内容

はじめに 1

1. Selenium の準備 1

1.1. 動作環境の構成 1

1.2. 必要なツール／ライブラリ 2

2. Selenium IDE の操作 7

3. コンパイル 10

4. Selenium の実行 14

5. 項目追加・変更対応（FindElement） 17

6. ブラウザとの同期（Wait、Sleep） 19

7. 画面の切り替え（WidowHandles） 22

8. アラート（AlertIsPresent）、モーダル画面 23

9. キャプチャ（GetScreenshot） 24

10. ダウンロード（AutoIt） 25

11. config から入力値の設定 27

12. 実行時のログ参照 29

13. 試験／診断結果の確認 30

13.1 実行結果の確認 30

13.2 Assert の組込み 31

14. 電源投入と停止・進行管理 32

試験自動化(Selenium,AutoIt,電源)

はじめに

Selenium はオペレータのブラウザに対する操作をシミュレートするツールです。

以下の 3 点がブラウザの操作を行う目的の主なものです。

① Web スクレイピング　…目的サイトの html から情報を抜き取り収集します

② AP テストの自動化　…Jenkins 等と組合わせて CI(continuous integration)環境を構築

③ 運用環境の診断　…画面操作を通して運用サーバ、通信経路の生死・性能診断

このうち、①の Web スクレイピングについては RPA(Robotic Process Automation)でも情報収集の目玉機能になっていますが、情報収集を禁止しているサイトもあるため注意が必要です。

ここでは、②、③を主目的とした Selenium の実装方法と Selenium を補完するためのツールとして AutoIt、自動運転のための自動電源投入とシャットダウンの方法を説明します。

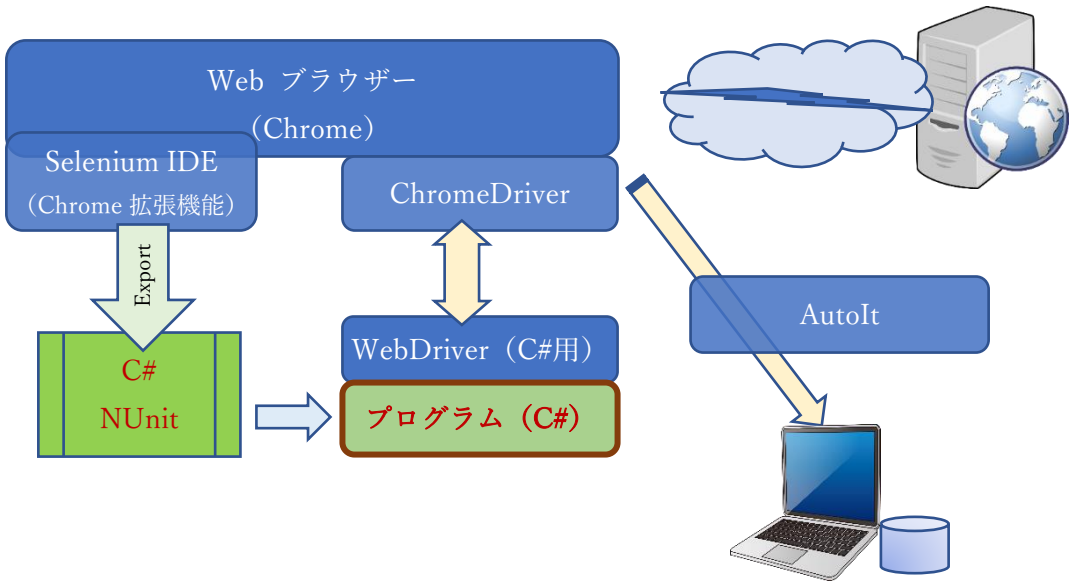
1. Selenium の準備

Selenium（関連プロジェクト）にはいくつかの実行形態があります。

以下では Chrome の拡張機能である Selenium IDE と Selenium WebDriver を組み合わせて環境を作ります。まず、Selenium IDE で実際に Chrome を操作した手順（テストケース）を C#で記録し、テストケースを編集して Selenium WebDriver で実行できるようにします。

1.1. 動作環境の構成

各資材の構成／関連は下図のとおり。



1.2. 必要なツール／ライブラリ

（1） Selenium 関連

Selenium の各種プロジェクトは下記 URL から辿ってダウンロードできます。

https://www.selenium.dev/projects/

① Selenium IDE

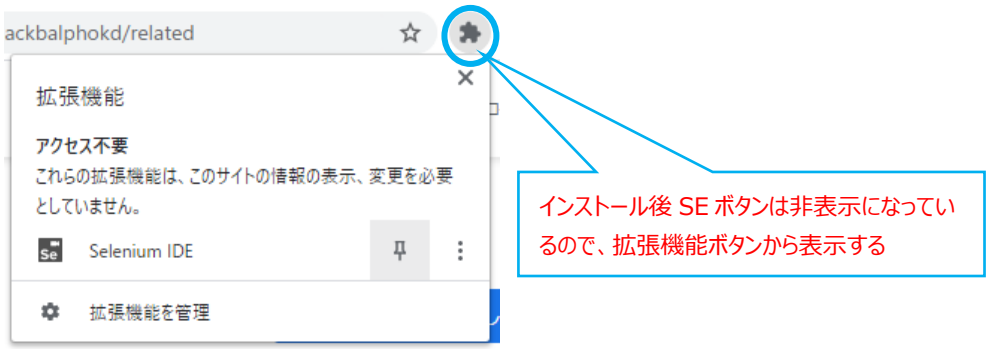
上記 URL の Selenium IDE > [Learn More]リンク押下

⇒[CHROME DOWNLOAD]のボタンを押下

⇒chrome ウェブストアに移って[Chrome に追加]を押下

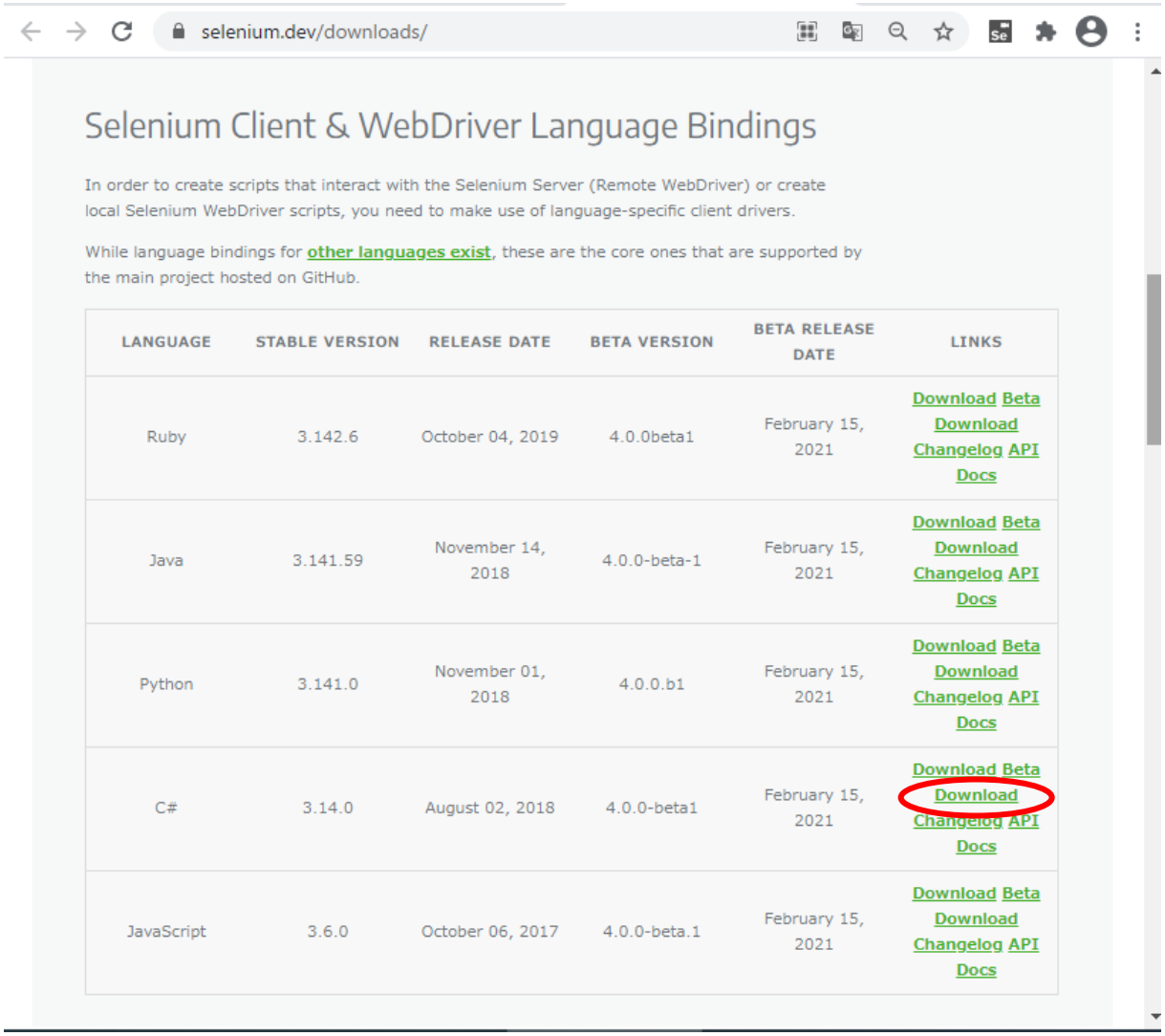
⇒確認のダイアログで[拡張機能を追加]ボタンを押下

※“SE”のアイコンがでたらインストール完了です。同期を有効にする必要はありません



② Selenium WebDriver

Selenium WebDriver はブラウザ（IE かそれ以外）とプログラミング言語で別物になります。
ここでは、C#の STABLE 版を選びます。



ダウンロードした selenium-dotnet-3.14.0.zip には以下の資材が入っています。

```
dist ┬㋐Selenium.Support.3.14.0.nupkg[1]
     ├㋑Selenium.WebDriver.3.14.0.nupkg
     └Selenium.WebDriverBackedSelenium.3.14.0.nupkg＜不使用＞
```

