Visual Studio Coypu AutomationFramework

User Guide

# 

# 

## 

**Updated: 06/28/2013**

**Author: Wes Phillips**

Contents

[Document Purpose 3](#_Toc360174023)

[Approach 3](#_Toc360174024)

[Framework 3](#_Toc360174025)

[1.1 AutomationCore 3](#_Toc360174026)

[1.2 DataDog 4](#_Toc360174027)

[1.3 page\_objects 4](#_Toc360174028)

[1.4 Tests 5](#_Toc360174029)

[1.5 Validations 5](#_Toc360174030)

[References 6](#_Toc360174031)

[2.1 Standard References 6](#_Toc360174034)

[2.2 Coypu 6](#_Toc360174037)

[2.3 NUnit 6](#_Toc360174038)

[2.4 Automation Core 6](#_Toc360174039)

[How To … 6](#_Toc360174041)

[3.1 How to add a reference 6](#_Toc360174045)

[3.2 Create a New Test 7](#_Toc360174046)

[3.3 Build and Run Tests 9](#_Toc360174047)

[3.4 Edit a Test 10](#_Toc360174048)

[3.5 Modify Excel 10](#_Toc360174049)

[Test Specifics 11](#_Toc360174050)

[4.1 FileLoad1 11](#_Toc360174052)

[4.2 POLA 11](#_Toc360174053)

[4.3 ProcedureInference 12](#_Toc360174054)

[4.4 RevisionOutliers 12](#_Toc360174055)

[4.5 SavingsOpportunityAnalysis 12](#_Toc360174056)

[4.6 VendorPriceFileAudit 12](#_Toc360174057)

[4.7 WorkQueueReports 12](#_Toc360174058)

# Document Purpose

This document walks through our approach to creating a framework utilizing Visual Studio with Coypu functionality in order to allow for maintainability and ease of use in automating the application.

# Approach

The approach taken for the HCA automation project makes use of Visual Studio to create a sustainable and powerful framework that utilizes the API present in the Coypu framework. To accomplish the creation of the framework, classes were created that represent navigation, verification, and the process of populating fields. The framework is also made up of utility classes, and input objects. Scripts were then created to make use of these classes creating the necessary objects to perform tests. These scripts are driven by data contained in an Excel document. Each Excel document represents one set of test data.

# Framework

The Visual Studio Coypu Framework is broken down into the following sections by folders in the project.

## AutomationCore

The AutomationCore is a repository for globally used classes and configuration project files. This section contains any functions, classes, or settings that can be used in multiple projects.

* + 1. Base\_tests
       1. **BaseTest.cs** – Contains application interface functions. This class will set the focus on a new browser window and closes all other browser windows. It then verifies and reports that the current open browser is the expected one. There is also provided a function to save a file from a browser. It also contains functions to close any open Excel windows and for retrieving data from Excel spreadsheets via OleDB Connections and can remove empty data rows.
    2. Input\_objects
       1. **InputObjects.cs** – Contains a standard Dictionary definition used for storing data from the input objects. It is setup to accept 1 header and multiple values for each row of data.
    3. Utility
       1. **FileReader.cs** – Provides functions to load a data input into a Dictionary as defined in “InputObjects.cs”.
       2. **HPGAssert.cs** – Contains basic assertions for evaluating and logging tests.
       3. **HPGElements.cs** - Contains element-specific assertions for evaluating and logging tests.
       4. **ScreenCapture.cs** – Provides a method to take a “Screen Capture” for future debugging purposes.
       5. **TableReader.cs** – Contains methods and functions to convert HTML tables into DataTables and retrieve values from the DataTables.
    4. Constants.cs
       1. Contains global setup constants
    5. SuperTest.cs
       1. Contains basic test preparation and closing functions used in all tests of all projects such as the working directory and usernames and passwords.

## DataDog

Each application automated should have an individual project that contains all test-specific project files. These project files contain the code to execute the tests.

**Input\_files**

* + - 1. Contain source data for data-driven searches. These files are processed through Library/input\_objects/InputObjects.cs
         * **Global/Main.xls** – The main URL data source. The navigation starts on the pages contained in this file.
         * **Test\_specific/**– Source data files containing page criteria and expected results for individual tests

