sheet1cell7 = null;

string sheet1cell8 = null;

string sheet2cell1 = null;

string sheet2cell2 = null;

string sheet2cell3 = null;

string sheet2cell4 = null;

string sheet2cell5 = null;

string sheet2cell6 = null;

string sheet2cell7 = null;

string sheet2cell8 = null;

string sheet1row = null;

string sheet2row = null;

[TestMethod]

public void excelCompareRandomCheckMethod()

{

//Create COM Objects. Create a COM object for everything that is referenced

Excel.Application xlApp1 = new Excel.Application();

Excel.Workbook xlWorkbook1 = xlApp1.Workbooks.Open(@"C:\Users\616181\Desktop\File - Copy.xls");

Excel.Application xlApp2 = new Excel.Application();

Excel.Workbook xlWorkbook2 = xlApp2.Workbooks.Open(@"C:\Users\616181\Desktop\File1 - Copy.xls");

Excel.\_Worksheet xlWorksheet1 = xlWorkbook1.Sheets[1];

Excel.\_Worksheet xlWorksheet2 = xlWorkbook2.Sheets[1];

Excel.Range xlRange1 = xlWorksheet1.UsedRange;

Excel.Range xlRange2 = xlWorksheet2.UsedRange;

int rowCount1 = xlRange1.Rows.Count;

Debug.WriteLine("row count1 : " + rowCount1);

int rowCount2 = xlRange2.Rows.Count;

Debug.WriteLine("row count2 : " + rowCount2);

for (int i = 1; i <= rowCount1; i++)

{

if (xlRange1.Cells[i, 1] != null && xlRange1.Cells[i, 1].Value2 != null &&

xlRange1.Cells[i, 2] != null && xlRange1.Cells[i, 2].Value2 != null &&

xlRange1.Cells[i, 3] != null && xlRange1.Cells[i, 3].Value2 != null &&

xlRange1.Cells[i, 4] != null && xlRange1.Cells[i, 4].Value2 != null &&

xlRange1.Cells[i, 5] != null && xlRange1.Cells[i, 5].Value2 != null &&

xlRange1.Cells[i, 6] != null && xlRange1.Cells[i, 6].Value2 != null &&

xlRange1.Cells[i, 7] != null && xlRange1.Cells[i, 7].Value2 != null &&

xlRange1.Cells[i, 8] != null && xlRange1.Cells[i, 8].Value2 != null)

{

sheet1cell1 = xlRange1.Cells[i, 1].Value2.ToString();

sheet1cell2 = xlRange1.Cells[i, 2].Value2.ToString();

sheet1cell3 = xlRange1.Cells[i, 3].Value2.ToString();

sheet1cell4 = xlRange1.Cells[i, 4].Value2.ToString();

sheet1cell5 = xlRange1.Cells[i, 5].Value2.ToString();

sheet1cell6 = xlRange1.Cells[i, 6].Value2.ToString();

sheet1cell7 = xlRange1.Cells[i, 7].Value2.ToString();

sheet1cell8 = xlRange1.Cells[i, 8].Value2.ToString();

sheet1row = sheet1cell1 + ' ' + sheet1cell2 + ' ' + sheet1cell3 + ' ' + sheet1cell4 + ' ' +

sheet1cell5 + ' ' + sheet1cell6 + ' ' + sheet1cell7 + ' ' + sheet1cell8;

Debug.WriteLine("sheet1row : " + sheet1row);

Debug.WriteLine("sheet1cell1 : " + sheet1cell1);

Debug.WriteLine("sheet1cell2 : " + sheet1cell2);

Debug.WriteLine("sheet1cell3 : " + sheet1cell3);

Debug.WriteLine("sheet1cell4 : " + sheet1cell4);

Debug.WriteLine("sheet1cell5 : " + sheet1cell5);

Debug.WriteLine("sheet1cell6 : " + sheet1cell6);

Debug.WriteLine("sheet1cell7 : " + sheet1cell7);

Debug.WriteLine("sheet1cell8 : " + sheet1cell8);

}

for (int j = 1; j <= rowCount2; j++)

{

// for (int j = 1; j <= colCount; j++)

// {

//new line

// if (j == 1)

// Debug.WriteLine("\r\n");

//write the value to the console

// if (xlRange1.Cells[i, j] != null && xlRange1.Cells[i, j].Value2 != null)

// Console.Write(xlRange1.Cells[i, j].Value2.ToString() + "\t");

// String val = xlRange1.Cells[i, j].Value2;

// String val1 = xlRange2.Cells[i, j].Value2;

if (xlRange2.Cells[j, 1] != null && xlRange2.Cells[j, 1].Value2 != null &&

xlRange2.Cells[j, 2] != null && xlRange2.Cells[j, 2].Value2 != null &&

xlRange2.Cells[j, 3] != null && xlRange2.Cells[j, 3].Value2 != null &&

xlRange2.Cells[j, 4] != null && xlRange2.Cells[j, 4].Value2 != null &&

xlRange2.Cells[j, 5] != null && xlRange2.Cells[j, 5].Value2 != null &&

xlRange2.Cells[j, 6] != null && xlRange2.Cells[j, 6].Value2 != null &&

xlRange2.Cells[j, 7] != null && xlRange2.Cells[j, 7].Value2 != null &&

xlRange2.Cells[j, 8] != null && xlRange2.Cells[j, 8].Value2 != null)

{

sheet2cell1 = xlRange2.Cells[j, 1].Value2.ToString();

sheet2cell2 = xlRange2.Cells[j, 2].Value2.ToString();

sheet2cell3 = xlRange2.Cells[j, 3].Value2.ToString();

sheet2cell4 = xlRange2.Cells[j, 4].Value2.ToString();

sheet2cell5 = xlRange2.Cells[j, 5].Value2.ToString();

sheet2cell6 = xlRange2.Cells[j, 6].Value2.ToString();

sheet2cell7 = xlRange2.Cells[j, 7].Value2.ToString();

sheet2cell8 = xlRange2.Cells[j, 8].Value2.ToString();

sheet2row = sheet2cell1 + ' ' + sheet2cell2 + ' ' + sheet2cell3 + ' ' + sheet2cell4 + ' ' +

sheet2cell5 + ' ' + sheet2cell6 + ' ' + sheet2cell7 + ' ' + sheet2cell8;

Debug.WriteLine("sheet2row : " + sheet2row);

Debug.WriteLine("sheet2cell1 : " + sheet2cell1);

Debug.WriteLine("sheet2cell2 : " + sheet2cell2);

Debug.WriteLine("sheet2cell3 : " + sheet2cell3);

Debug.WriteLine("sheet2cell4 : " + sheet2cell4);

Debug.WriteLine("sheet2cell5 : " + sheet2cell5);

Debug.WriteLine("sheet2cell6 : " + sheet2cell6);

Debug.WriteLine("sheet2cell7 : " + sheet2cell7);

Debug.WriteLine("sheet2cell8 : " + sheet2cell8);

if (sheet1row.Equals(sheet2row))

{

// if (xlRange1.Cells[i, 19].Value2.ToString().Equals("Y") && xlRange2.Cells[i, 19].Value2.ToString().Equals("Y"))

// {

xlRange1.Cells[i, 1].Interior.ColorIndex = 4;

xlRange1.Cells[i, 2].Interior.ColorIndex = 4;

xlRange1.Cells[i, 3].Interior.ColorIndex = 4;

xlRange1.Cells[i, 4].Interior.ColorIndex = 4;

xlRange1.Cells[i, 5].Interior.ColorIndex = 4;

xlRange1.Cells[i, 6].Interior.ColorIndex = 4;

xlRange1.Cells[i, 7].Interior.ColorIndex = 4;

xlRange1.Cells[i, 8].Interior.ColorIndex = 4;

xlRange2.Cells[j, 1].Interior.ColorIndex = 4;

xlRange2.Cells[j, 2].Interior.ColorIndex = 4;

xlRange2.Cells[j, 3].Interior.ColorIndex = 4;

xlRange2.Cells[j, 4].Interior.ColorIndex = 4;

xlRange2.Cells[j, 5].Interior.ColorIndex = 4;

xlRange2.Cells[j, 6].Interior.ColorIndex = 4;

xlRange2.Cells[j, 7].Interior.ColorIndex = 4;

xlRange2.Cells[j, 8].Interior.ColorIndex = 4;

xlRange2.Cells[j, 20].Value2 = "row" + i;

break;

// Debug.WriteLine("val :" + "\t" + val);

}

//else

//{

// xlRange1.Cells[i, 1].Interior.ColorIndex = 3;

// xlRange1.Cells[i, 2].Interior.ColorIndex = 3;

// xlRange1.Cells[i, 3].Interior.ColorIndex = 3;

// xlRange1.Cells[i, 4].Interior.ColorIndex = 3;

// xlRange1.Cells[i, 5].Interior.ColorIndex = 3;

// xlRange1.Cells[i, 6].Interior.ColorIndex = 3;

// xlRange2.Cells[j, 1].Interior.ColorIndex = 3;

// xlRange2.Cells[j, 2].Interior.ColorIndex = 3;

// xlRange2.Cells[j, 3].Interior.ColorIndex = 3;

// xlRange2.Cells[j, 4].Interior.ColorIndex = 3;

// xlRange2.Cells[j, 5].Interior.ColorIndex = 3;

// xlRange2.Cells[j, 6].Interior.ColorIndex = 3;

//}

}

}

break;

}

// }

//cleanup

GC.Collect();

GC.WaitForPendingFinalizers();

//rule of thumb for releasing com objects:

// never use two dots, all COM objects must be referenced and released individually

// ex: [somthing].[something].[something] is bad

//release com objects to fully kill excel process from running in the background

Marshal.ReleaseComObject(xlRange1);

Marshal.ReleaseComObject(xlWorksheet1);

Marshal.ReleaseComObject(xlRange2);

Marshal.ReleaseComObject(xlWorksheet2);

// save the workbook

xlWorkbook1.Save();

xlWorkbook2.Save();

//close and release

xlWorkbook1.Close();

Marshal.ReleaseComObject(xlWorkbook1);

xlWorkbook2.Close();

Marshal.ReleaseComObject(xlWorkbook2);

//quit and release

xlApp1.Quit();

Marshal.ReleaseComObject(xlApp1);

xlApp2.Quit();

Marshal.ReleaseComObject(xlApp2);

}

}

}

ExcelCompareRandomCheckGetterSetter:

// this program will read each and every row form source workbook and search it in destination workbook,

// if the match found it will highlight the row in green color, if not highlight in red color.