【必要資材の取り出し】

拡張子の”nupkg”を”zip”に変えて解凍し、以下のファイルを取り出します。

㋐ Selenium.Support.3.14.0.zip/lib/net45/WebDriver.Support.dll

㋑ Selenium.WebDriver.3.14.0.zip//lib/net45/WebDriver.dll

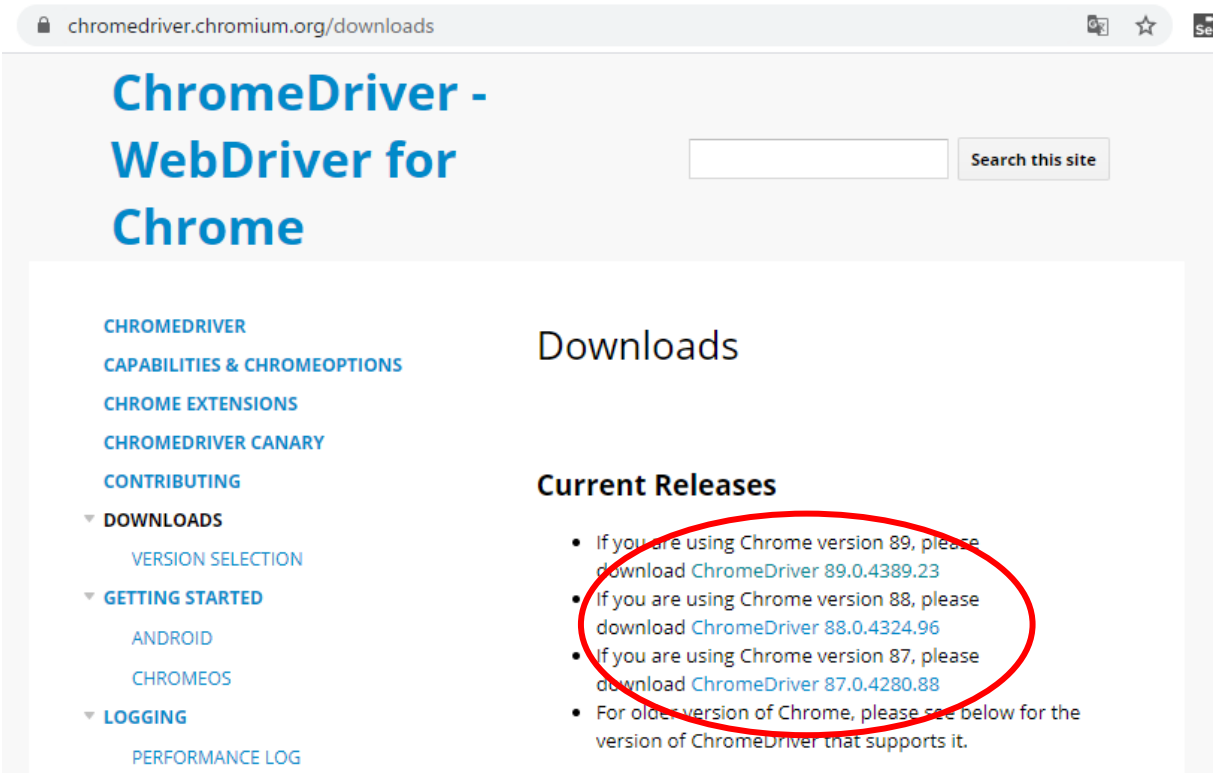
※㋐はコンパイルに㋑はコンパイルと実行時に使います（後述）

[1] nupkg: .NET 用のパッケージ(NuGet)で zip 形式と互換がある

③ chromedriver.exe

WebDriver.dll から起動されて実際に Chrome を制御するのが chromedriver.exe です。下記 URL より操作対象とする Crome のバージョンに合うものをダウンロードします。

http://chromedriver.chromium.org/downloads



解凍してできる chromedriver.exe は独立したアプリケーションとして環境変数の path のなかから探して起動されます。(後述)

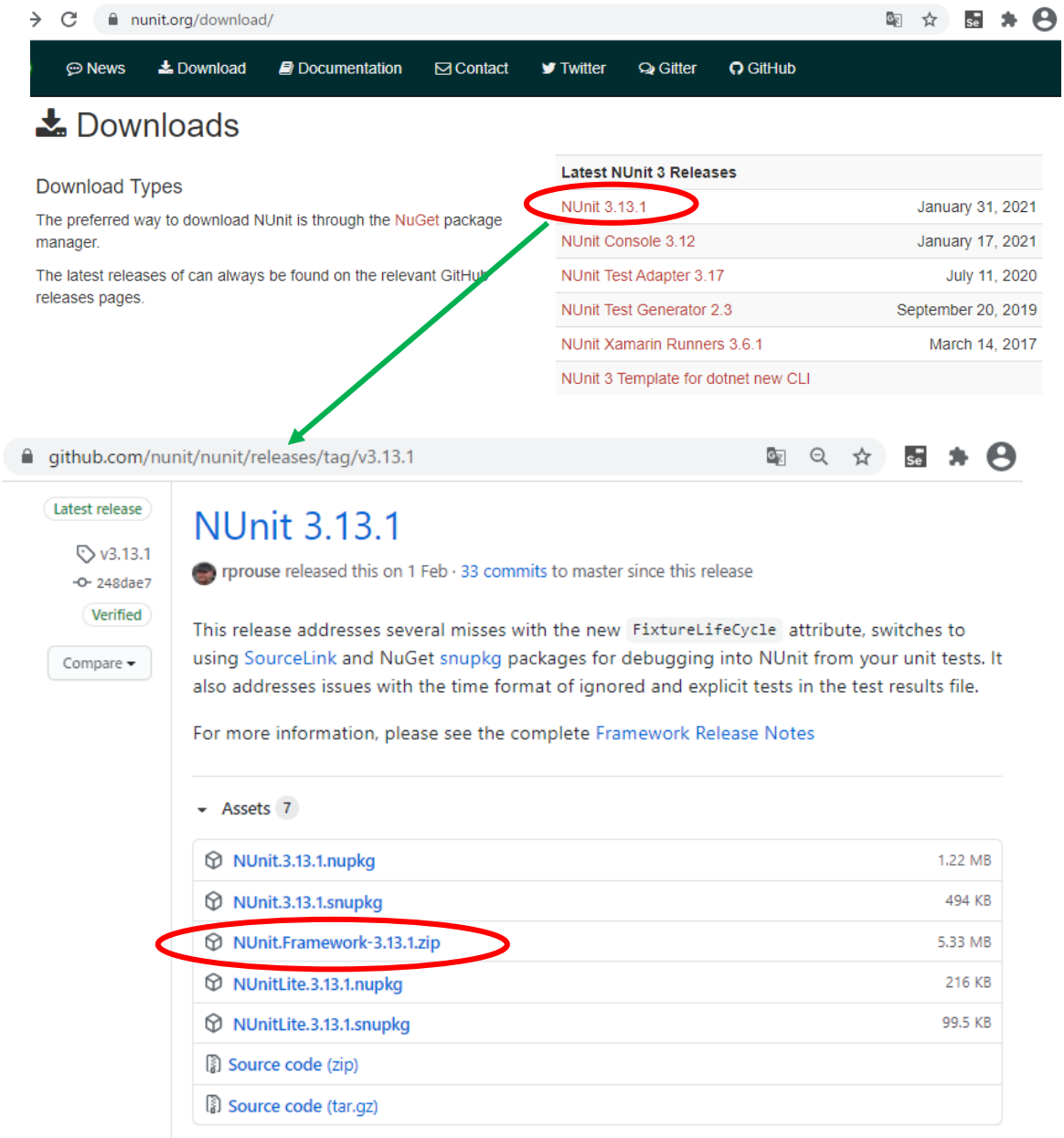
（２） NUnit

NUnit は.NET 用の単体テストツールです。

Selenium IDE は記録した操作を各種言語に Export できますが、どれも単体テストツールの利用を前提として出力されます。ここでは C# ＋ NUnit の組合せを使います。

下記 URL から辿ってダウンロードします。

https://nunit.org/download/



NUnit.Framework-3.13.1.zip を解凍し、bin/net45 をフォルダごと使います。

（３） AutoIt

AutoIt の Introduction には以下のように書かれています（Google Translate による和訳)。

『AutoIt v3 は、WindowsGUI と一般的なスクリプトを自動化するために設計されたフリーウェアの BASIC のようなスクリプト言語です。』

Selenium がブラウザの制御に限定されるのに対して WindowsGUI 一般の自動応答を行うことができ、ブラウザが開いた OS 組込みのダイアログに対する応答等が可能になります。

下記 URL からダウンロード。

https://www.autoitscript.com/site/autoit/

（４） C#コンパイラ

C#の開発は一般に Visual Studio が使われますが、ここでは Selenium IDE から Export したソースを基にするためコード作成の負荷が小さいので Visual Studio は使わずコマンドラインからコンパイラを起動します。

【コンパイラのバージョン】

インストール済の CSC を dir コマンドで確認（32 ビット版）

C:¥Users¥User>**dir %WINDIR%¥Microsoft.NET¥Framework¥csc.exe /s/b**
C:¥WINDOWS¥Microsoft.NET¥Framework¥v2.0.50727¥csc.exe
C:¥WINDOWS¥Microsoft.NET¥Framework¥v3.5¥csc.exe
C:¥WINDOWS¥Microsoft.NET¥Framework¥v4.0.30319¥csc.exe ←コレを使う

※64 ビット版もあります[2]が、32 ビットの v4.0.30319¥csc.exe を使います[3]

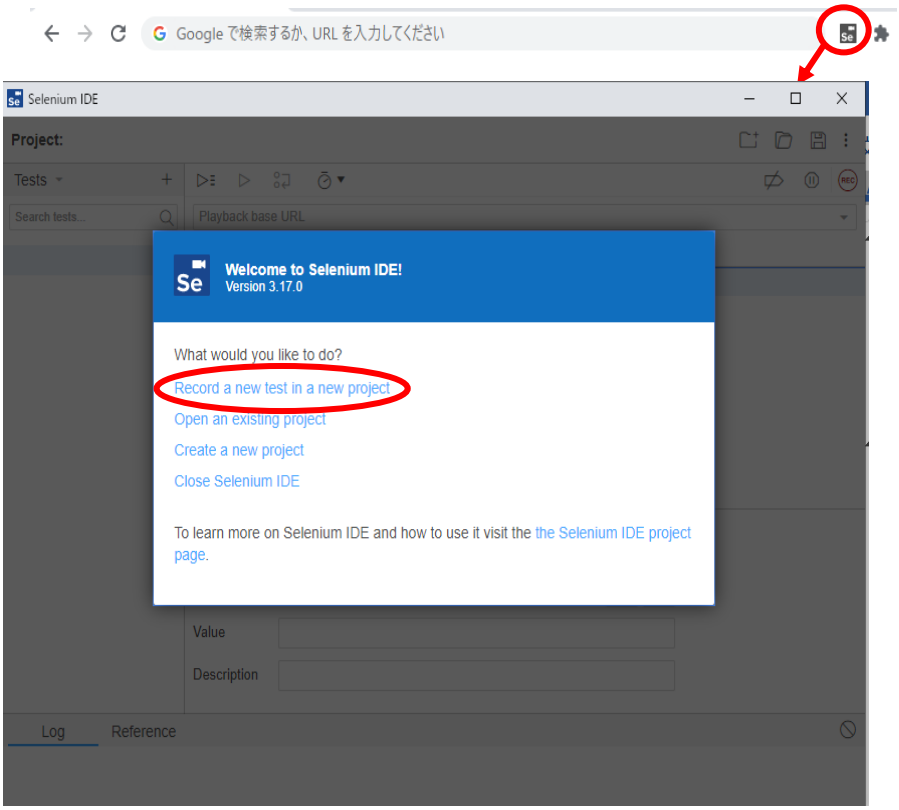
フォルダ名は v4.0 ですがは.NETFramewaok v5 迄はこれが対応します