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.Runtime.InteropServices;

using Excel = Microsoft.Office.Interop.Excel;

using System.Diagnostics;

using System.Collections;

namespace ExcelCompareRandomCheckGetterSetter.Utilities

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void excelCompareRandomCheckGetterSetterMethod()

{

ArrayList excel1 = new ArrayList();

ArrayList excel2 = new ArrayList();

excelbook ebook = new excelbook();

//Create COM Objects. Create a COM object for everything that is referenced

Excel.Application xlApp1 = new Excel.Application();

Excel.Workbook xlWorkbook1 = xlApp1.Workbooks.Open(@"D:\Acompare\ASampleFile1.xls");

Excel.Application xlApp2 = new Excel.Application();

Excel.Workbook xlWorkbook2 = xlApp2.Workbooks.Open(@"D:\Acompare\ASampleFile2.xls");

Excel.\_Worksheet xlWorksheet1 = xlWorkbook1.Sheets[1];

Excel.\_Worksheet xlWorksheet2 = xlWorkbook2.Sheets[1];

Excel.Range xlRange1 = xlWorksheet1.UsedRange;

Excel.Range xlRange2 = xlWorksheet2.UsedRange;

int rowCount = xlRange1.Rows.Count;

int colCount = xlRange1.Columns.Count;

for (int i = 1; i <= rowCount; i++)

{

ebook = new excelbook();

Double d = 0;

double.TryParse(xlRange1.Cells[i, 1].Value2.ToString(), out d);

ebook.e = d;

double.TryParse(xlRange1.Cells[i, 2].Value2.ToString(), out d);

ebook.h = d;

ebook.j = xlRange1.Cells[i, 3].Value2.ToString();

ebook.s = xlRange1.Cells[i, 4].Value2.ToString();

excel1.Add(ebook);

}

rowCount = xlRange2.Rows.Count;

colCount = xlRange2.Columns.Count;

for (int i = 1; i <= rowCount; i++)

{

ebook = new excelbook();

Double d = 0;

double.TryParse(xlRange2.Cells[i, 1].Value2.ToString(), out d);

ebook.e = d;

double.TryParse(xlRange2.Cells[i, 2].Value2.ToString(), out d);

ebook.h = d;

ebook.j = xlRange2.Cells[i, 3].Value2.ToString();

ebook.s = xlRange2.Cells[i, 4].Value2.ToString();

excel2.Add(ebook);

}

for (int i = 0; i < excel1.Count; i++)

{

excelbook empRec = (excelbook)(excel1[i]);

// Debug.WriteLine("failure records" + empRec.e + ", " + empRec.h + ", " + empRec.j + ", " + empRec.s);

// Debug.WriteLine("i = "+i);

for (int j = 0; j < excel2.Count; j++)

{

Debug.WriteLine("j=" + j);

excelbook empRec1 = (excelbook)(excel2[j]);

if (empRec.e == empRec1.e && empRec.h == empRec1.h && empRec.j == empRec1.j)

{

if (empRec.s.Equals(empRec1.s) == true)

{

// Debug.WriteLine("success");

}

else

{

Debug.WriteLine("failure records" + empRec1.e + ", " + empRec1.h + ", " + empRec1.j + ", " + empRec1.s);

// Debug.WriteLine("failure records" + empRec.e + ", " + empRec.h + ", " + empRec.j + ", " + empRec.s);

// Debug.WriteLine("failure records" + i+"\t "+j);

}

}

empRec1 = null;

}

empRec = null;

}

Debug.WriteLine(excel1.Count);

//cleanup

// GC.Collect();

// GC.WaitForPendingFinalizers();

//rule of thumb for releasing com objects:

// never use two dots, all COM objects must be referenced and released individually

// ex: [somthing].[something].[something] is bad

//release com objects to fully kill excel process from running in the background

Marshal.ReleaseComObject(xlRange1);

Marshal.ReleaseComObject(xlWorksheet1);

Marshal.ReleaseComObject(xlRange2);

Marshal.ReleaseComObject(xlWorksheet2);

// save the workbook

xlWorkbook1.Save();

xlWorkbook2.Save();

//close and release

xlWorkbook1.Close();

Marshal.ReleaseComObject(xlWorkbook1);

xlWorkbook2.Close();

Marshal.ReleaseComObject(xlWorkbook2);

//quit and release

xlApp1.Quit();

Marshal.ReleaseComObject(xlApp1);

xlApp2.Quit();

Marshal.ReleaseComObject(xlApp2);

}

}

public class excelbook

{

public double e { get; set; }

public double h { get; set; }

public String j { get; set; }

public String s { get; set; }

}

}

ExcelCompareSpecificColumnsTwo:

// this program will get hot coded tow values and search it in destination workbook,

// if the match found it will highlight the row in green color, if not highlight in red color.

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.Runtime.InteropServices;

using Excel = Microsoft.Office.Interop.Excel; //microsoft Excel 14 object in references-> COM tab

using System.Diagnostics;

namespace SeleniumCsharpPractice.Utilities

{

[TestClass]

public class ExcelCompareSpecificColumnsTwo

{

string val1 = null;

string val2 = null;

string val3 = null;

string val4 = null;

[TestMethod]

public void excelCompareSpecificColumnsMethodTwo()

{

//Create COM Objects. Create a COM object for everything that is referenced

Excel.Application xlApp1 = new Excel.Application();

Excel.Workbook xlWorkbook1 = xlApp1.Workbooks.Open(@"C:\Users\616181\Desktop\File - Copy.xls");

Excel.Application xlApp2 = new Excel.Application();

Excel.Workbook xlWorkbook2 = xlApp2.Workbooks.Open(@"C:\Users\616181\Desktop\File1 - Copy.xls");

Excel.\_Worksheet xlWorksheet1 = xlWorkbook1.Sheets[1];

Excel.\_Worksheet xlWorksheet2 = xlWorkbook2.Sheets[1];

Excel.Range xlRange1 = xlWorksheet1.UsedRange;

Excel.Range xlRange2 = xlWorksheet2.UsedRange;

int rowCount = xlRange1.Rows.Count;

Debug.WriteLine("row count : " + rowCount);

int colCount = xlRange1.Columns.Count;

for (int i = 1; i <= rowCount; i++)

{

// for (int j = 1; j <= colCount; j++)

// {

//new line

// if (j == 1)

// Debug.WriteLine("\r\n");

//write the value to the console

// if (xlRange1.Cells[i, j] != null && xlRange1.Cells[i, j].Value2 != null)

// Console.Write(xlRange1.Cells[i, j].Value2.ToString() + "\t");

// String val = xlRange1.Cells[i, j].Value2;

// String val1 = xlRange2.Cells[i, j].Value2;

if (xlRange1.Cells[i, 5] != null && xlRange1.Cells[i, 5].Value2 != null &&

xlRange1.Cells[i, 8] != null && xlRange2.Cells[i, 8].Value2 != null &&

xlRange2.Cells[i, 5] != null && xlRange1.Cells[i, 8].Value2 != null &&

xlRange2.Cells[i, 8] != null && xlRange2.Cells[i, 8].Value2 != null)

val1 = xlRange1.Cells[i, 5].Value2.ToString();

val2 = xlRange1.Cells[i, 8].Value2.ToString();

val3 = xlRange1.Cells[i, 5].Value2.ToString();

val4 = xlRange1.Cells[i, 8].Value2.ToString();

Debug.WriteLine("value : " + val1);

Debug.WriteLine("value : " + val2);

Debug.WriteLine("value : " + val3);

Debug.WriteLine("value : " + val4);

{

if (val1.Equals("51113") && val2.Equals("408000447318") &&

val3.Equals("51113") && val4.Equals("408000447318"))

{

if (xlRange1.Cells[i, 19].Value2.ToString() == "Y" && xlRange2.Cells[i, 19].Value2.ToString() == "Y")

{

xlRange1.Cells[i, 19].Interior.ColorIndex = 4;

// Debug.WriteLine("val :" + "\t" + val);

}

else

{

xlRange1.Cells[i, 19].Interior.ColorIndex = 3;

}

}

}

}

// }

//cleanup

GC.Collect();

GC.WaitForPendingFinalizers();

//rule of thumb for releasing com objects:

// never use two dots, all COM objects must be referenced and released individually

// ex: [somthing].[something].[something] is bad

//release com objects to fully kill excel process from running in the background

Marshal.ReleaseComObject(xlRange1);

Marshal.ReleaseComObject(xlWorksheet1);

Marshal.ReleaseComObject(xlRange2);

Marshal.ReleaseComObject(xlWorksheet2);

// save the workbook

xlWorkbook1.Save();

xlWorkbook2.Save();

//close and release

xlWorkbook1.Close();

Marshal.ReleaseComObject(xlWorkbook1);

xlWorkbook2.Close();

Marshal.ReleaseComObject(xlWorkbook2);

//quit and release

xlApp1.Quit();

Marshal.ReleaseComObject(xlApp1);

xlApp2.Quit();

Marshal.ReleaseComObject(xlApp2);

}

}

}

ExcelCompareSpecificColumns:

// this program will read specific two values form source workbook and search it in destination workbook,

// if the match found it will highlight the row in green color, if not highlight in red color.

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.Runtime.InteropServices;

using Excel = Microsoft.Office.Interop.Excel; //microsoft Excel 14 object in references-> COM tab

using System.Diagnostics;

namespace SeleniumCsharpPractice.Utilities

{

[TestClass]

public class ExcelCompareSpecificColumns

{

string val1 = null;

string val2 = null;

string val3 = null;

string val4 = null;

string val5 = null;

string val6 = null;

[TestMethod]

public void excelCompareSpecificColumnsMethod()

{

//Create COM Objects. Create a COM object for everything that is referenced

Excel.Application xlApp1 = new Excel.Application();

Excel.Workbook xlWorkbook1 = xlApp1.Workbooks.Open(@"C:\Users\616181\Desktop\File - Copy.xls");

Excel.Application xlApp2 = new Excel.Application();

Excel.Workbook xlWorkbook2 = xlApp2.Workbooks.Open(@"C:\Users\616181\Desktop\File1 - Copy.xls");

Excel.Application xlApp3 = new Excel.Application();

Excel.Workbook xlWorkbook3 = xlApp3.Workbooks.Open(@"C:\Users\616181\Desktop\Values.xlsx");

Excel.\_Worksheet xlWorksheet1 = xlWorkbook1.Sheets[1];

Excel.\_Worksheet xlWorksheet2 = xlWorkbook2.Sheets[1];

Excel.\_Worksheet xlWorksheet3 = xlWorkbook3.Sheets[1];

Excel.Range xlRange1 = xlWorksheet1.UsedRange;

Excel.Range xlRange2 = xlWorksheet2.UsedRange;

Excel.Range xlRange3 = xlWorksheet3.UsedRange;

int rowCount = xlRange1.Rows.Count;

Debug.WriteLine("row count : " + rowCount);

int rowCount1 = xlRange3.Rows.Count;

Debug.WriteLine("row count1 : " + rowCount1);

for (int j = 1; j <= rowCount1; j++)

{

if (xlRange3.Cells[j, 1] != null && xlRange3.Cells[j, 1].Value2 != null &&

xlRange3.Cells[j, 2] != null && xlRange3.Cells[j, 2].Value2 != null)

{

val5 = xlRange3.Cells[j, 1].Value2.ToString();

val6 = xlRange3.Cells[j, 2].Value2.ToString();

Debug.WriteLine("value5 : " + val5);

Debug.WriteLine("value6 : " + val6);

}

for (int i = 1; i <= rowCount; i++)

{

// for (int j = 1; j <= colCount; j++)

// {

//new line

// if (j == 1)

// Debug.WriteLine("\r\n");

//write the value to the console

// if (xlRange1.Cells[i, j] != null && xlRange1.Cells[i, j].Value2 != null)

// Console.Write(xlRange1.Cells[i, j].Value2.ToString() + "\t");

// String val = xlRange1.Cells[i, j].Value2;

// String val1 = xlRange2.Cells[i, j].Value2;

if (xlRange1.Cells[i, 5] != null && xlRange1.Cells[i, 5].Value2 != null &&

xlRange1.Cells[i, 8] != null && xlRange2.Cells[i, 8].Value2 != null &&

xlRange2.Cells[i, 5] != null && xlRange1.Cells[i, 8].Value2 != null &&

xlRange2.Cells[i, 8] != null && xlRange2.Cells[i, 8].Value2 != null)

{

val1 = xlRange1.Cells[i, 5].Value2.ToString();

val2 = xlRange1.Cells[i, 8].Value2.ToString();

val3 = xlRange1.Cells[i, 5].Value2.ToString();

val4 = xlRange1.Cells[i, 8].Value2.ToString();

Debug.WriteLine("value1 : " + val1);

Debug.WriteLine("value2 : " + val2);

Debug.WriteLine("value3 : " + val3);

Debug.WriteLine("value4 : " + val4);

if (val1.Equals(val5) && val2.Equals(val6) &&

val3.Equals(val5) && val4.Equals(val6))

{

if (xlRange1.Cells[i, 19].Value2.ToString().Equals("Y") && xlRange2.Cells[i, 19].Value2.ToString().Equals("Y"))

{

xlRange1.Cells[i, 19].Interior.ColorIndex = 4;

// Debug.WriteLine("val :" + "\t" + val);

}

else

{

xlRange1.Cells[i, 19].Interior.ColorIndex = 3;

}

}

}

}

}

//cleanup

GC.Collect();

GC.WaitForPendingFinalizers();

//rule of thumb for releasing com objects:

// never use two dots, all COM objects must be referenced and released individually

// ex: [somthing].[something].[something] is bad

//release com objects to fully kill excel process from running in the background

Marshal.ReleaseComObject(xlRange1);

Marshal.ReleaseComObject(xlWorksheet1);

Marshal.ReleaseComObject(xlRange2);

Marshal.ReleaseComObject(xlWorksheet2);

Marshal.ReleaseComObject(xlRange3);

Marshal.ReleaseComObject(xlWorksheet3);

// save the workbook

xlWorkbook1.Save();

xlWorkbook2.Save();

xlWorkbook3.Save();

//close and release

xlWorkbook1.Close();

Marshal.ReleaseComObject(xlWorkbook1);

xlWorkbook2.Close();

Marshal.ReleaseComObject(xlWorkbook2);

xlWorkbook3.Close();

Marshal.ReleaseComObject(xlWorkbook3);

//quit and release

xlApp1.Quit();

Marshal.ReleaseComObject(xlApp1);

xlApp2.Quit();

Marshal.ReleaseComObject(xlApp2);

xlApp3.Quit();

Marshal.ReleaseComObject(xlApp3);

}

}

}

ReadFromExcel:

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.Diagnostics;

using System.Runtime.InteropServices;

using Excel = Microsoft.Office.Interop.Excel; //microsoft Excel 14 object in references-> COM tab

namespace SeleniumCsharpPractice.Utilities

{

[TestClass]

public class ReadFromExcel

{

[TestMethod]

public String readFromExcelMethod(int row, int col)

{

String val = null;

//Create COM Objects. Create a COM object for everything that is referenced

Excel.Application xlApp = new Excel.Application();

Excel.Workbook xlWorkbook = xlApp.Workbooks.Open(@"D:\Workspace\TestData\TestData.xlsx");

Excel.\_Worksheet xlWorksheet = xlWorkbook.Sheets[1];

Excel.Range xlRange = xlWorksheet.UsedRange;

int rowCount = xlRange.Rows.Count;

int colCount = xlRange.Columns.Count;

//iterate over the rows and columns and print to the console as it appears in the file

//excel is not zero based!!

//for (int i = 1; i <= rowCount; i++)

//{

// for (int j = 1; j <= colCount; j++)

// {

// //new line

// if (j == 1)

// Debug.WriteLine("\r\n");

//write the value to the console

if (xlRange.Cells[row, col] != null && xlRange.Cells[row, col].Value2 != null)

{

Console.Write(xlRange.Cells[row, col].Value2.ToString() + "\t");

val = xlRange.Cells[row, col].Value2;

Debug.WriteLine("val :" + "\t" + val);

}

// }

// return val;

//}

//return val;

// xlRange.Cells[4, 4].value = "baskar";

// xlRange.Cells[4, 4].Interior.ColorIndex = 3;

//cleanup

GC.Collect();

GC.WaitForPendingFinalizers();

//rule of thumb for releasing com objects:

// never use two dots, all COM objects must be referenced and released individually

// ex: [somthing].[something].[something] is bad

//release com objects to fully kill excel process from running in the background

Marshal.ReleaseComObject(xlRange);

Marshal.ReleaseComObject(xlWorksheet);

// save the workbook

xlWorkbook.Save();

//close and release

xlWorkbook.Close();

Marshal.ReleaseComObject(xlWorkbook);

//quit and release

xlApp.Quit();

Marshal.ReleaseComObject(xlApp);

return val;

}

}

}

ExcelToTextCompare:

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.Runtime.InteropServices;

using Excel = Microsoft.Office.Interop.Excel; //microsoft Excel 14 object in references-> COM tab

using System.Diagnostics;

namespace SeleniumCsharpPractice.Utilities

{

[TestClass]

public class ExcelToTextCompare

{

[TestMethod]

public void excelToTextCompareMethod()

{

string val = null;

string val1 = null;

string val2 = null;

string line = null;

int i = 1;

//Create COM Objects. Create a COM object for everything that is referenced

Excel.Application xlApp = new Excel.Application();

Excel.Workbook xlWorkbook = xlApp.Workbooks.Open(@"C:\Users\616181\Desktop\ExcelFile.xlsx");

Excel.\_Worksheet xlWorksheet = xlWorkbook.Sheets[1];

Excel.Range xlRange = xlWorksheet.UsedRange;

int rowCount = xlRange.Rows.Count;

int colCount = xlRange.Columns.Count;

// Read text file

System.IO.StreamReader file = new System.IO.StreamReader(@"C:\Users\616181\Desktop\TextFile.txt");

while ((line = file.ReadLine()) != null)

{

for (int j = 1; j <= colCount; j++)

{

if (xlRange.Cells[i, j] != null && xlRange.Cells[i, j].Value2 != null)

{

val = xlRange.Cells[i, j].Value2.ToString();

val1 = val.Trim();

Debug.WriteLine("val :" + "\t" + val);

string[] words = line.Split(' ');

val2 = words[j - 1].Trim();

Debug.WriteLine("val2 :" + "\t" + val2);

if (val1 != val2)

{

xlWorksheet.Cells[i, j].Interior.ColorIndex = 3;

}

}

}

i++;

}

//cleanup

GC.Collect();

GC.WaitForPendingFinalizers();

//rule of thumb for releasing com objects:

// never use two dots, all COM objects must be referenced and released individually

// ex: [somthing].[something].[something] is bad

//release com objects to fully kill excel process from running in the background

Marshal.ReleaseComObject(xlRange);

Marshal.ReleaseComObject(xlWorksheet);

// save the workbook

xlWorkbook.Save();

//close and release

xlWorkbook.Close();

Marshal.ReleaseComObject(xlWorkbook);

//quit and release

xlApp.Quit();

Marshal.ReleaseComObject(xlApp);

}

}

}

WriteToExcel:

====================================================

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.Diagnostics;

using System.Runtime.InteropServices;

using Excel = Microsoft.Office.Interop.Excel; //microsoft Excel 14 object in references-> COM tab

namespace SeleniumCsharpPractice.Utilities

{

[TestClass]

public class WriteToExcel

{

[TestMethod]

public void writeExcelFile(int row, int col, string val)

{

//Create COM Objects. Create a COM object for everything that is referenced

Excel.Application xlApp = new Excel.Application();

Excel.Workbook xlWorkbook = xlApp.Workbooks.Open(@"D:\Workspace\TestData\WebTable.xlsx");

Excel.\_Worksheet xlWorksheet = xlWorkbook.Sheets[1];

Excel.Range xlRange = xlWorksheet.UsedRange;

int rowCount = xlRange.Rows.Count;

int colCount = xlRange.Columns.Count;

//iterate over the rows and columns and print to the console as it appears in the file

//excel is not zero based!!