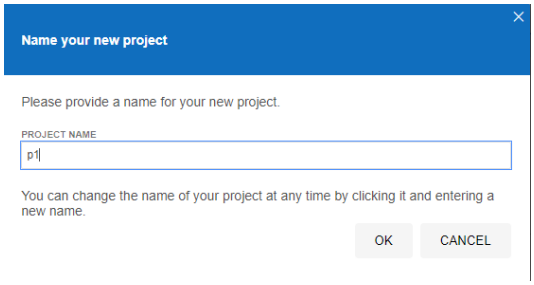
[2] 64 ビット版コンパイラは”～¥Framework64¥csc.exe”。Selenium は 64 ビットでは特定の命令（SendKeys）で異常に遅くなるという現象がでる

[3] Windows7 以降、v4.0.30319 の後継バージョンは出ていません。下記 URL の Note 日本語訳『.NET Framework4.8 は.NETFramework の最後のバージョン』参照
https://docs.microsoft.com/en-us/dotnet/framework/migration-guide/versions-and-dependencies

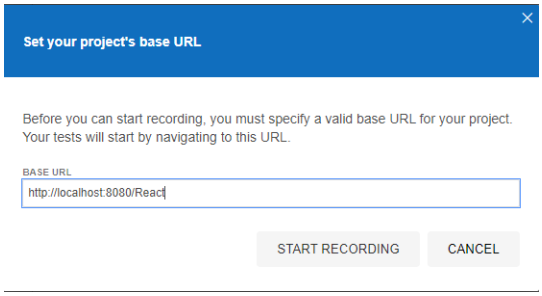
2. Selenium IDE の操作

ツールバーから Selenium IDE 拡張機能を起動し、[Record a new test in a new project]を選びます。Selenium IDE はプロジェクト＞テスト という単位で扱います。



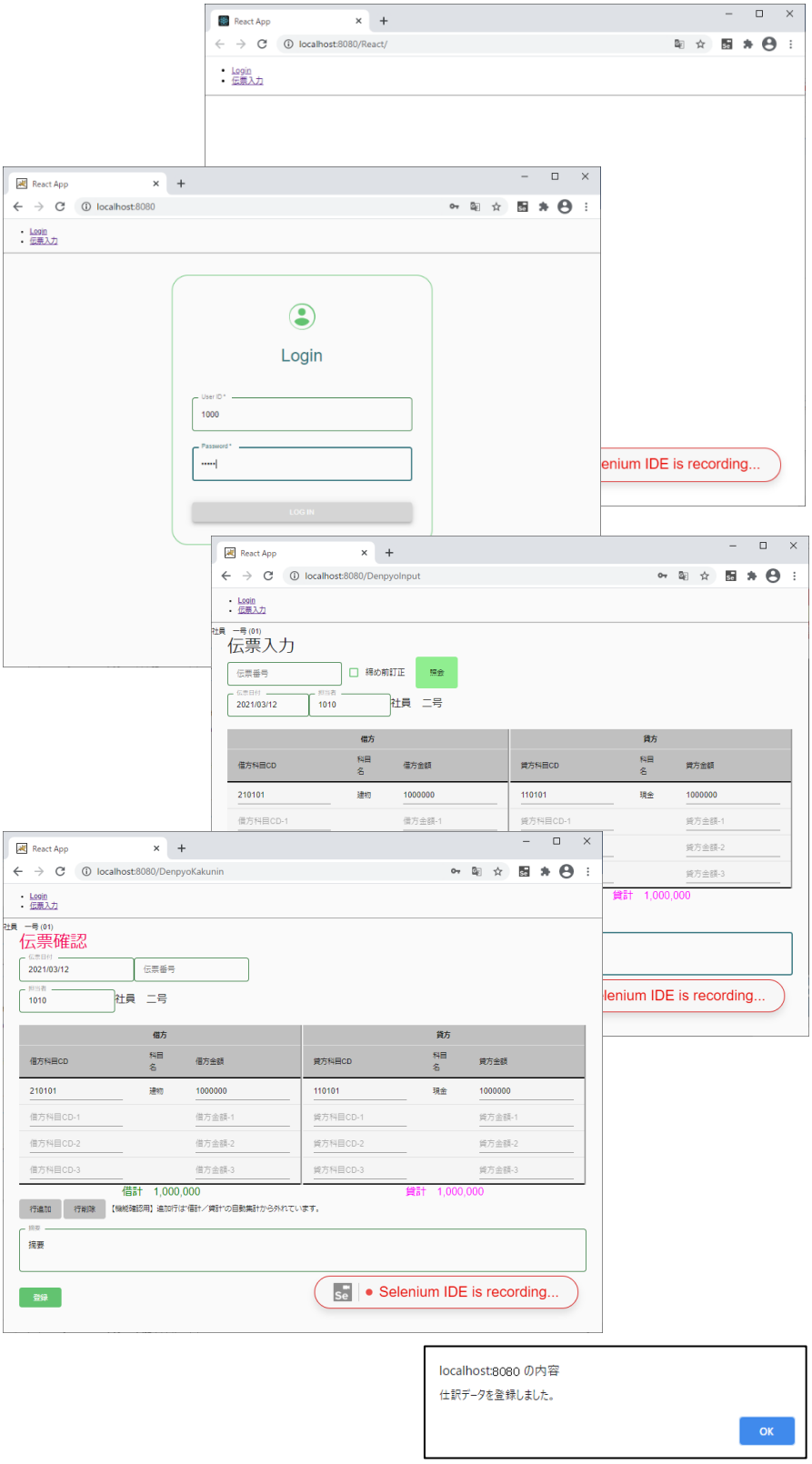


① プロジェクト名を指定（以降説明上“p1”とする）



② 試験対象の Web アプリの URL を指定し、[START RECORDING]押下

㋐ ②で指定した URL が開く



㋑ ログイン

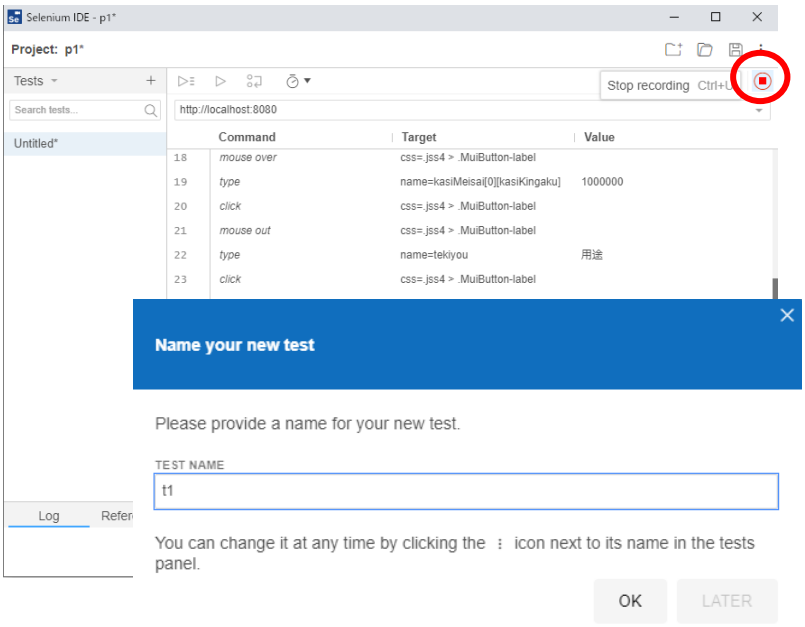
㋒ 伝票入力

㋓ 伝票確認

㋔ 登録正常終了

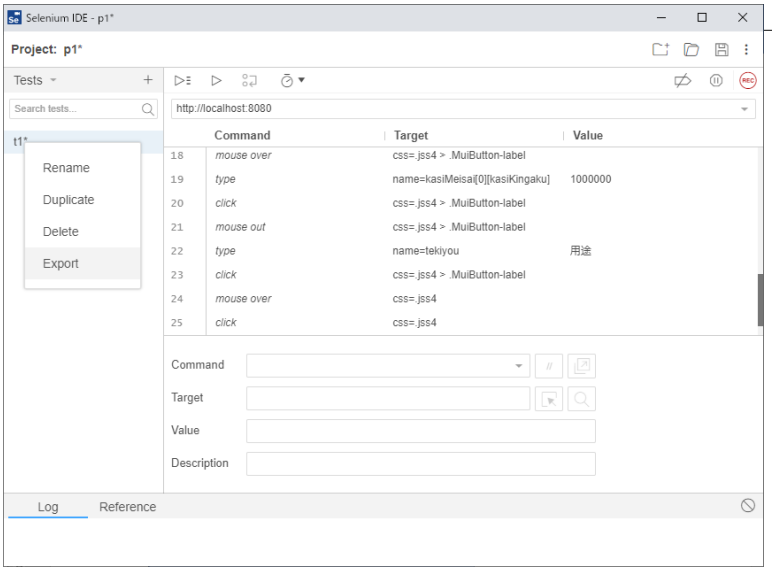
試験自動化(Selenium,AutoIt,電源)

試験対象範囲の一連の操作が終わったら...

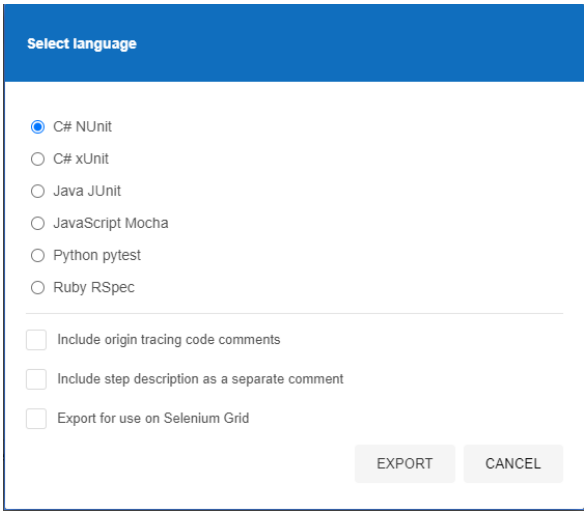


③ Selenium IDE 拡張機能の記録を停止

④ テスト名を指定し OK
(以降説明上“t1”とする)



⑤ 画面左の[Tests]から
“t1”(④参照)を右クリックし
Export を左クリック



⑥ Export の内容に[C# NUnit]
を選択し[EXPORT]押下
ファイル名を付けて保存
(以降説明上“T1Test.cs”とする)

3. コンパイル

コンパイル・実行を行う環境として、以下のフォルダ構成を作成します。ダウンロードした資材をbin、lib のフォルダに入れてください。

作業用フォルダ
├──bin
│　　chromedriver.exe
│　　WebDriver.dll
│　└ {NUnit.Framework-3.13.1.zip bin/net45 フォルダ内の全ファイルを bin 直下へ}
│
├──lib
│　　nunit.framework.dll
│　　WebDriver.Support.dll
│
└──P1　（プロジェクト名）
　　go.bat　…別途説明
　　T1Test.dll　…コンパイルして作った動的リンクライブラリ
　　T1Test.pdb …コンパイルで作られるデバッガー用シンボルファイル
　　
　　├──log
　　　　TestResult.xml …Selenium の実行により作られる
　　
　　└──T1　（テスト名）
　　　　T1Test.cs　…Selenium IDE から Export し修正したソース
　　　　コンパイル.bat　…この後説明

（1） コンパイル・スクリプト(コンパイル.bat)

```
setlocal enabledelayedexpansion

cd /d C:¥Windows¥Microsoft.NET¥Framework¥v4.0.30319 ①

set path=%path%;C:¥Windows¥Microsoft.NET¥Framework¥v4.0.30319 ①
set dllPaths=system.dll,system.drawing.dll,system.windows.forms.dll,system.io.dll,System.Reflection.dll

for %%I in (%~dp0*.cs) do (
  set exePath=%~dp0..¥%%~nI.dll
  csc.exe /optimize+ /t:library /debug+ /platform:anycpu /out:!exePath! %%I /r:%dllPaths% ②
/r:%~dp0..¥..¥lib¥nunit.framework.dll,%~dp0..¥..¥bin¥WebDriver.dll,%~dp0..¥..¥lib¥WebDriver.Support.dll
)

if %ERRORLEVEL% == 0 (
  goto SUCCESS
)

echo ERROR %ERRORLEVEL% %exePath%
goto endc

:SUCCESS
echo SUCCESS %exePath%

:endc
pause
```

① 項番 1.2（4）の C#コンパイラのフォルダを指定する

② コンパイル

```
for %%I in (%~dp0*.cs) do (
   set exePath=%~dp0..¥%%~nI.dll
   csc.exe /optimize+ /t:library /debug+ /platform:anycpu /out:!exePath! %%I
/r:%dllPaths%
/r:%~dp0..¥..¥lib¥nunit.framework.dll,%~dp0..¥..¥bin¥WebDriver.dll,%~dp0..¥..¥lib¥WebDriver.Support.dll
)
```

※行っている内容は、”コンパイル.bat”があるフォルダから拡張子“cs”のファイルを探し、コンパイルして一階層上のフォルダに dll（動的リンクライブラリ）を作成しています。

この時、コンパイラと同じフォルダに入っている

system.dll, system.drawing.dll, system.windows.forms.dll, system.io.dll, System.Reflection.dll

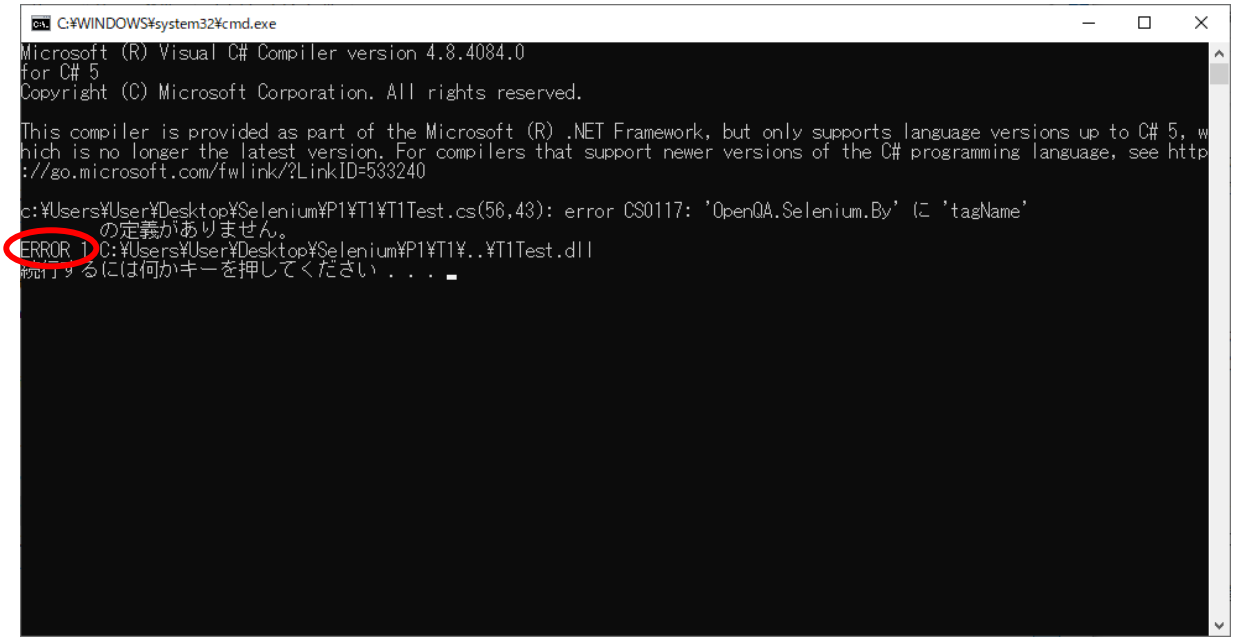
と、コンパイル.bat”が存在しているフォルダから 2 階層上の lib(..¥..¥lib)bin(..¥..¥bin)から

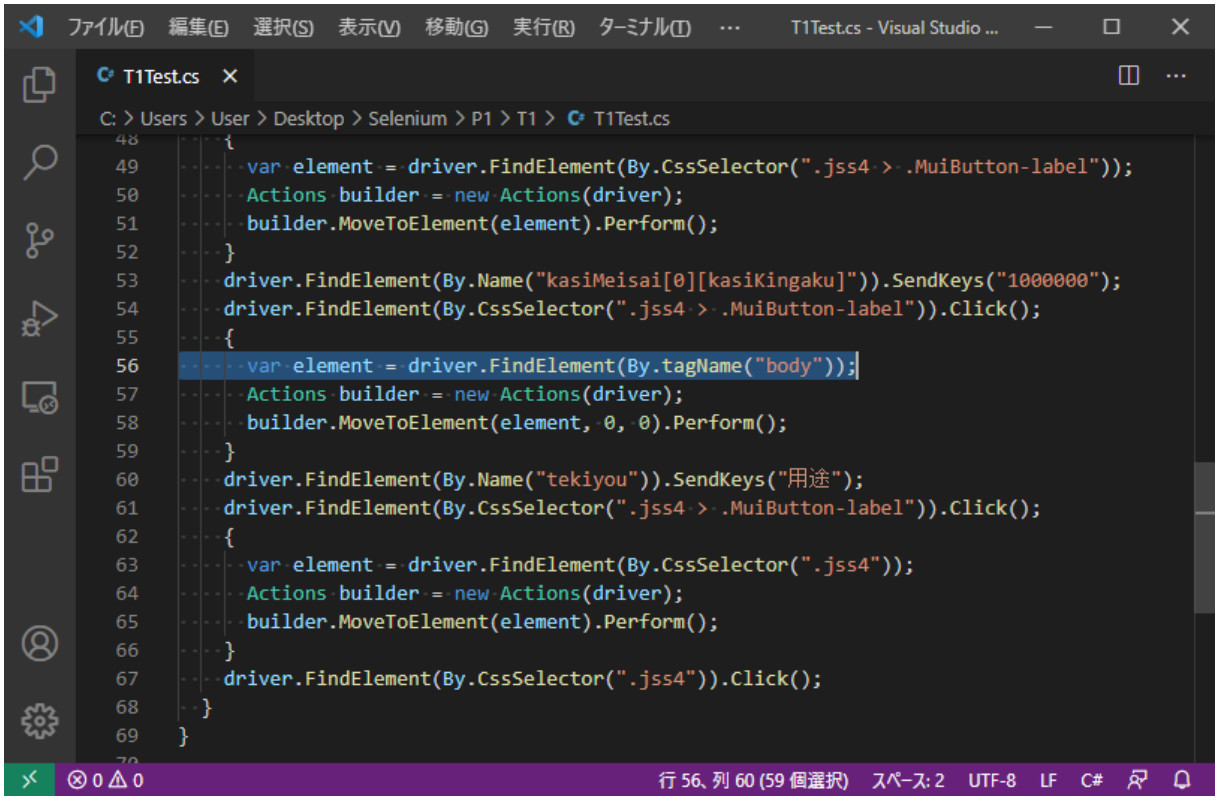
nunit.framework.dll, WebDriver.dll, WerDriver.Support.dll

をリンクしています（/r オプション）。

（2） コンパイルエラーの対処

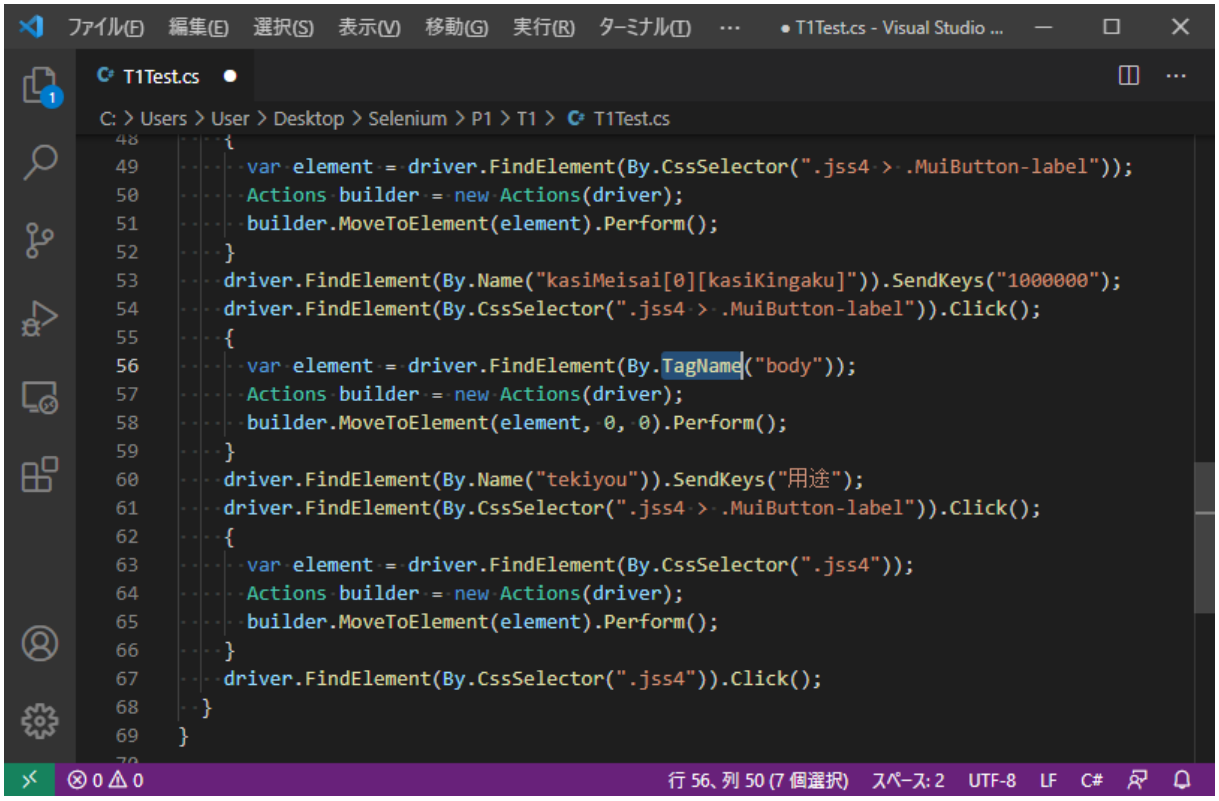
Selenium IDE から Export した T1Test.cs をコンパイルすると 56 行目にエラーが発生しています。