//for (int i = 1; i <= rowCount; i++)

//{

// for (int j = 1; j <= colCount; j++)

// {

// //new line

// if (j == 1)

// Debug.WriteLine("\r\n");

//write the value to the console

//if (xlWorksheet.Cells[row, col] != null && xlWorksheet.Cells[row, col].Value2 != null)

// {

// Console.Write(xlRange.Cells[row, col].Value2.ToString() + "\t");

xlWorksheet.Cells[row + 1, col + 1] = val;

// xlRange.Cells[row, col].Value2 = val;

// Debug.WriteLine("val :" + "\t" + val);

// }

// }

// return val;

//}

//return val;

// xlRange.Cells[4, 4].value = "baskar";

// xlRange.Cells[4, 4].Interior.ColorIndex = 3;

//cleanup

GC.Collect();

GC.WaitForPendingFinalizers();

//rule of thumb for releasing com objects:

// never use two dots, all COM objects must be referenced and released individually

// ex: [somthing].[something].[something] is bad

//release com objects to fully kill excel process from running in the background

Marshal.ReleaseComObject(xlRange);

Marshal.ReleaseComObject(xlWorksheet);

// save the workbook

xlWorkbook.Save();

//close and release

xlWorkbook.Close();

Marshal.ReleaseComObject(xlWorkbook);

//quit and release

xlApp.Quit();

Marshal.ReleaseComObject(xlApp);

}

}

}

SeleniumCsharpPractice:

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using OpenQA.Selenium.Chrome;

using OpenQA.Selenium;

using System.Diagnostics;

using OpenQA.Selenium.Support.UI;

using System.Threading;

using System.Collections.Generic;

using System.Collections.ObjectModel;

namespace SeleniumCsharpPractice.Utilities

{

[TestClass]

public class GoogleSearchTest

{

[TestMethod]

public void GoogleSearch()

{

for (int z = 1; z <= 2; z++)

{

// implicit wait, webdriver wait and excel read example

/\* IWebDriver driver = new ChromeDriver();

driver.Url = ("https://www.google.com");

Debug.WriteLine("chrome browser launched");

driver.Manage().Window.Maximize();

driver.Manage().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(30));

driver.Manage().Timeouts().SetPageLoadTimeout(TimeSpan.FromSeconds(60));

WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30));

IWebElement e = wait.Until(ExpectedConditions.ElementIsVisible(By.Name("q"))); \*/

ReadFromExcel re = new ReadFromExcel();

// e.SendKeys(re.readFromExcelMethod(z, 1));

Debug.WriteLine("val :" + re.readFromExcelMethod(z, 1));

Thread.Sleep(3000);

}

// excel read end

// frame example and excel write

// driver.Url = ("http://jqueryui.com/draggable/");

// driver.Manage().Window.Maximize();

// driver.Manage().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(30));

// driver.Manage().Timeouts().SetPageLoadTimeout(TimeSpan.FromSeconds(60));

// driver.FindElement(By.LinkText("Selectable")).Click();

// driver.SwitchTo().Frame(driver.FindElement(By.ClassName("demo-frame")));

// WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30));

// wait.Until(ExpectedConditions.ElementIsVisible(By.TagName("li")));

// IList<IWebElement> all = driver.FindElements(By.TagName("li"));

// Debug.WriteLine("total values : " + all.Count);

//// all[4].Click();

//// Thread.Sleep(3000);

// WriteToExcel we = new WriteToExcel();

// for (int i = 0; i < all.Count; i++)

// {

// Debug.WriteLine("value " + i + ": " + all[i].Text);

// we.writeExcelFile(5, i+1, all[i].Text);

// }

// //foreach (IWebElement links in all)

// //{

// // Debug.WriteLine("values " + links.Text);

// //}

// frame example end

// window example

//driver.Url = ("https://legacy.crystalcruises.com/login.aspx");

//driver.Manage().Window.Maximize();

//driver.Manage().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(30));

//driver.Manage().Timeouts().SetPageLoadTimeout(TimeSpan.FromSeconds(60));

//WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(60));

//wait.Until(ExpectedConditions.ElementToBeClickable(By.LinkText("GUEST CHECK-IN")));

////driver.FindElement(By.LinkText("Sign In")).Click();

//driver.FindElement(By.LinkText("GUEST CHECK-IN")).Click();

////wait for new window to open

//Thread.Sleep(2000);

////get the current window handles

//string popupHandle = string.Empty;

//ReadOnlyCollection<string> windowHandles = driver.WindowHandles;

//Debug.WriteLine("window size : " + windowHandles.Count);

//foreach (string handle in windowHandles)

//{

// // if (handle != existingWindowHandle)

// // {

// //switch to new window

// driver.SwitchTo().Window(handle);

// // }

//}

//driver.FindElement(By.Id("txtFirstName")).SendKeys("baskar");

//Thread.Sleep(3000);

//window end

// alert start

/\* driver.Url = ("http://irctc.co.in");

driver.Manage().Window.Maximize();

driver.Manage().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(30));

driver.Manage().Timeouts().SetPageLoadTimeout(TimeSpan.FromSeconds(60));

WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30));

wait.Until(ExpectedConditions.ElementToBeClickable(By.CssSelector("input[type='button'][value='Close']")));

driver.FindElement(By.CssSelector("input[type='button'][value='Close']")).Click();

driver.FindElement(By.Id("loginbutton")).Click();

IAlert a = driver.SwitchTo().Alert();

Debug.WriteLine("alert text : " + a.Text);

a.Accept(); \*/

// driver.Quit();

// }

}

}

}

TakeSnap:

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using OpenQA.Selenium.Chrome;

using OpenQA.Selenium;

using System.Diagnostics;

using OpenQA.Selenium.Support.UI;

using System.Threading;

using System.Collections.Generic;

using OpenQA.Selenium.Interactions;

using System.Linq;

namespace SeleniumCsharpPractice.Utilities

{

[TestClass]

public class TakeSnap

{

[TestMethod]

public void TestMethod1()

{

IWebDriver driver = new ChromeDriver();

driver.Url = ("http://api.checklist.com/register");

driver.Manage().Window.Maximize();

driver.Manage().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(30));

driver.Manage().Timeouts().SetPageLoadTimeout(TimeSpan.FromSeconds(60));

WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30));

IWebElement dl = driver.FindElement(By.Id("country"));

SelectElement s = new SelectElement(dl);

//s.SelectByIndex(3);

IList<IWebElement> li = s.Options;

li.ElementAt(4).Click();

IWebElement element = driver.FindElement(By.CssSelector("button[type='submit'][class='btn btn btn-primary pull-right']"));

//IJavaScriptExecutor executor = (IJavaScriptExecutor)driver;

//executor.ExecuteScript("arguments[0].click();", element);

IJavaScriptExecutor executor = (IJavaScriptExecutor)driver;

executor.ExecuteScript("arguments[0].scrollIntoView(true);", element);

//create Actions object

//Actions builder = new Actions(driver);

//create a chain of actions

//builder.ClickAndHold(li.ElementAt(5)).Build().Perform();

Screenshot ss = ((ITakesScreenshot)driver).GetScreenshot();

ss.SaveAsFile("D:\\Workspace\\TestData.Jpeg", System.Drawing.Imaging.ImageFormat.Jpeg);

}

}

}

Webtable:

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using OpenQA.Selenium.Chrome;

using OpenQA.Selenium;

using System.Diagnostics;

using OpenQA.Selenium.Support.UI;

using System.Threading;

using System.Collections.Generic;

namespace SeleniumCsharpPractice.Utilities

{

[TestClass]

public class Webtable

{

[TestMethod]

public void WebtableMethod()

{

IWebDriver driver = new ChromeDriver();

driver.Url = ("http://erail.in");

driver.Manage().Window.Maximize();

driver.Manage().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(30));

driver.Manage().Timeouts().SetPageLoadTimeout(TimeSpan.FromSeconds(60));

WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30));

wait.Until(ExpectedConditions.ElementToBeClickable(By.Id("chkSelectDateOnly")));

driver.FindElement(By.Id("chkSelectDateOnly")).Click();

Thread.Sleep(3000);

IWebElement wt = driver.FindElement(By.XPath("//\*[@id='divTrainsList']/table[1]"));

IList<IWebElement> row = wt.FindElements(By.TagName("tr"));

Debug.WriteLine("row :" + row.Count);

WriteToExcel we = new WriteToExcel();

for (int i = 0; i < row.Count; i++)

{

IList<IWebElement> col = row[i].FindElements(By.TagName("td"));

for (int j = 0; j < col.Count; j++)

{

Debug.WriteLine("col :" + col.Count);

Debug.WriteLine("val :" + col[j].Text.ToString());

we.writeExcelFile(i, j, col[j].Text.ToString());

// break;

}

// break;

}

}

}

}

MSTest:

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.Diagnostics;

namespace NUnitProject

{

[TestClass]

public class MSTest

{

[ClassInitialize]

public static void BeforeAll(TestContext context)

{

// run before all test cases

Debug.WriteLine("ClassInitialize");

}

[TestInitialize]

public void Before()

{

// run before each test case

Debug.WriteLine("TestInitialize");

}

[TestMethod]

public void TestCase1()

{

// one test case

Debug.WriteLine("TestMethod1");

}

[TestMethod]

public void TestCase2()

{

// another test case

Debug.WriteLine("TestMethod2");

}

[TestCleanup]

public void After()

{

// run after each test case

Debug.WriteLine("TestCleanup");

}

[ClassCleanup]

public static void AfterAll()

{

// run after all test cases, typically, close browser

Debug.WriteLine("ClassCleanup");

}

}

}

NUnitUnitTest:

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using OpenQA.Selenium;

using OpenQA.Selenium.Chrome;

using OpenQA.Selenium.Support.UI;

using System.Collections.Generic;

using System.Linq;

using NUnit.Framework;

using OpenQA.Selenium.IE;

namespace NUnitProject

{

[TestFixture]

[Parallelizable]

public class NUnitUnitTest

{

IWebDriver driver;

[SetUp]

public void SetUp()

{

driver = new ChromeDriver();

driver.Url = ("http://api.checklist.com/register");

Console.WriteLine("chrome browser launched");

}

[Test]

public void ExecuteTest()

{

driver.Manage().Window.Maximize();

driver.Manage().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(30));

driver.Manage().Timeouts().SetPageLoadTimeout(TimeSpan.FromSeconds(60));

WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30));

IWebElement dl = driver.FindElement(By.Id("country"));

SelectElement s = new SelectElement(dl);

//s.SelectByIndex(3);

IList<IWebElement> li = s.Options;

li.ElementAt(4).Click();

}

[TearDown]

public void EndTest()

{

Screenshot ss = ((ITakesScreenshot)driver).GetScreenshot();

ss.SaveAsFile("D:\\Workspace\\TestData.Jpeg", System.Drawing.Imaging.ImageFormat.Jpeg);

}

}

}

NUnitUnitTest1:

using System;

//using Microsoft.VisualStudio.TestTools.UnitTesting;

using OpenQA.Selenium;

using OpenQA.Selenium.Chrome;

using OpenQA.Selenium.Support.UI;

using System.Collections.Generic;

using System.Linq;

using NUnit.Framework;

using OpenQA.Selenium.IE;

namespace NUnitProject

{

[TestFixture]