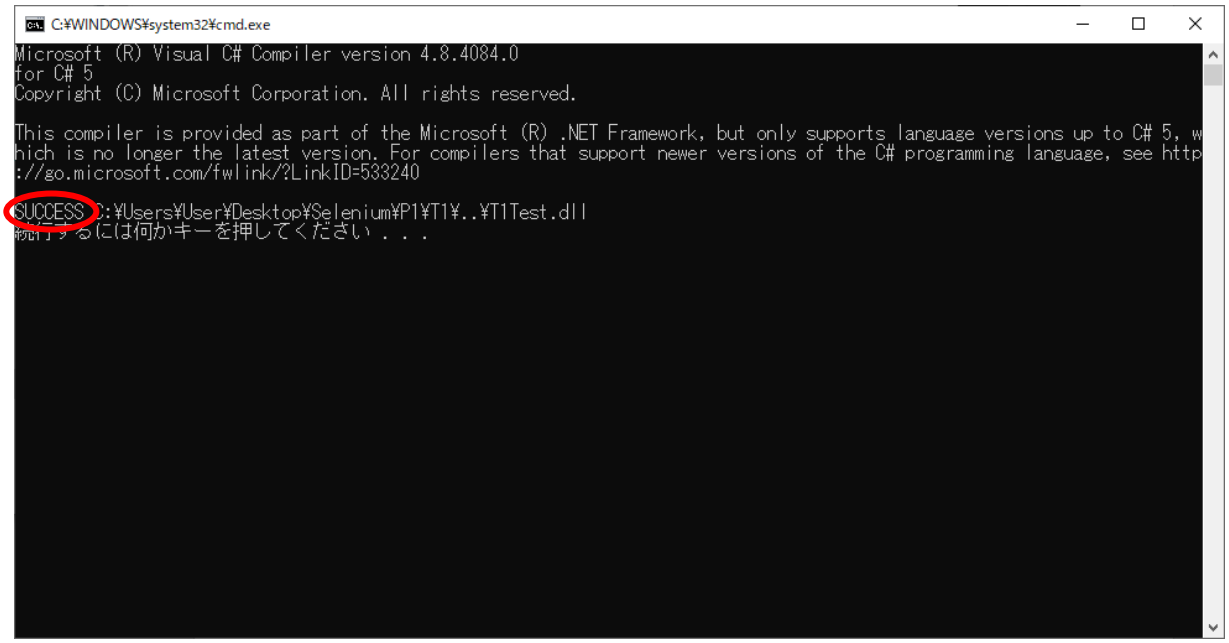




エラー内容は tagName()というメソッドが定義されていないというものですが、これは Selenium IDE のバグです。C#の場合、メソッド名は一文字目は大文字の TagName()となるので、「By.tagName("body")」を「By.TagName("body")」と修正します。

【改修後】





コンパイルが成功して、T1Test.dll が T1 フォルダの一階層上（P1 フォルダ）にできました。

4. Selenium の実行

（1） 実行スクリプト(go.bat)

以下の内容の bat ファイルを P1 フォルダに格納します。

bat ファイルで行っている内容は以下のとおりです。

```
set path=%path%;%~dp0..¥bin ①
cd /d %~dp0log ②

nunitlite-runner.exe ../T1Test.dll --trace=Debug ③

pause
```

① 作業フォルダの bin フォルダを path 環境変数に含めて chromedriver.exe と WebDriver.dll が実行時に参照できるようにします

② bat ファイルと同梱の log フォルダをカレントディレクトリに設定します

※NUnit はカレントディレクトリに実行結果とトレースを出力します

③ NUnit のテストランナーに前項のコンパイルで作った dll の実行を依頼します

（2） スクリプトの実行

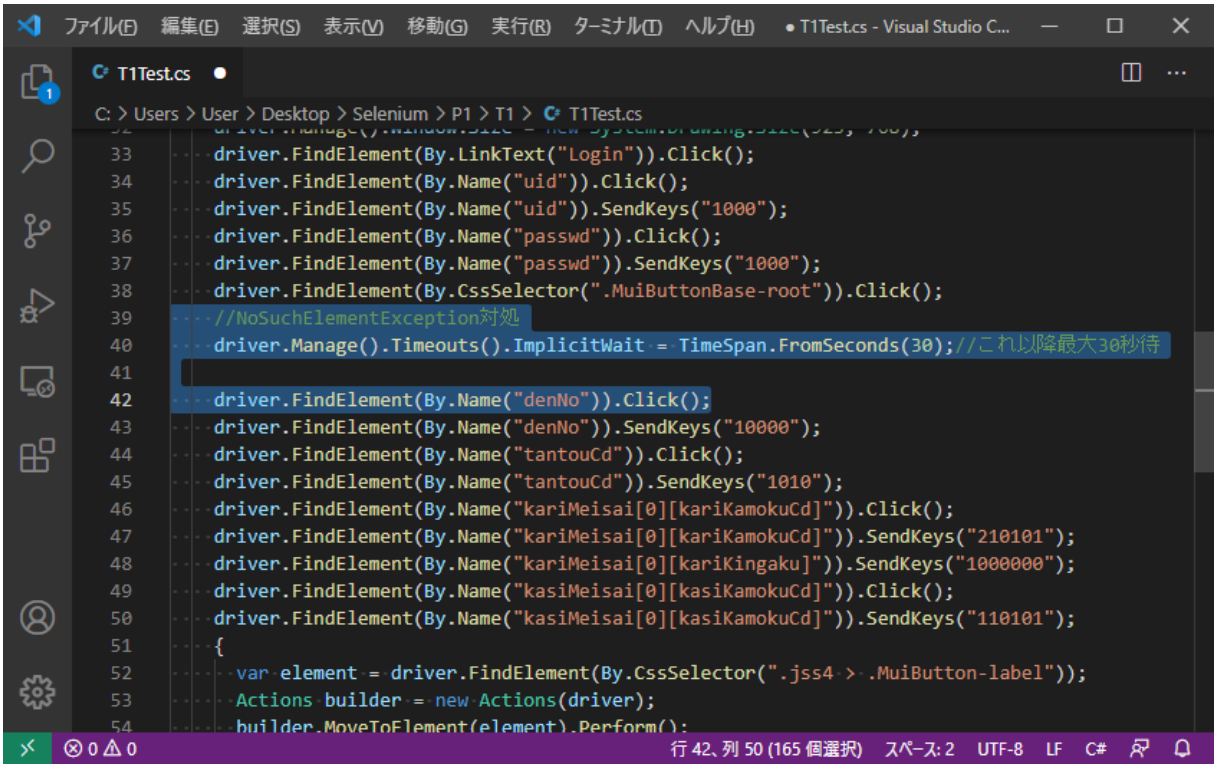
go.bat を実行すると log フォルダに“TestResult.xml”というファイルが作られます。

```
<?xml version="1.0" encoding="utf-8"?>
<test-run id="2" name="T1Test.dll"
fullname="C:/Users/User/Desktop/Selenium/P1/T1Test.dll" testcasecount="1"
result="Failed" start-time="2021-03-15T02:00:54.5489784Z" end-time="2021-03-
15T02:00:59.6343393Z" duration="5.084387" total="1" passed="0" failed="1"
inconclusive="0" skipped="0" warnings="0" asserts="0" random-seed="28615658">
  <command-line><![CDATA[nunitlite-runner.exe  ../T1Test.dll --trace=Debug]]></command-
line>
  <filter />
  <test-suite type="Assembly" id="1002" name="T1Test.dll"
fullname="C:/Users/User/Desktop/Selenium/P1/T1Test.dll" runstate="Runnable"

        （中略）

        <failure>
          <message><![CDATA[OpenQA.Selenium.NoSuchElementException : no such element:
Unable to locate element: {"method":"css selector","selector":"*[name="denNo"]"}
  (Session info: chrome=89.0.4389.82)]]></message>
```

メッセージに実行結果が表示されます。この場合は“denNo”という name を持つ項目が存在せず“NoSuchElementException”いう例外が発生しています。以降で対処方法を説明します。

（３） NoSuchElementException の対処

NoSuchElementException は一番発生し易い例外で、ブラウザが画面上に描画する前の要素（html タグ）を操作しようとして発生します。これは WebDriver と ChromeDraiver が非同期で動作しているのでプログラミング上の対処によりタイミングを合わせる必要があります。



● NoSuchElementException が発生した項目の操作前に以下の処理を追加します。

これは描画待機時間の指定（暗黙的な待機）で、FindElement は TimeSpan.FromSeconds(30)を超えるまで 0.5 秒毎に確認を繰り返すようになります。

driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(30);//これ以降最大 30 秒待

---- 改修／コンパイル／実行 後 log¥TestResult.xml----

<**test-suite** type="TestFixture" id="1000" name="T1Test" fullname="T1Test" classname="T1Test" runstate="Runnable" testcasecount="1" result="Passed" （以下略）

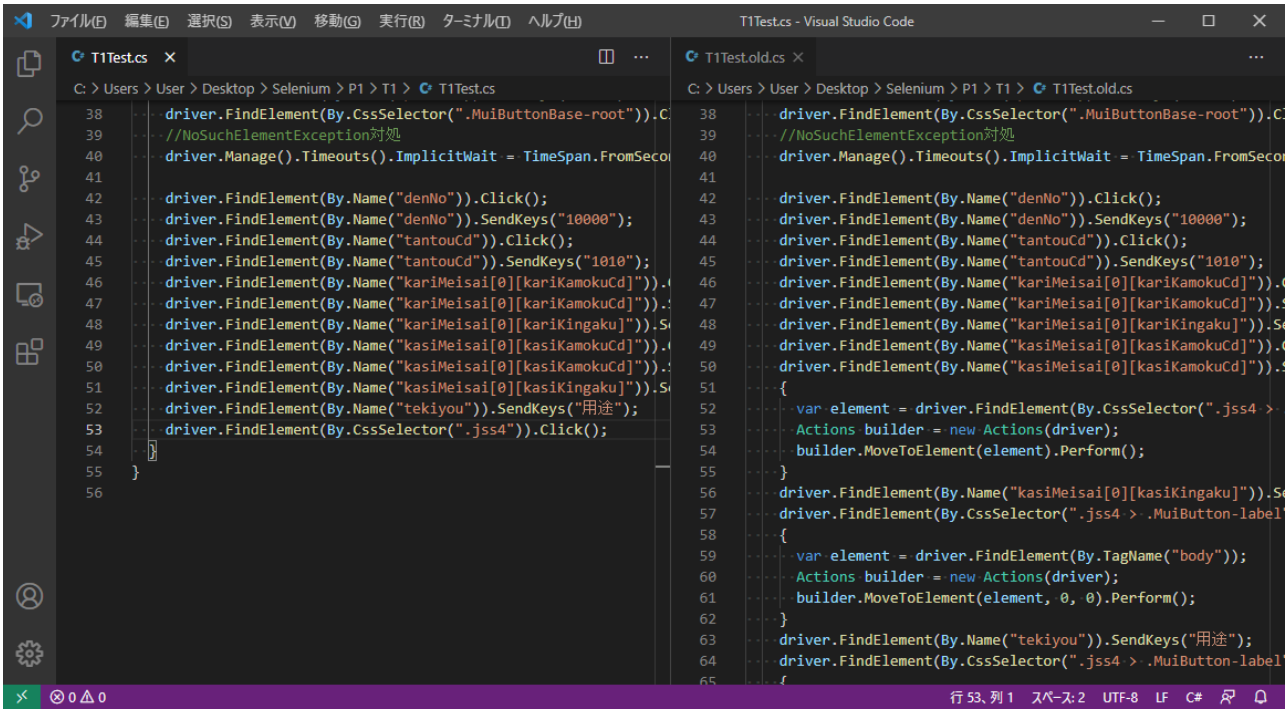
※このように result=”Passed”が出力されたら正常終了です

（４） その他の補正

Selenium IDE は全ての画面操作を記録してしまうため、試験に関係ない操作（カーソルを遷移させるためのクリック等）が記録され Selenimu で再現時にエラーになる場合があります。

① 不要処理の削除

この画面では、input 項目【By.Name(“KasiMeisa[0][KasiXXXX”])】、【By.Name(”tekiyou”)】への入力後は確認ボタン【By.CssSelector(“.jss4”)】の Click だけが試験進行として必要な処理なので、左側のように不要な処理を削除して想定外のエラー発生を抑止することができます。



※項番 2.Selenium IDE の操作㋐～㋓の画面は SPA（シングルページアプリケーション）として作っており、物理的なページ遷移は起きていません（アドレスバー及び画面の内容は JavaScript で書き換えている）。このため、できあがった C#にページ遷移に関連するコードはでてきません。

② セレクターの見直し

Selenium IDE が作り出すコードがどんな状況でも動作するわけではありません。html 上の class は項目の表示状態を変更するために動的に追加・削除が行われます。

以下で指定している” .makeStyles-submit-4”は消される場合があります。

driver.FindElement(By.CssSelector(".makeStyles-submit-4 > .MuiButton-label")).Click();

↓この場合、固定の属性で項目を特定するように変更した方が良いでしょう。

driver.FindElement(By.XPath("//span[text()='確認']")).Click();

※セレクターについては、次項を参照してください

5. 項目追加・変更対応（FindElement）

Selenium IDE から Export して運用開始後に画面の改修が入った場合、項目設定の処理を追加する必要があります。処理の殆どは、「対象を特定」して「操作」するになります。

（1） 対象（要素：Element）を特定する方法（**セレクター**）

driver.FindElement(要素 x)でブラウザが表示している「要素 x」に対して操作できるようにします。要素は以下の形式で特定します。(FindElement 実行時に要素が表示域から隠れていた場合は Selenium が表示領域まで自動的にスクロールします)

① By.Id：属性=id で要素を特定します（html 仕様の規定では、id は文書中で一意のハズ）

② By.Name：属性=name で要素を特定します（name は input 項目につける属性。form で送られる）

例：driver.FindElement(By.Name("uid")).Click();

③ By.LinkText：a タグで表示されるテキストで特定します。複数の要素が該当する場合は[n](n は 1~)番目で指定し、指定がない場合は最初の要素が操作対照になります。

例：driver.FindElement(By.LinkText("Login")).Click()

④ By.XPath：目的のタグを抽出条件や、タグ階層（”/”で階層を継続）で特定します。”//”でドキュメントの先頭から、“@属性名=’x~x’” で各種の属性を使い要素を特定します。

例：driver.FindElement(By.XPath("//input[@type='image']")).Click();

※html 先頭(”//”)から、input タグで type 属性が’image’の条件に合致する要素を探します

⑤ 複数の要素から選択（n 番目）する

複数の要素が抽出される可能性がある場合は[n]（n 番目）で一つの要素に特定する。

例：By.XPath("(//div[@class='div_1'])[1]/table/tbody/tr/td[5]"))

※**[注意]**一つの要素に class が複数指定されている場合は、XPath では数と順番が完全に一致していなければならない。また、**class は JavaScript で動的に追加／削除が行われる**ため自分で意識して付加した class を By.CssSelector で指定した方がよい

⑥ By.CssSelector：スタイルで探します

例：By. CssSelector (".makeStyles-submit-4 > .MuiButton-label")

※CssSelector は、スタイルシートと同様の表現で要素を選択します

”.”：クラス…“.makeStyles-submit-4”は class=”makeStyles-submit-4”の指定がある要素

“#”：id …”#idsample“は id=”idsample”の指定がある要素

なし：html タグ…”*”を指定すると全てのタグが合致。”input.cl”は cl クラス付の input タグ

※※“>”は親子関係で、右側が左側の要素の直下にあることを表します

http://www.htmq.com/selector/child.shtml

⑦ SELECT タグの扱い

id で select タグの階層を選択し、文字列で option の一つを選択（半角カナの場合、認識できない場合がある⇒④の応用で実装）

例：new SelectElement(driver.FindElement(By.Id("selection1"))).SelectByText("opt1");

例：driver.FindElement(By.Id("popup-items")).FindElement(By.XPath(".//li[.='B']")).Click();

※二つ目の FilendElement の XPath を”.”から始めることで、最初の FindElement の結果(Id)を起点に探索をしている。（上記条件の内容は、タグ内に”B”という文字列がある li タグを見付ける）

＜その他の条件の例＞

By.XPath("//input[@name='print_type' and @value='1']")･･･複数属性の and 条件

By.XPath(".//li[contains(., '探索文字列')]")　　･･･文字列を含む

By.XPath("//div[starts-with(@id, 'circle_')][last()]")　･･･文字列で始まる(“last()”は要 Linq* 1)

* 1 :using System.Linq;

参考サイト：https://www.selenium.dev/documentation/ja/webdriver/web_element/

●操作したい要素の特定が難しい場合

Chrome の F12 で Elements の上でカーソルを上下するとブラウザ上にその要素を表示している部分がハイライトされます。

（２） 操作する基本的な方法

以下が主な操作です。

・Click()･･･要素に対してマウスクリックを行ったのと同じ

・Clear()･･･入力欄に設定されている入力済みテキストを消去します

・SendKeys(文字列)･･･要素の表示中の文字列の末尾に文字列を追加します（通常 Clear 後に行う）

※その他に、マウスの操作をシミュレートする Actions クラスがあります。右クリックや HTML5 の Canvas を扱えますが長くなるのでここでは割愛します

https://www.selenium.dev/documentation/ja/support_packages/mouse_and_keyboard_actions_in_detail/

6. ブラウザとの同期（Wait、Sleep）

ブラウザは操作を行うたびに内容が変化（通信、画面描画、etc.）しています。要素の特定（FindElement）を要求した時点で常に期待した要素が存在し、かつ操作可能であるとは限りません。

（1）ブラウザの準備が整うのを待つ

以下のタイミングをに合わせる必要があります。

① サーバから受け取った html の描画や、Javascript の動作終了を待つ

② クリック等で内容が書き換わった後に操作を行いたい⇒旧画面の廃棄を待ち①へ

③ 表示されているが操作を受け付けられない場合（Element is not clickable 発生のとき）
⇒要素がクリック可能になるのを待つ

④ サーバとの通信中に画面を覆って操作を抑止している場合
（Other element would receive the click 発生のとき）⇒覆いが取れるのを待つ

＜同期の方法＞

① 描画待機時間の指定（暗黙的な待機）

```
driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(30);
```

※FindElement はこの時間を超えるまで 0.5 秒毎に確認を繰り返すようになります

② 同一画面／別画面の描画開始を待つ（旧画面破棄待）

```
System.Threading.Thread.Sleep(2000);//2 秒処理を停止
```

※更新前の要素が破棄され新たに画面描画が始まった後は、FindElement(s)で暗黙的に待機

※この待機は暗黙的な待機と異なり指定した時間だけ確実に停止する。多用しない方がよいが、DB 更新待ちや環境起因の**偶発的な同期失敗が発生する場合はこの方法による調整**が必要です。

③ 要素がクリック可能になるのを待つ（明示的な待機）

```
var wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10)); //Element 変化最大待秒数
(wait.Until(
    ExpectedConditions*1.ElementToBeClickable(By.Id("btnYesNo")))
).Click(); //状態変化後 Click
```

※通常は①の暗黙的な待機で FindElement(s)を行うだけで十分ですが、要素は現出しているにもかかわらず Click できない状態が発生した場合は、準備が整うまで明示的に待つ必要があります

④ 覆いがとれるのを待つ

```
var wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30)); //Element 変化最大待秒数
wait.Until(
    ExpectedConditions*1 InvisibilityOfElementLocated(By.XPath("//iframe[@id=shield]"))
); //消待ち
```

※操作対象の要素の準備が整っていても別の要素が意図的に覆っている場合は、覆っている要素が消えるのを待つ必要があります

*1【注意】C#版 WebDriver では Vup に伴い ExpectedConditions が廃要素にりました
（Java から移植が難しかったそうです）⇒次ページに対応コード

【明示的な待機 …ExpectedConditions を C#でどうするか...】

ExpectedConditions の後は FindElement で発生する可能性がある例外で、以下があります。

・NoSuchElementException 使用できない要素

・ElementNotVisibleException 要素がページ内に表示されない（非表示）

・StaleElementReferenceException 要素が Web ページに存在しなくなった

〔クリックできる（ElementToBeClickable）迄待機させたい場合は以下のコードで事足ります〕

```
WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));//10 秒間試行
wait.Until(e => e.FindElement(By.XPath("//要素"))).Click();
```