[Parallelizable]

public class NUnitUnitTest1

{

IWebDriver driver;

[SetUp]

public void SetUp()

{

driver = new InternetExplorerDriver();

driver.Url = ("http://api.checklist.com/register");

Console.WriteLine("IE browser launched");

}

[Test]

// [Ignore("ignore test")]

[TestCase("country")]

public void ExecuteTest1(string value)

{

driver.Manage().Window.Maximize();

driver.Manage().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(30));

driver.Manage().Timeouts().SetPageLoadTimeout(TimeSpan.FromSeconds(60));

WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30));

IWebElement dl = driver.FindElement(By.Id(value));

SelectElement s = new SelectElement(dl);

//s.SelectByIndex(3);

IList<IWebElement> li = s.Options;

li.ElementAt(4).Click();

Assert.AreEqual(li.ElementAt(4).Text, "American Samoa");

}

[TearDown]

public void EndTest()

{

Screenshot ss = ((ITakesScreenshot)driver).GetScreenshot();

ss.SaveAsFile("D:\\Workspace\\TestData.Jpeg", System.Drawing.Imaging.ImageFormat.Jpeg);

}

}

}

System.IO – File Handling:

// A Hello World! program in C#.

using System;

using System.IO;

namespace HelloWorld

{

class Hello

{

public static void WriteFile(string Filename)

{

File.WriteAllText(Filename, "hello world");

}

public static void ReadFile(string Filename)

{

Console.WriteLine(File.ReadAllText(Filename));

}

public static void Main(string[] args)

{

string Filename = @"D:\Workspace\NUnitProject\test.xlsx";

WriteFile(Filename);

ReadFile(Filename);

Console.WriteLine("started");

}

}

}

ReadXMl:

DLL:

System.XML

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.Xml;

using System.Collections.Generic;

using NUnit.Framework;

using System;

namespace PioneerAutomation.E2S.EPEN.Automation.Utilities

{

[TestFixture]

public class ReadXML

{

[Test]

public List<string> readXMLMethod()

{

string DestinationPath = null;

string CallbackServiceUri = null;

string AuthService = null;

string TargetSystem = null;

List<string> xml = new List<string>();

{

// Start with XmlReader object

//here, we try to setup Stream between the XML file nad xmlReader

using (XmlReader reader = XmlReader.Create(@"D:\Workspace\PioneerAutomation\PioneerAutomation\E2S.EPEN.Automation.Datatables\ControlFile.xml"))

{

while (reader.Read())

{

if (reader.IsStartElement())

{

//return only when you have START tag

switch (reader.Name.ToString())

{

case "DestinationPath":

DestinationPath = reader.ReadString();

xml.Add(DestinationPath);

// Console.WriteLine("DestinationPath is : " + xml[0]);

break;

case "CallbackServiceUri":

CallbackServiceUri = reader.ReadString();

xml.Add(CallbackServiceUri);

// Console.WriteLine("CallbackServiceUri is : " + xml[1]);

break;

case "AuthService":

AuthService = reader.ReadString();

xml.Add(AuthService);

// Console.WriteLine("AuthService is : " + xml[2]);

break;

case "TargetSystem":

TargetSystem = reader.ReadString();

xml.Add(TargetSystem);

// Console.WriteLine("TargetSystem is : " + xml[3]);

break;

}

}

}

}

}

return xml;

}

}

}

ReadDatabase:

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.Collections.Generic;

using System.Data.SqlClient;

using NUnit.Framework;

using System;

namespace PioneerAutomation.E2S.EPEN.Automation.Utilities

{

[TestFixture]

public class ReadDatabase

{

[Test]

public List<string> ReadDatabaseMethod()

{

string DestinationPath = null;

string CallbackServiceUri = null;

string AuthService = null;

string TargetSystem = null;

List<string> db = new List<string>();

string queryString = "select u.description, i.uri from InstanceUri i join uritype u on i.uritypeid = u.uritypeid join paper p on i.instanceid = p.instanceid where p.importprojectid = '207002' and u.uritypeid in(12,13,14,15)";

string connectionString = "Data Source=LO3WTSCNSDB010;Initial Catalog=EpenConfig;User ID=DWS;Password=P0T8TO!;";

using (SqlConnection connection = new SqlConnection(connectionString))

{

SqlCommand command = new SqlCommand(queryString, connection);

connection.Open();

SqlDataReader reader = command.ExecuteReader();

try

{

while (reader.Read())

{

switch (reader.GetString(0))

{

case "DestinationPath":

DestinationPath = reader[1].ToString();

db.Add(DestinationPath);

// Console.WriteLine("DestinationPath is : " + DestinationPath);

break;

case "CallbackServiceUri":

CallbackServiceUri = reader[1].ToString();

db.Add(CallbackServiceUri);

// Console.WriteLine("CallbackServiceUri is : " + CallbackServiceUri);

break;

case "AuthService":

AuthService = reader[1].ToString();

db.Add(AuthService);

// Console.WriteLine("AuthService is : " + AuthService);

break;

case "TargetSystem":

TargetSystem = reader[1].ToString();

db.Add(TargetSystem);

// Console.WriteLine("TargetSystem is : " + TargetSystem);

break;

}

// Debug.WriteLine(String.Format("{0} \t | {1} \t | {2} \t | {3}",

// reader[0], reader[1], reader[2], reader[3]));

}

}

finally

{

reader.Close();

}

}

return db;

}

}

}

//select\* from instanceuri

//select\* from uritype

//select\* from Instance

//select\* from Paper where ImportProjectId ='207002'

//INSERT INTO UriType(Description) VALUES('DestinationPath');

//INSERT INTO UriType(Description) VALUES('CallbackServiceUri');

//INSERT INTO UriType(Description) VALUES('AuthService');

//INSERT INTO UriType(Description) VALUES('TargetSystem');

//INSERT INTO InstanceUri(instanceid, uri, uritypeid) VALUES(2,'\\LO3UDOP2FIL001\data1', 12);

//INSERT INTO InstanceUri(instanceid, uri, uritypeid) VALUES(2,'https://someserver/with/uk/api', 13);

//INSERT INTO InstanceUri(instanceid, uri, uritypeid) VALUES(2,'https://someserver/with/uk/auth/api', 14);

//INSERT INTO InstanceUri(instanceid, uri, uritypeid) VALUES(2,'DWSESIE', 15);

//select i.uritypeid, u.description, i.uri from InstanceUri i

//join uritype u on i.uritypeid = u.uritypeid

//join paper p on i.instanceid = p.instanceid where p.importprojectid= '207002' and u.uritypeid in(12,13,14,15)

RestSharp Get:

using System;

using PioneerAutomation.E2S.EPEN.Automation.Utilities;

using RestSharp;

using System.Net;

using NUnit.Framework;

namespace PioneerAutomation.E2S.EPEN.Automation.Test

{

[TestFixture]

[Parallelizable]

public class Oauth

{

string EndpointUrl = null;

string Username = null;

string Password = null;

string Grant\_type = null;

string ResponseCode = null;

string ResponseContent = null;

int rowcnt;

[Test]

public void oauthMethod()

{

string classname = this.GetType().Name;

// Console.WriteLine("classname : " + classname);

ExcelRowCount rowcount = new ExcelRowCount();

rowcnt = rowcount.getExcelRowCount(classname);

for (int row = 2; row <= rowcnt; row++)

{

ReadFromExcel readexcel = new ReadFromExcel();

EndpointUrl = readexcel.getValueFromExcel(row, 1, classname);

Username = readexcel.getValueFromExcel(row, 2, classname);

Password = readexcel.getValueFromExcel(row, 3, classname);

Grant\_type = readexcel.getValueFromExcel(row, 4, classname);

ResponseCode = readexcel.getValueFromExcel(row, 5, classname);

//Console.WriteLine("EndpointUrl : " + EndpointUrl);

//Console.WriteLine("Username : " + Username);

//Console.WriteLine("Password : " + Password);

//Console.WriteLine("Grant\_type : " + Grant\_type);

// RequestHeader = EndpointUrl + "?" + "username=" + Username + "password=" + Password + "grant\_type=" + Grant\_type;

// Debug.WriteLine("RequestHeader :" + RequestHeader);

var client = new RestClient(EndpointUrl);

var request = new RestRequest(Method.GET);

request.AddParameter("username", Username, ParameterType.QueryString);

request.AddParameter("password", Password, ParameterType.QueryString);

request.AddParameter("grant\_type", Grant\_type, ParameterType.QueryString);

request.RequestFormat = DataFormat.Json;

IRestResponse response = client.Execute(request);

// res = response.Headers.ElementAt(0).Value.ToString();

// Debug.WriteLine("Response : " + res);

HttpStatusCode statusCode = response.StatusCode;

int numericStatusCode = (int)statusCode;

WriteToExcel writeexcel = new WriteToExcel();

try

{

Assert.AreEqual(ResponseCode.ToString(), numericStatusCode.ToString());

writeexcel.writeValueToExcel(row, 7, "PASS", classname);

Console.WriteLine("ResponseCode is matched: " + numericStatusCode);

}

catch (Exception)

{

writeexcel.writeValueToExcel(row, 7, "FAIL", classname);

Console.WriteLine("ResponseCode is not matched: " + numericStatusCode);

}

ResponseContent = response.Content;

writeexcel.writeValueToExcel(row, 6, ResponseContent, classname);

// Debug.WriteLine("statusCode: " + numericStatusCode);

// Debug.WriteLine("Content: " + response.StatusCode);

// Debug.WriteLine("Content: " + response.Content);

}

}

}

}

RestSharp Post,put,Delete:

using System;

using PioneerAutomation.E2S.EPEN.Automation.Utilities;

using RestSharp;

using System.Net;

using NUnit.Framework;

namespace PioneerAutomation.E2S.EPEN.Automation.Scenarios

{

[TestFixture]

[Parallelizable]

public class PulledPaperAPIPost

{

string EndpointUrl = null;

string Header\_ContentType = null;

string Header\_XAuthToken = null;

string Jsonbody = null;

string ResponseCode = null;

string ResponseContent = null;

int rowcnt;

[Test]

public void PulledPaperAPIPosttMethod()

{

string classname = this.GetType().Name;

ExcelRowCount rowcount = new ExcelRowCount();

rowcnt = rowcount.getExcelRowCount(classname);

for (int row = 2; row <= rowcnt; row++)

{

ReadFromExcel readexcel = new ReadFromExcel();

EndpointUrl = readexcel.getValueFromExcel(row, 3, classname);

Header\_ContentType = readexcel.getValueFromExcel(row, 4, classname);

Header\_XAuthToken = readexcel.getValueFromExcel(row, 5, classname);

Jsonbody = readexcel.getValueFromExcel(row, 6, classname);

ResponseCode = readexcel.getValueFromExcel(row, 7, classname);

Console.WriteLine("EndpointUrl : " + EndpointUrl);

Console.WriteLine("Header\_ContentType : " + Header\_ContentType);

Console.WriteLine("Header\_XAuthToken : " + Header\_XAuthToken);

Console.WriteLine("Jsonbody : " + Jsonbody);

Console.WriteLine("ResponseCode : " + ResponseCode);

var client = new RestClient(EndpointUrl);

var request = new RestRequest(Method.POST);

request.AddHeader("Content-Type", Header\_ContentType);

request.AddHeader("X-Auth-Token", Header\_XAuthToken);

request.AddParameter("application/json", Jsonbody, ParameterType.RequestBody);

request.RequestFormat = DataFormat.Json;

IRestResponse response = client.Execute(request);

// res = response.Headers.ElementAt(0).Value.ToString();

// Debug.WriteLine("Response : " + res);

HttpStatusCode statusCode = response.StatusCode;

int numericStatusCode = (int)statusCode;

WriteToExcel writeexcel = new WriteToExcel();

try

{

Assert.AreEqual(ResponseCode.ToString(), numericStatusCode.ToString());

writeexcel.writeValueToExcel(row, 9, "PASS", classname);

Console.WriteLine("ResponseCode is matched: " + numericStatusCode);

}

catch (Exception)

{

writeexcel.writeValueToExcel(row, 9, "FAIL", classname);

Console.WriteLine("ResponseCode is not matched: " + numericStatusCode);

}

ResponseContent = response.Content;

Console.WriteLine("ResponseContent :" + ResponseContent);

writeexcel.writeValueToExcel(row, 8, ResponseContent, classname);

// Debug.WriteLine("statusCode: " + numericStatusCode);

// Debug.WriteLine("Content: " + response.StatusCode);

// Debug.WriteLine("Content: " + response.Content);

}

}

}

}

ControlFileValidation (Test Class)

/\* This class is call the ReadXML method to get the XML values and call the ReadDatabase method

\* to get the db values and compare both the values give the ressults.

\*

\* ============================================================================

\* Create By Create Date Modified Date Modification Description

\* Baskar S(616181) 03/09/2017 03/09/2017

\* ============================================================================

\*/

using System;

using PioneerAutomation.E2S.EPEN.Automation.Utilities;

using System.Collections.Generic;

using NUnit.Framework;

namespace PioneerAutomation.E2S.EPEN.Automation.Scenarios

{

[TestFixture]

[Parallelizable]

public class ControlFileValidation

{

[Test]

public void controlFileValidationMethod()

{

//XML values

// Console.Write("class name: " + this.GetType().Name);

string DestinationPathXML = null;

string CallbackServiceUriXML = null;

string AuthServiceXML = null;

string TargetSystemXML = null;

//DB values

string DestinationPathDB = null;

string CallbackServiceUriDB = null;

string AuthServiceDB = null;

string TargetSystemDB = null;

//method call readXMLMethod

ReadXML xml = new ReadXML();

List<string> xmlValue = xml.readXMLMethod();

DestinationPathXML = xmlValue[0];

CallbackServiceUriXML = xmlValue[1];

AuthServiceXML = xmlValue[2];

TargetSystemXML = xmlValue[3];

//Console.WriteLine("DestinationPathXML : " + DestinationPathXML);

//Console.WriteLine("CallbackServiceUriXML : " + CallbackServiceUriXML);

//Console.WriteLine("AuthServiceXML : " + AuthServiceXML);

//Console.WriteLine("TargetSystemXML : " + TargetSystemXML);

//method call ReadDatabaseMethod

ReadDatabase db = new ReadDatabase();

List<string> dbValue = db.ReadDatabaseMethod();

DestinationPathDB = dbValue[0];

CallbackServiceUriDB = dbValue[1];

AuthServiceDB = dbValue[2];

TargetSystemDB = dbValue[3];

//Console.WriteLine("DestinationPathDB : " + DestinationPathDB);

//Console.WriteLine("CallbackServiceUriDB : " + CallbackServiceUriDB);

//Console.WriteLine("AuthServiceDB : " + AuthServiceDB);

//Console.WriteLine("TargetSystemDB : " + TargetSystemDB);

try

{

Assert.AreEqual(DestinationPathXML, DestinationPathDB);

Console.WriteLine("Destination path is matched " + "Value from XML is : " + DestinationPathXML

+ " ||" + " Value from DB is : " + DestinationPathDB);

}

catch (Exception)

{

Console.WriteLine("Destination path is not matched " + " Value from XML is : " + DestinationPathXML

+ " ||" + " Value from DB is : " + DestinationPathDB);

}

try

{

Assert.AreEqual(CallbackServiceUriXML, CallbackServiceUriDB);

Console.WriteLine("CallbackServiceUri is matched " + "Value from XML is : " + CallbackServiceUriXML

+ " ||" + " Value from DB is : " + CallbackServiceUriDB);

}

catch (Exception)

{

Console.WriteLine("CallbackServiceUri is not matched " + " Value from XML is : " + CallbackServiceUriXML

+ " ||" + " Value from DB is : " + CallbackServiceUriDB);

}

try

{

Assert.AreEqual(AuthServiceXML, AuthServiceDB);

Console.WriteLine("AuthService is matched " + "Value from XML is : " + AuthServiceXML

+ " ||" + " Value from DB is : " + AuthServiceDB);

}

catch (Exception)

{

Console.WriteLine("AuthService is not matched " + " Value from XML is : " + AuthServiceXML

+ " ||" + " Value from DB is : " + AuthServiceDB);

}

try

{

Assert.AreEqual(TargetSystemXML, TargetSystemDB);

Console.WriteLine("TargetSystem is matched " + "Value from XML is : " + TargetSystemXML

+ " ||" + " Value from DB is : " + TargetSystemDB);

}

catch (Exception)

{

Console.WriteLine("TargetSystem is not matched " + " Value from XML is : " + TargetSystemXML

+ " ||" + " Value from DB is : " + TargetSystemDB);

}

}

}

}

Getter – Setter:

using System;

using System.Diagnostics;

using Microsoft.VisualStudio.TestTools.UnitTesting;

[TestClass]

public class Genre

{

public string Name { get; set; }

}

[TestClass]

public class MainClass

{

[TestMethod]

public void Print()

{

Genre g1 = new Genre();

Genre g2 = new Genre();

Genre g3 = new Genre();

g1.Name = "Hip Hop";

g2.Name = "Rock";

g3.Name = "Country";

Debug.WriteLine("Genres: {0}, {1}, {2}", g1.Name, g2.Name, g3.Name);

}

}

Utilities:

ReadFromExcel:

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.Runtime.InteropServices;

using Excel = Microsoft.Office.Interop.Excel; //microsoft Excel 14 object in references-> COM tab

using NUnit.Framework;

namespace PioneerAutomation.E2S.EPEN.Automation.Utilities

{

[TestFixture]

public class ReadFromExcel

{

String val = null;

[Test]

public String getValueFromExcel(int row, int col, string classname)

{

//Create COM Objects. Create a COM object for everything that is referenced

Excel.Application xlApp = new Excel.Application();

Excel.Workbook xlWorkbook = xlApp.Workbooks.Open(@"D:\Workspace\PioneerAutomation\PioneerAutomation\E2S.EPEN.Automation.Datatables\" + classname + ".xlsx");

Excel.\_Worksheet xlWorksheet = xlWorkbook.Sheets[1];

Excel.Range xlRange = xlWorksheet.UsedRange;

int rowCount = xlRange.Rows.Count;

int colCount = xlRange.Columns.Count;

//write the value to the console

if (xlRange.Cells[row, col] != null && xlRange.Cells[row, col].Value2() != null)

{

val = xlRange.Cells[row, col].Value2().ToString();

}

//cleanup

GC.Collect();

GC.WaitForPendingFinalizers();

//rule of thumb for releasing com objects:

// never use two dots, all COM objects must be referenced and released individually

// ex: [somthing].[something].[something] is bad

//release com objects to fully kill excel process from running in the background

Marshal.ReleaseComObject(xlRange);

Marshal.ReleaseComObject(xlWorksheet);

// save the workbook

xlWorkbook.Save();

//close and release

xlWorkbook.Close();

Marshal.ReleaseComObject(xlWorkbook);

//quit and release

xlApp.Quit();

Marshal.ReleaseComObject(xlApp);

return val;

}

}

}

WriteToExcel:

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.Diagnostics;

using System.Runtime.InteropServices;

using Excel = Microsoft.Office.Interop.Excel; //microsoft Excel 14 object in references-> COM tab

using NUnit.Framework;

namespace PioneerAutomation.E2S.EPEN.Automation.Utilities

{

[TestFixture]

public class WriteToExcel

{

[Test]

public void writeValueToExcel(int row, int col, string val,string classname)

{

//Create COM Objects. Create a COM object for everything that is referenced

Excel.Application xlApp = new Excel.Application();

Excel.Workbook xlWorkbook = xlApp.Workbooks.Open(@"D:\Workspace\PioneerAutomation\PioneerAutomation\E2S.EPEN.Automation.Datatables\" + classname + ".xlsx");

Excel.\_Worksheet xlWorksheet = xlWorkbook.Sheets[1];

Excel.Range xlRange = xlWorksheet.UsedRange;

xlWorksheet.Cells[row, col] = val;

if(val.Equals("PASS"))

{

xlWorksheet.Cells[row, col].Interior.ColorIndex = 4;

}

else if (val.Equals("FAIL"))

{

xlWorksheet.Cells[row, col].Interior.ColorIndex = 3;

}

GC.Collect();

GC.WaitForPendingFinalizers();

//rule of thumb for releasing com objects:

// never use two dots, all COM objects must be referenced and released individually

// ex: [somthing].[something].[something] is bad

//release com objects to fully kill excel process from running in the background

Marshal.ReleaseComObject(xlRange);

Marshal.ReleaseComObject(xlWorksheet);

// save the workbook

xlWorkbook.Save();

//close and release

xlWorkbook.Close();

Marshal.ReleaseComObject(xlWorkbook);

//quit and release

xlApp.Quit();