※Until の中はラムダ式で、詳しくは Web で。

内容は、FindElement の結果が空(C#では false 扱い)以外になるかタイムアウトになる（wait 秒数）まで繰り返し、要素が 10 秒の間に現れなかった場合、以下を表示して終了します。

「 OpenQA.Selenium.NoSuchElementException : no such element: Unable to locate element: {"method":"xpath","selector":"//要素"}」

〔例外を見極めて対処を振り分けたい場合は、以下のようにします〕

```
public static Func<IWebDriver, IWebElement> ElementToBeClickable(By locator)
{
    return (driver) =>
    {
        var element = ElementIfVisible(driver.FindElement(locator));
        try{
            if (element != null && element.Enabled){
                return element;
            }
            else{
                return null;
            }
        }catch (StaleElementReferenceException){
                return null;
        }
    };
}
private static IWebElement ElementIfVisible(IWebElement element){
    return element.Displayed ? element : null;
}
```


（２）条件により要素が表示されない／作られない場合

本来作られるべき要素が一定時間作られないのは試験としてはエラーですが、試験の趣旨と関係ない周辺条件の変化で要素が存在しない場合もあります。この状況は以下の方法で迂回できます。

① 表示されない（「存在する=定義はある」けど不可視）

```
if ( driver.FindElement(By.Id("matchingAlert")).Displayed ) {
} else {　～不可視の場合の処理～　}
```

② 作られない（定義がない）

```
if ( driver.FindElements(By.Id("matchingAlert")).Count > 0 ) {
    driver.FindElement(By.Id("matchingAlertClose")).Click();
}
```

※FindElement"s"でも該当 Element がない場合は「暗黙的な待機時間」が発生します

③ 選択肢が存在しない場合がある

```
var wTime = driver.Manage().Timeouts().ImplicitWait;//待機時間設定値保存
driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(1);//暗黙的待ち時間 1 秒
try {
    driver.FindElement(By.Id("popup-items-division")).FindElement(By.XPath(".//li[.='あ']")).Click();
} catch ( Exception ) {}
driver.Manage().Timeouts().ImplicitWait = wTime;//待機時間設定値戻し
```

※待機時間を最小にして NoSuchElementException を発生させ、有無を判定する

7. 画面の切り替え（WidowHandles）

要素の操作は driver(WebDriver)を通して行いますが、driver は常に一つの画面に関係付いており自動的に切り替わることはありません。操作する画面を変える場合は明示的に画面を指定します。

① 子画面への切り替え

```
driver.FindElement(By.XPath("(//img[@alt='子画面表示'])[1]")).Click();//子画面を表示
SwitchScreen(“子画面”)
/**************************************
目的のスクリーンを前面に持ってきます。
<para> 1.html の title
**************************************/
protected void SwitchScreen(String scr) {
    int cnt = 0;
    bool whloop = true;
    while ( whloop ) {
        System.Threading.Thread.Sleep(500);
        Console.WriteLine (scr + ">画面表示確認");//TestResult.xml に出力されます
        Assert.IsFalse(++cnt > 30, "画面が表示されませんでした。<" + scr +">");
        try {
            foreach (string wh in driver.WindowHandles) {
                driver.SwitchTo().Window(wh);
                if(driver.Title == scr){
                    ((IJavaScriptExecutor)driver).ExecuteScript("window.focus();");
                    whloop = false;
                    break;
                }
            }
        } catch ( NoSuchWindowException ) {
            Console.WriteLine (scr + ">画面表示待ち");
        }
    }
}
```

※driver.WindowHandles でブラウザのウィンド（またはタブ）を取得し、タイトルを比較して目的のウィンドを見つけたら JavaScript でフォーカスをあてている

② 画面を閉じる

```
driver.Close(); //制御中の画面（子画面）を閉じ...
SwitchScreen("自画面");//driver の制御を子画面表示元に戻す＆活性化
```

※driver.Close()で制御対象の画面が閉じますが、自動的に別の画面に制御が移ることはありません。SwithScreen()で実行している“driver.SwitchTo().Window()”が必要です。

8. アラート（AlertIsPresent)、モーダル画面

アラートとモーダル画面に対してはそれに対して操作を行うまで先に進めません。特にアラートに対しては表示している文言のチェックを行っています。

① アラート

＜アラート・チェック例＞

```
Assert.That(driver.SwitchTo().Alert().Text, Is.EqualTo("仕訳データを登録しました。"));
```

※Selenium IDE が Export するソースは Alert の文言チェックを上記のように行っており、通常は正常に動作しますが、selenium.dev サイトのサンプルでは以下のようになっています

```
//アラート表示待ち
IAlert alert = wait.Until(ExpectedConditions.AlertIsPresent());
//アラート文言取得
string text = alert.Text;
//OK button
alert.Accept();
//Cancel button
alert.Dismiss();
```

② モーダル画面

モーダル画面は、表示の土台になっている画面の一部として操作可能です。表示中の状態で F12 を使い要素を特定して操作を行ってください。

9. キャプチャ（GetScreenshot）

Selenium の実行途中でブラウザに表示している内容のイメージをファイルに保存することができます。

① 必要であれば画面の再描画開始待ち（更新前の画面をキャプチャしてしまうのを防ぐ）
System.Threading.Thread.Sleep(2000);

② 画面最後部の要素をFindElementして描画完了を待つ
driver.FindElement(waitElement);//一番下の項目

③ 画面イメージをファイル(jpeg)に保存
((ITakesScreenshot)driver).GetScreenshot().SaveAsFile(@"*ファイルパス+ファイル名.jpg*",
ScreenshotImageFormat.Jpeg);

<ファイルパス・ファイル名指定例>
@"..¥log¥Screenshot_" + "_" + DateTime.Now.ToString("HHmmss.fff") + ".jpg"

※上記例は log フォルダに“Screanshot_*タイムスタンプ*.jpeg”というファイル名で、実行中の画面が保存されます

10. ダウンロード（AutoIt）

Selenium で「右クリック＋名前を付けて保存」はできません。

対象のファイルのリンクを取得してダウンロードする方法もありますが静的なファイルだけに限られます。AutoIt を使うと Web アプリで動的に作成する PDF をダウンロードしたり、印刷のダイアログを操作することもできます。

① 以下のサイトよりインストーラか zip をダウンロードします
https://www.autoitscript.com/site/autoit/downloads/
「AutoIt Full Installation.」か「AutoIt– Self Extracting Archive」を選択します。

使うのは、以下の 2 つだけです。
・AutoItX3.Assembly.dll
・AutoItX3_x64.dll

② AutoItX3.Assembly.dll と AutoItX3_x64.dll を WebDriver.dll と同じフォルダに入れる

③ コンパイル.bat のリンク対象に AutoItX3.Assembly.dll を追加（/r オプション）

```
for %%I in (%~dp0*.cs) do (
 set exePath=%~dp0..¥%%~nI.dll
 csc.exe /optimize+ /t:library /debug+ ^
/platform:anycpu /out:!exePath! %%I ^
/r:%dllPaths% /r:%~dp0..¥..¥lib¥nunit.framework.dll^
,%~dp0..¥..¥bin¥WebDriver.dll^
,%~dp0..¥..¥lib¥WebDriver.Support.dll^
,%~dp0..¥..¥bin¥AutoItX3.Assembly.dll
)
```

※本来 1 行で書く csc.exe のパラメータを、行末に“^”（エスケープ文字）をおいて改行しています

（次ページにコード例）

④ コード例

```
using AutoIt; //AutoIt の使用宣言

        <中略>・・・マーカーがない部分は、Selenium IDE から Export した部分

  [TearDown]
  protected void TearDown() {
    driver.Quit();
  }
    public string waitForWindow(int timeout) {
    try {
      Thread.Sleep(timeout);
    } catch(Exception e) {
      Console.WriteLine("{0} Exception caught.", e);
    }
    var whNow = ((IReadOnlyCollection<object>)driver.WindowHandles).ToList();
    var whThen = ((IReadOnlyCollection<object>)vars["WindowHandles"]).ToList();
    if (whNow.Count > whThen.Count) {
      return whNow.Except(whThen).First().ToString();
    } else {
      return whNow.First().ToString();
    }
  }
  [Test]
  public void t2() {
    driver.Navigate().GoToUrl("http://focs.co.jp/");
    driver.Manage().Window.Size = new System.Drawing.Size(1146, 768);
    driver.FindElement(By.CssSelector("li:nth-child(6) img")).Click();
    vars["WindowHandles"] = driver.WindowHandles;
    driver.FindElement(By.LinkText("Windows に Docker のコンテナを稼働【PDF】")).Click();
    vars["win642"] = waitForWindow(2000);
    driver.SwitchTo().Window(vars["win642"].ToString()); //ダウンロード対象のタブに driver を設定

    AutoItX.Send("^s"); //ctrl+s を押下
    AutoItX.WinWaitActive("名前を付けて保存", "", 2000); //「名前を付けて保存」ダイアログを待って 2 秒待機
    AutoItX.ControlFocus("名前を付けて保存", "", "Edit1"); //「名前を付けて保存」「ファイル名」*1にカーソル
    AutoItX.ControlSend("名前を付けて保存", "", "Edit1", @"C:¥Users¥User¥Desktop¥Selenium¥P2¥log¥save" + "_" +
DateTime.Now.ToString("HHmmss.fff") + ".pdf", 1); //「名前を付けて保存」「ファイル名」にパス+ファイル名を入力
    AutoItX.Sleep(300);//300 ミリ秒待機

    //To click on save button
    AutoItX.ControlFocus("名前を付けて保存", "", "1");//("名前を付けて保存", "", "Button1")//「保存」ボタンにカーソル
    AutoItX.ControlClick("名前を付けて保存", "", "1");//「保存」ボタンをクリック
    AutoItX.Sleep(5000); //ダウンロード完了まで 5 秒待機
    driver.Close();

  }
```

*1：ダイアログ上のコントロールは AutoIt に同梱の Au3Info.exe を起動し
Oprions メニューの“Freeze”を解除してコントロールにカーソルをあてると Basic Control info に表示される情報から設定する…「ファイル名」は Class:”Edit”, instance:”1”⇒“Edit1“