Marshal.ReleaseComObject(xlApp);

}

}

}

ExcelRowCount:

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.Runtime.InteropServices;

using Excel = Microsoft.Office.Interop.Excel; //microsoft Excel 14 object in references-> COM tab

using NUnit.Framework;

namespace PioneerAutomation.E2S.EPEN.Automation.Utilities

{

[TestFixture]

public class ExcelRowCount

{

int rowCount;

[Test]

public int getExcelRowCount(string classname)

{

//Create COM Objects. Create a COM object for everything that is referenced

Excel.Application xlApp = new Excel.Application();

Excel.Workbook xlWorkbook = xlApp.Workbooks.Open(@"D:\Workspace\PioneerAutomation\PioneerAutomation\E2S.EPEN.Automation.Datatables\" + classname + ".xlsx");

Excel.\_Worksheet xlWorksheet = xlWorkbook.Sheets[1];

Excel.Range xlRange = xlWorksheet.UsedRange;

rowCount = xlRange.Rows.Count;

//cleanup

GC.Collect();

GC.WaitForPendingFinalizers();

//rule of thumb for releasing com objects:

// never use two dots, all COM objects must be referenced and released individually

// ex: [somthing].[something].[something] is bad

//release com objects to fully kill excel process from running in the background

Marshal.ReleaseComObject(xlRange);

Marshal.ReleaseComObject(xlWorksheet);

// save the workbook

xlWorkbook.Save();

//close and release

xlWorkbook.Close();

Marshal.ReleaseComObject(xlWorkbook);

//quit and release

xlApp.Quit();

Marshal.ReleaseComObject(xlApp);

return rowCount;

}

}

}

Windows Service start,stop:

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.ServiceProcess;

using NUnit.Framework;

namespace PioneerWebServiceAutomation.E2S.EPEN.Automation.Utilities

{

[TestFixture]

public class WindowsService

{

[Test]

public void TestMethod1()

{

ServiceController service = new ServiceController("Jenkins");

try

{

TimeSpan timeout = TimeSpan.FromMilliseconds(1000);

if (service.Status == ServiceControllerStatus.Running)

service.Stop();

if (service.Status == ServiceControllerStatus.Stopped)

service.Start();

Console.WriteLine("started");

service.WaitForStatus(ServiceControllerStatus.Running, timeout);

}

catch

{

Console.WriteLine("error");

}

//ServiceController sc = new ServiceController("Epen Services", "10.166.149.30");

//sc.Start();

}

}

}

Service example:

Note: we need to add WSDL/SVC/XML as service reference

using System;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Configuration;

using Framework\_Utilities;

using CRAFT.SupportLibraries;

using Amazon.S3.Model;

using Amazon.S3;

using Amazon.S3.IO;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using CRAFT.businesscomponents.DBValidation;

using Framework\_Reporting;

using CRAFT.componentgroups;

namespace CRAFT.businesscomponents.Services.ConfigServices

{

class ConfigService\_GetInstanceUriList\_Validation : ReusableLibrary

{

public ConfigService\_GetInstanceUriList\_Validation(ScriptHelper scriptHelper)

: base(scriptHelper)

{

}

public void ConfigService\_GetInstanceUriList\_Validation\_Method()

{

string ImportProjectId = DataTable.GetData("General\_Data", "ImportProjectId");

int[] intarr = new int[23];

intarr[0] = int.Parse(DataTable.GetData("General\_Data", "iUriType1"));

intarr[1] = int.Parse(DataTable.GetData("General\_Data", "iUriType2"));

intarr[2] = int.Parse(DataTable.GetData("General\_Data", "iUriType3"));

intarr[3] = int.Parse(DataTable.GetData("General\_Data", "iUriType4"));

intarr[4] = int.Parse(DataTable.GetData("General\_Data", "iUriType5"));

intarr[5] = int.Parse(DataTable.GetData("General\_Data", "iUriType6"));

intarr[6] = int.Parse(DataTable.GetData("General\_Data", "iUriType7"));

intarr[7] = int.Parse(DataTable.GetData("General\_Data", "iUriType8"));

intarr[8] = int.Parse(DataTable.GetData("General\_Data", "iUriType9"));

intarr[9] = int.Parse(DataTable.GetData("General\_Data", "iUriType10"));

intarr[10]= int.Parse(DataTable.GetData("General\_Data", "iUriType12"));

intarr[11] = int.Parse(DataTable.GetData("General\_Data", "iUriType13"));

intarr[12] = int.Parse(DataTable.GetData("General\_Data", "iUriType14"));

intarr[13] = int.Parse(DataTable.GetData("General\_Data", "iUriType15"));

intarr[14] = int.Parse(DataTable.GetData("General\_Data", "iUriType16"));

intarr[15] = int.Parse(DataTable.GetData("General\_Data", "iUriType17"));

intarr[16] = int.Parse(DataTable.GetData("General\_Data", "iUriType18"));

intarr[17] = int.Parse(DataTable.GetData("General\_Data", "iUriType24"));

intarr[18] = int.Parse(DataTable.GetData("General\_Data", "iUriType25"));

intarr[19] = int.Parse(DataTable.GetData("General\_Data", "iUriType26"));

intarr[20] = int.Parse(DataTable.GetData("General\_Data", "iUriType27"));

intarr[21] = int.Parse(DataTable.GetData("General\_Data", "iUriType28"));

intarr[22] = int.Parse(DataTable.GetData("General\_Data", "iUriType30"));

string uri = string.Empty;

string uridescription = string.Empty;

string uritypeid = string.Empty;

Report.UpdateTestLog("DB Connection Validation", "Fetch the instanceurilist from Epenconfig DB", Status.DONE);

//DB Connection and fetch the instanceurilist from EpenConfig

String query = @"select i.uritypeid, u.description, i.uri from InstanceUri i

join uritype u on i.uritypeid = u.uritypeid

join paper p on i.instanceid = p.instanceid where p.importprojectid = " + ImportProjectId + " and u.uritypeid in(1,2,3,4,5,6,7,8,9,10,12,13,14,15,16,17,18,24,25,26,27,28,30)";

List<String[]> dbRows = new BackendValidation().ExecuteQuery(query, 3, "EpenConfig");

Report.UpdateTestLog("DB Connected", "Instanceurilist is fetched from Epenconfig DB", Status.PASS);

for (int i = 0; i < dbRows.Count; i++)

{

if (dbRows.Count >= 0)

{

uritypeid = dbRows[i][0];

uridescription = dbRows[i][1];

uri = dbRows[i][2];

ConfigService.ConfigServiceClient client = new ConfigService.ConfigServiceClient();

ConfigService.InstanceUri[] resarr = client.GetInstanceUriList(ImportProjectId, intarr);

try

{

Assert.AreEqual(uri, resarr[i].Uri);

{

Console.WriteLine("uri is matched and the value is : " + resarr[i].Uri);

Report.UpdateTestLog("uri validation", "uri is matched and the value from response is : " + resarr[i].Uri + " and the value from db is : " + uri, Status.PASS);

}

}

catch (Exception)

{

Report.UpdateTestLog("uri validation", "uri is not matched and the value from response is : " + resarr[i].Uri + " and the value from db is : " + uri, Status.FAIL);

}

try

{

Assert.AreEqual(uridescription, resarr[i].UriDescription);

{

Console.WriteLine("UriDescription is matched and the value is : " + resarr[i].UriDescription);

Report.UpdateTestLog("UriDescription validation", "UriDescription is matched and the value from response is : " + resarr[i].UriDescription + " and the value from db is : " + uridescription, Status.PASS);

}

}

catch (Exception)

{

Report.UpdateTestLog("UriDescription validation", "UriDescription is not matched and the value from response is : " + resarr[i].UriDescription + " and the value from db is : " + uridescription, Status.FAIL);

}

try

{

Assert.AreEqual(uritypeid, resarr[i].UriTypeId.ToString());

{

Console.WriteLine("UriTypeId is matched and the value is : " + resarr[i].UriTypeId.ToString());

Report.UpdateTestLog("UriTypeId validation", "UriTypeId is matched and the value from response is : " + resarr[i].UriTypeId.ToString() + " and the value from db is : " + uritypeid, Status.PASS);

}

}

catch (Exception)

{

Report.UpdateTestLog("UriTypeId validation", "UriTypeId is not matched and the value from response is : " + resarr[i].UriTypeId.ToString() + " and the value from db is : " + uritypeid, Status.FAIL);

}

}

else

{

Report.UpdateTestLog("Epen Config DB", "Instanceurilist is not fetched from Epenconfig DB", Status.FAIL);

}

}

}

}

}

Service Example 2:

using System;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Configuration;

using Framework\_Utilities;

using CRAFT.SupportLibraries;

using Amazon.S3.Model;

using Amazon.S3;

using Amazon.S3.IO;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using CRAFT.businesscomponents.DBValidation;

using CRAFT.ResponseService;

using Framework\_Reporting;

namespace CRAFT.businesscomponents.Services.ResponseServices

{

public class ResponseService\_AddResponse\_Validation : ReusableLibrary

{

public ResponseService\_AddResponse\_Validation(ScriptHelper scriptHelper)

: base(scriptHelper)

{

}

public void ResponseService\_AddResponse\_Validation\_Method()

{

Report.UpdateTestLog("Addresponse execution validation", "Addresponse execution started", Status.PASS);

string importProjectId = DataTable.GetData("General\_Data", "ImportProjectId");

string ResponseIdentity1 = DataTable.GetData("General\_Data", "ResponseIdentity1");

string ResponseIdentity2 = DataTable.GetData("General\_Data", "ResponseIdentity2");

#region Data

var myResponseList = new List<ResponseContract>

{

new ResponseContract

{

ExaminationBarcode = "RU1234567",

ExaminationTypeIdentity =importProjectId,

ExaminationIdentity = "0248100252154",

Name = "",

ResponseIdentity = ResponseIdentity1,

ExaminationBatchIdentity = "33344400",

ScoringSystemBatchIdentity ="DWS|12345670",

ResponseMedias = new List<MediaContract>

{

new MediaContract{MediaLocation =@"\TestFolder\Test2.tiff", MediaSize=222222, MediaType="TIFF", MediaClipName="6GP01\_01\_Q02x\_P2"},

new MediaContract{MediaLocation =@"\TestFolder\Test3.tiff", MediaSize=333333, MediaType="TIFF", MediaClipName="6GP01\_01\_Q02x\_P3"},

new MediaContract{MediaLocation =@"\TestFolder\Test4.tiff", MediaSize=444444, MediaType="TIFF", MediaClipName="6GP01\_01\_Q02x\_P4"},

new MediaContract{MediaLocation =@"\TestFolder\Test5.tiff", MediaSize=555555, MediaType="TIFF", MediaClipName="6GP01\_01\_Q02x\_P5"},

new MediaContract{MediaLocation =@"\TestFolder\Test6.tiff", MediaSize=666666, MediaType="TIFF", MediaClipName="6GP01\_01\_Q02x\_P6"},

new MediaContract{MediaLocation =@"\TestFolder\Test7.tiff", MediaSize=777777, MediaType="TIFF", MediaClipName="6GP01\_01\_Q02x\_P7"},

new MediaContract{MediaLocation =@"\TestFolder\Test8.tiff", MediaSize=888888, MediaType="TIFF", MediaClipName="6GP01\_01\_Q02x\_P8"},

new MediaContract{MediaLocation =@"\TestFolder\Test9.tiff", MediaSize=999999, MediaType="TIFF", MediaClipName="6GP01\_01\_Q02x\_P9"},

new MediaContract{MediaLocation =@"\TestFolder\Test10.tiff", MediaSize=000000, MediaType="TIFF", MediaClipName="6GP01\_01\_Q02x\_P10"}

}

},

new ResponseContract

{

ExaminationBarcode = "RU1234567",

ExaminationTypeIdentity =importProjectId,

ExaminationIdentity = "0248100252154",

Name = "",

ResponseIdentity = ResponseIdentity2,

ExaminationBatchIdentity = "33344400",

ScoringSystemBatchIdentity ="DWS|12345670",

ResponseMedias = new List<MediaContract>

{

new MediaContract{MediaLocation =@"\TestFolder\Test1.tiff", MediaSize=111111, MediaType="TIFF", MediaClipName="6GP01\_01\_Q04y\_P1"},

new MediaContract{MediaLocation =@"\TestFolder\Test2.tiff", MediaSize=222222, MediaType="TIFF", MediaClipName="6GP01\_01\_Q04y\_P2"},

new MediaContract{MediaLocation =@"\TestFolder\Test3.tiff", MediaSize=333333, MediaType="TIFF", MediaClipName="6GP01\_01\_Q04y\_P3"},

new MediaContract{MediaLocation =@"\TestFolder\Test4.tiff", MediaSize=444444, MediaType="TIFF", MediaClipName="6GP01\_01\_Q04y\_P4"},

new MediaContract{MediaLocation =@"\TestFolder\Test5.tiff", MediaSize=555555, MediaType="TIFF", MediaClipName="6GP01\_01\_Q04y\_P5"},

new MediaContract{MediaLocation =@"\TestFolder\Test6.tiff", MediaSize=666666, MediaType="TIFF", MediaClipName="6GP01\_01\_Q04y\_P6"},

new MediaContract{MediaLocation =@"\TestFolder\Test7.tiff", MediaSize=777777, MediaType="TIFF", MediaClipName="6GP01\_01\_Q04y\_P7"},

new MediaContract{MediaLocation =@"\TestFolder\Test8.tiff", MediaSize=888888, MediaType="TIFF", MediaClipName="6GP01\_01\_Q04y\_P8"},

new MediaContract{MediaLocation =@"\TestFolder\Test9.tiff", MediaSize=999999, MediaType="TIFF", MediaClipName="6GP01\_01\_Q04y\_P9"},

new MediaContract{MediaLocation =@"\TestFolder\Test10.tiff", MediaSize=000000, MediaType="TIFF", MediaClipName="6GP01\_01\_Q04y\_P10"}

}

}

};

#endregion

var myResponseService = new ResponseService.ResponseServiceClient();

var myAddResponseRequest = new AddResponsesRequest

{

ImportProjectId = importProjectId,

Responses = myResponseList

};

var responseResponse = myResponseService.AddResponses(myAddResponseRequest);

string ICFBatchFilename = responseResponse.BatchFilename;

List<ResponseContract> res = responseResponse.ResponsesOutcome;

bool success = res[0].Success;

if (success == true)

{

Report.UpdateTestLog("Addresponse response validation", "Addresponse executed successfully and cdf&icf file moved to aws s3successfully and file name is : " + ICFBatchFilename, Status.PASS);

CDF\_ICF\_AWS\_Validation cdficf = new CDF\_ICF\_AWS\_Validation(ScriptHelper);

string CDFBatchFilename = ICFBatchFilename.Substring(0,14);

cdficf.CDF\_AWS\_Success\_Validation\_Method(CDFBatchFilename);

cdficf.ICF\_AWS\_Success\_Validation\_Method(ICFBatchFilename);

}

else

{

Report.UpdateTestLog("Addresponse response validation", "Addresponse not executed successfully and cdf&icf file not moved to aws s3", Status.FAIL);

DBValidationComponents dbc = new DBValidationComponents(ScriptHelper);

dbc.ImageTransferFailureValidation();

}

}

}

}

AWS example:

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using PioneerAutomation.ResponseService;

using Amazon.S3.Model;

using Amazon.S3;

using Amazon.S3.IO;

using NUnit.Framework;

using System.Collections.Generic;

using System.Linq;

namespace PioneerAutomation.E2S.EPEN.Automation.Modules.E2S.EPEN.Automation.Pioneer.Modules

{

[TestFixture]

public class CDF\_ICF\_AWS\_Validation\_Original

{

IAmazonS3 s3Client = null;

[Test]

public void CDF\_ICF\_AWS\_Validation()

{

int cnt = 0;

try

{

string awsAccessKeyID = "AKIAIWFRUQTVR2R7V6NQ";

string awsSecretAccessKey = "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";

s3Client = new AmazonS3Client(awsAccessKeyID, awsSecretAccessKey, Amazon.RegionEndpoint.EUWest2);

//Validate bucket present in S3 (AWS).

S3Bucket bucket = s3Client.ListBuckets().Buckets.Where(b => b.BucketName == "epen2-uk-dev-lambdacode").Single();

if (bucket != null)

{

Console.WriteLine("bucket 'epen2-uk-dev-lambdacode' is exist in AWS S3");

//Check CDF path present in S3 (AWS) Bucket

S3DirectoryInfo s3Root = new S3DirectoryInfo(s3Client, bucket.BucketName);

S3DirectoryInfo cdf\_icf\_dir = s3Root.GetDirectory("data1\\CDF\\E9999");

if (cdf\_icf\_dir.Exists)

{

Console.WriteLine("CDF/ICF Directory exist and directory name is : " + cdf\_icf\_dir.FullName.ToString());

foreach (var file in cdf\_icf\_dir.GetFiles())

{

// Console.WriteLine("File name is : " + file.Name.ToString());

// Console.WriteLine("File extension is : " + file.Extension.ToString());

cnt = cnt + 1;

}

Console.WriteLine("File count is : " + cnt);

}

else

{

Console.WriteLine("CDF/ICF Directory not exist");

}

}

else

{

Console.WriteLine("Bucket 'epen2-uk-dev-lambdacode' not exist in AWS S3");

}

}

catch (AmazonS3Exception amazonS3Exception)

{

// Catch AWS exception.

if (amazonS3Exception.ErrorCode != null && (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId") ||

amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))

{

Console.WriteLine("Not able to connect AWS S3");

}

}

// return cnt;

}

}

}

AWS read all directory from specified bucket:

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using PioneerAutomation.ResponseService;

using Amazon.S3.Model;

using Amazon.S3;

using Amazon.S3.IO;

using NUnit.Framework;

using System.Collections.Generic;

using System.Linq;

namespace PioneerAutomation.E2S.EPEN.Automation.Modules.E2S.EPEN.Automation.Pioneer.Modules

{

[TestFixture]

public class CDF\_ICF\_AWS\_Validation

{

IAmazonS3 s3Client = null;

[Test]

public void CDF\_ICF\_AWS\_Validation\_Method()

{

string awsAccessKeyID = "AKIAIWFRUQTVR2R7V6NQ";

string awsSecretAccessKey = "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";

s3Client = new AmazonS3Client(awsAccessKeyID, awsSecretAccessKey, Amazon.RegionEndpoint.EUWest2);

// S3Bucket bucket = s3Client.ListBuckets().Buckets.Where(b => b.BucketName == "epen2-uk-dev-lambdacode").Single();

// if (bucket != null)

// {

// Console.WriteLine("Bucket Exists");

// }

// List all objects

ListObjectsRequest listRequest = new ListObjectsRequest

{

BucketName = "epen2-uk-dev-lambdacode",

};

ListObjectsResponse listResponse;

do

{

// Get a list of objects

listResponse = s3Client.ListObjects(listRequest);

foreach (S3Object obj in listResponse.S3Objects)

{

Console.WriteLine("Object - " + obj.Key);

Console.WriteLine(" Size - " + obj.Size);

Console.WriteLine(" LastModified - " + obj.LastModified);

Console.WriteLine(" Storage class - " + obj.StorageClass);

}

// Set the marker property

listRequest.Marker = listResponse.NextMarker;

} while (listResponse.IsTruncated);

}

}

}

MSSQL db connection:

public List<string[]> ExecuteQuery(string query, int columnsCount,String initialCatalog)

{

string dbDataSource = ConfigurationManager.AppSettings["DB.DataSource"];

string dbInitialCatalog = initialCatalog; //ConfigurationManager.AppSettings["DB.InitialCatalog"];

string dbUserId = ConfigurationManager.AppSettings["DB.UserId"];

string dbEncrptedPassword = ConfigurationManager.AppSettings["DB.EncrptedPassword"];

string connectionString = "Data Source=" + dbDataSource + ";Initial Catalog=" + dbInitialCatalog + ";User=" +

dbUserId + ";Password=" + dbEncrptedPassword;

using (SqlConnection sqlCon = new SqlConnection(connectionString))

{

sqlCon.Open();

SqlCommand command = new SqlCommand(query, sqlCon);

SqlDataReader reader = command.ExecuteReader();

try

{

List<string[]> rowList = new List<string[]>();

while (reader.Read())

{

string[] row = new String[columnsCount];

for (int i = 0; i < columnsCount; i++)

{

row[i] = reader.GetValue(i).ToString();

}

rowList.Add(row);

}

return rowList;

}

catch (Exception ex)

{

throw new FrameworkException("Exception SQL Server Query Execution: " + ex.Message);

}

finally

{

if (reader != null)

{

reader.Close();

}

if (command != null)

{

sqlCon.Close();

}

sqlCon.Close();

}

}

}