11. config から入力値の設定

接続先の環境やアカウント識別等を実行時に変えるために、config ファイルとしてプログラムの外に切り出すことができます。

① 実装例

```
[Test]
public void t1() {
//環境設定情報読込み
  AppSettingsSection CONFIG = new AppSettingsSection();
  //configファイル読み込み【コマンドラインパラメータのenvより取得】
  var env = TestContext.Parameters["env"];
  if ( env == null ) {
    Console.WriteLine("***  固有config[--params=env= ]は指定されませんでした。");
  }else {
    var envFile = @"..¥" + env + ".config";
    if ( !File.Exists(envFile) ) {
      Console.WriteLine("⚠⚠⚠ envパラメータで指定された、" + env + ".conig ファイルが存在しません");
    }else{
      var envConfFileMap = new ExeConfigurationFileMap { ExeConfigFilename = envFile };
      Configuration exConfig = ConfigurationManager.OpenMappedExeConfiguration(envConfFileMap,
ConfigurationUserLevel.None);
      Console.WriteLine("以下のConfig設定値を受け取りました。");
      foreach(var s in exConfig.AppSettings.Settings.AllKeys) {
        if (CONFIG.Settings[s] == null) {
          CONFIG.Settings.Add(s, exConfig.AppSettings.Settings[s].Value);
          Console.WriteLine("追加>" + exConfig.FilePath + " ... Key: " + s + " Value: " +
exConfig.AppSettings.Settings[s].Value);
        }else{
          CONFIG.Settings[s].Value = exConfig.AppSettings.Settings[s].Value;
          Console.WriteLine("置換>" + exConfig.FilePath + " ... Key: " + s + " Value: " +
exConfig.AppSettings.Settings[s].Value);
        }
      }
      Console.WriteLine("Config設定値は以上です。");
    }
  }
  //描画待機時間の指定（暗黙的な待機）
  driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(5); //5秒
  //driver.Navigate().GoToUrl("http://localhost:3000/");
  driver.Navigate().GoToUrl(CONFIG.Settings["TEST_URL"].Value);
  driver.Manage().Window.Size = new System.Drawing.Size(933, 768);
  driver.FindElement(By.Name("uid")).SendKeys(CONFIG.Settings["uid"].Value);
  driver.FindElement(By.Name("passwd")).SendKeys(CONFIG.Settings["passwd"].Value);
  driver.FindElement(By.Name("passwd")).SendKeys(Keys.Enter);
 (以下略)
```


② *T1*.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v5.0" />
    </startup>
    <!--  -->
    <appSettings>
    <!-- 試験対象のURL -->
        <add key="TEST_URL" value="http://localhost:3000"/>
         <!--
         ●同一のKeyが複数ある場合は、最後のvalueが使用されます
         -->
        <add key="uid" value="1000"/>
        <add key="passwd" value="1000"/>
    </appSettings>
</configuration>
```

③ 実行パラメータの指定

nunitlite-runner.exe ../T1Test.dll --trace=Info --params=env=*T1*

12. 実行時のログ参照

Seleniumの実行環境は標準では実行経過を出力しません。例外発生時にC# (.NETFramework) が出力するスタックトレース等はNUnitを経由（Console.Write, Console.WriteLineの出力先が切り替えられています）して実行終了後にカレントディレクトリのTestResult.xmlに出力されます。

① 標準出力

Console.WriteLine() / Console.Write()

TestContext.Out.WriteLine() / TestContext.Out.Write()

※Console.WriteはC#標準の書き方で、TextContextはNUnitのAPIです。出力経路は同一です。

② エラー

TestContext. Error.WriteLine() / TestContext. Error.Write()

③ 進捗

TestContext.Progress.WriteLine() / TestContext.Progress.Write()

※①の書き方は同様にTestResult.xmlに出力されます。③Progressは実行中のプロンプトに続けて出力されるように説明されていますが、NUnit3.13.1 のNUnitLight-runnerでは出力されませんでした（Ver3旧版では出力を確認しています)。

13. 試験／診断結果の確認

13.1 実行結果の確認

自動で対象のアプリを走行しても、結果を確認しないと正常な状態なのか否かは分かりません。
最後まで正常に動作したか否かは走行後の TestResult.xml を確認します。

```
<?xml version="1.0" encoding="utf-8"?>
<test-run id="2" name="T1Test.dll" fullname="C:/Users/User/Desktop/Selenium/P1/T1Test.dll" testcasecount="1"
result="Passed" start-time="2021-03-19T05:27:03.7238661Z" end-time="2021-03-19T05:27:30.7735716Z"
duration="27.040423" total="1" passed="1" failed="0" inconclusive="0" skipped="0" warnings="0" asserts="3"
random-seed="904187140">
                                        (中略)
    <test-suite type="TestFixture" id="1000" name="T1Test" fullname="T1Test" classname="T1Test"
runstate="Runnable" testcasecount="1" result="Passed" start-time="2021-03-19T05:27:03.7394851Z" end-
time="2021-03-19T05:27:30.7735716Z" duration="27.025996" total="1" passed="1" failed="0" warnings="0"
inconclusive="0" skipped="0" asserts="3">
      <test-case id="1001" name="t1" fullname="T1Test.t1" methodname=
runstate="Runnable" seed="1728791141" result="Passed" start-time="2021-
time="2021-03-19T05:27:30.7579403Z" duration="27.006323" asserts="3">
        <output><![CDATA[以下の Config 設定値を受け取りました。
追加>C:¥Users¥User¥Desktop¥Selenium¥P1¥T1.config ... Key: TEST_URL Value: http://localhost:3000
追加>C:¥Users¥User¥Desktop¥Selenium¥P1¥T1.config ... Key: uid Value: 1000
追加>C:¥Users¥User¥Desktop¥Selenium¥P1¥T1.config ... Key: passwd Value: 1000
Config 設定値は以上です。
Assert-1 elTekiyo.Text=用途を入れてください。
Assert-2 elTekiyo is 0
]]></output>
      </test-case>
    </test-suite>
  </test-suite>
</test-run>
```

試験の結果

Console.WriteLine()の出力

〔試験の結果〕
total 実行されたテストケースの総数
passed 合格したテストケースの数
failed 失敗したテストケースの数
warnings 警告メッセージの数
inconclusive 結果が確定しなかったテストケースの数
skipped スキップされたテストケースの数
asserts テスト実行で実行された assert の数

※全てが正常に動作した場合、passed=total(faild, warnings, inconclusive, skipped 全て”0”)になります。asserts は Selenium 実行プログラムに組み込んだ Assert の数です。

13.2 Assert の組込み

運用環境の診断として走行するのであれば Selenium IDE で作ったテストケースを実行し、試験結果の passed=total であれば問題ないかもしれません。

リグレッションテストとして実行するのであれば、Web アプリが仕様通りの応答を返しているかを日々確認する必要があります。Assert を組込んでおけば確認が自動化できます。

```
driver.FindElement(By.LinkText("Login")).Click();
driver.FindElement(By.Name("uid")).SendKeys(CONFIG.Settings["uid"].Value);
driver.FindElement(By.Name("passwd")).SendKeys(CONFIG.Settings["passwd"].Value);
driver.FindElement(By.Name("passwd")).SendKeys(Keys.Enter);
//伝票入力画面
driver.FindElement(By.Name("denNo")).SendKeys("10000");
driver.FindElement(By.Name("tantouCd")).SendKeys("1010");
driver.FindElement(By.Name("kariMeisai[0][kariKamokuCd]")).SendKeys("210100");
driver.FindElement(By.Name("kariMeisai[0][kariKingaku]")).SendKeys("10000");
driver.FindElement(By.Name("kasiMeisai[0][kasiKamokuCd]")).SendKeys("110100");
driver.FindElement(By.Name("kasiMeisai[0][kasiKingaku]")).SendKeys("10000");

driver.FindElement(By.XPath("//span[text()='確認']")).Click();//摘要未入力のまま確認ボタン押下
//エラーの発生を確認
var elTekiyo = driver.FindElement(By.XPath("//span[text()='用途を入れてください。']"));
Console.WriteLine("Assert-1 elTekiyo.Text=" + elTekiyo.Text);
Assert.That(elTekiyo, Is.Not.Null);//Assert①
//エラーの解消を確認
driver.FindElement(By.Name("tekiyou")).SendKeys("摘要");//摘要入力
var arTekiyo = driver.FindElements(By.XPath("//span[text()='用途を入れてください。']")); //複数形で例外回避
Console.WriteLine("Assert-2 elTekiyo is " + arTekiyo.Count() );
Assert.That(arTekiyo, Is.Empty); //Assert②

driver.FindElement(By.XPath("//span[text()='確認']")).Click();
System.Threading.Thread.Sleep(1000); //サーバ応答待機
driver.FindElement(By.XPath("//span[text()='登録']")).Click();
//確認画面に表示するダイアログの文言確認
Assert.That(driver.SwitchTo().Alert().Text, Is.EqualTo("仕訳データを登録しました。"));//Assert③
```

Assert①：「摘要」が未入力だったときにエラーメッセージが出ることを確認
Assert②：「摘要」に入力することでエラーメッセージが消えることを確認
Assert③：正常に DB 更新ができて、期待通りのアラート文言が表示されていることを確認

※表示されている全ての span タグから文言（'用途を入れてください。'）を探しているのは、メッセージが表示されるタグ固有の識別（id,クラス等）がなかったためです。複雑な画面であれば明示的にクラスを付加する等の対策をした方が良いでしょう...

14. 電源投入と停止・進行管理

継続的インテグレーション（CI:continuous integration）という言葉を聞くようになりました。アジャイル（Ajail）型開発を行う場合、各チームが作った成果物を統合せねばならず、品質保証の手順が必要になった結果でしょうか。チームの数が増えると成果物の統合作業が遅れがちになるので、夜中の人出による作業は避けるのに使えそうな技術を以下に挙げます。

（1）電源投入

早朝、メンバーの出社前に CI/CD を済ませたいというとき、Automatic Power-On で統合用サーバを自動で立ち上げることができます。機種によりますが Window10 をサポートしている HP、Dell のデスクトップであれば PC 起動時の BIOS 設定で自動電源投入の時間が設定できる可能性が十分あります。

（2）電源停止

① シャットダウン

```
shutdown /s /t 30
```

※オプション /s 電源停止(shutdown)、/t 手続き開始までの時間（秒：デフォルトは 30）

② シャットダウン中止

```
shutdown /a
```

※オプション /a 実行の中止(abort)

③ 画面のロック（Selenium 終了後等に GUI の使用を停止）

```
rundll32.exe user32.dll,LockWorkStation
```

（3）進行管理

以下の例では、タスクスケジューラに P プロジェクトのテスト（T1Test、T2Test）と画面をロックする処理を登録し、テスト終了時にシグナルを送って(waitfor)画面をロックします

[タスクスケジューラで起動]

```
start /d %~dp0 zLockScreen.bat
start /d %~dp0 P1.bat
```

[P1.bat]…P プロジェクトテスト（）

```
cmd /c T1Test.bat < nul
ren log¥TestResult.xml TestResult_1-1.xml　※TestResult.xml をリネームして T1 の結果を保存
rem 全テスト終了後 TestResult*.xml から"< test-suite"を grep して結果を集計する
cmd /c T2Test.bat < nul
waitfor /si P1End && exit
```

[zLockScreen.bat]…画面ロック

```
waitfor P1End
rundll32.exe user32.dll,LockWorkStation
```