## page\_objects

* + - 1. Defines the classes to be instantiated in the tests for each page of navigation
         * **Master.cs** – Basic template used in all other page\_objects.
         * **MicroStrategyHome.cs** - Contains a set of navigation instructions to open various links from the MicroStrategy home page.
         * **DataDogHome.cs** – Tests navigation to and checks existence of the “Home/SourceTrust DataDog” page.
         * **SharedReports.cs** – Tests navigation to and checks existence of the “Home/SourceTrust DataDog/Shared Reports” page.
         * **AnalysisReports.cs** – Tests navigation to and checks existence of the “Home/SourceTrust DataDog/Shared Reports/Analysis Reports” page.
         * **DetailedSOASummary.cs** – Tests navigation to and checks existence of the “Home/SourceTrust DataDog/Shared Reports/Detailed SOA Summary” page.
         * **QuickSOASummary.cs** – Tests navigation to and checks existence of the “Home/SourceTrust DataDog/Shared Reports/Quick SOA Summary” page.
         * **POLineAnalysis.cs** - Contains a set of navigation instructions to open various links from the “Home/SourceTrust DataDog/Shared Reports/Analysis Reports/PO Line Analysis” (POLA) page.
         * **WorkQueue.cs** - Contains a set of navigation instructions to open various links from the “Home/SourceTrust DataDog/Shared Reports/Analysis Reports/Work Queue” page.
         * **AuditTools.cs** – Tests navigation to and checks existence of the “Home/SourceTrust DataDog/Shared Reports/Audit Tools” page.
         * **VendorPriceFileAuditReport.cs** – Tests navigation to and checks existence of the “Home/SourceTrust DataDog/Shared Reports/Vendor Price File Audit Report” page.
         * **UploadingTools.cs** – Tests navigation to and checks existence of the “Home/SourceTrust DataDog/Shared Reports/Uploading Tools” page.
         * **FileStatistics.cs** – Tests navigation to and checks existence of the “Home/SourceTrust DataDog/Shared Reports/Uploading Tools/File Statistics” page.
         * **FileUploadManagement.cs** – Tests navigation to and checks existence of the “Home/SourceTrust DataDog/Shared Reports/Uploading Tools/File Upload Management” page.
         * **FileRejectedLines.cs** – Contains element definitions for the Rejected Lines pop-up window.

## Tests

* + - 1. Contains executable code to test navigation and search queries.
         * **POLA.cs** –Class containing all tests involving the PO Line Analysis report.
         * **SavingsOpportunityAnalysis.cs** –Class containing all tests involving the Savings Opportunity Analysis reports (Both Quick SOA and Detailed SOA).
         * **WorkQueue.cs** –Class containing all tests involving the Work Queue reports.
         * **FileLoad1.cs** –Class containing all tests involving the File Load pages.
         * **ProcedureInference.cs** –Class containing all tests involving the Procedure Inference in the POLA reports.
         * **RevisionOutliers.cs** –Class containing all tests involving the Revision Outliers in the POLA reports.
         * **VendorPriceFileAudit.cs** –Class containing all tests involving the Vendor Price File Audit report.

## Validations

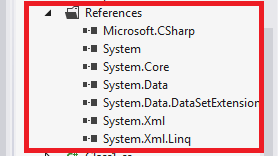
* + - 1. Validation classes used to verify if the returned data exists and determines whether it is correct.
         * **MasterValidations.cs** –General definition of a validation. Used in all other validations.
         * **AnalysisReportsValidations.cs** – Checks existence of expected links on the “Analysis Reports” page.  
           **DataDogHomeValidations.cs** – Checks existence of expected links on the “DataDog Home” page.
         * **FileUploadManagementValidations.cs** – Checks existence of expected links on the “File Upload Management” page.
         * **MicroStrategyHomeValidation.cs** – Checks existence of expected links on the “MicroStrategy Home” page.
         * **SharedReportsValidations.cs** – Checks existence of expected links on the “Shared Reports” page.
         * **UploadingToolsValidations.cs** – Checks existence of expected links on the “Uploading Tools” page.

# References



## Standard References

* + 1. Whenever you create a new project within Visual Studio 2012, there are standard references that are always applied to your solution.



* + 1. Leave all the standard references within your project



## Coypu

* + 1. For our solution to work correctly, we need to utilize Coypu.
    2. Coypu is a robust wrapper for browser automation tools on .Net, such as Selenium WebDriver that eases automating ajax-heavy websites and reduces coupling to the HTML, CSS & JS
    3. Download Coypu from GitHub - <https://github.com/featurist/coypu>
    4. Compile Coypu
    5. After compiling Coypu move the coypu.dll to your project bin\debug folder
    6. Reference this DLL within your projects in the solution

## NUnit

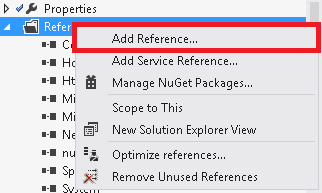
## Automation Core

## Within the project, you will also need to reference the AutomationCore.dll

# How To …

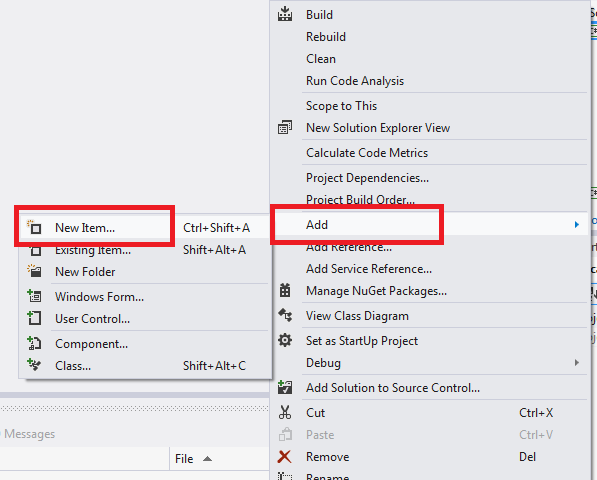


## How to add a reference

* + 1. In the “Solution Explorer”, expand the “References” object of the AutomationCore.
    2. Right-click on the “References” object.
    3. Select “Add Reference” from the context menu  
       
    4. Follow prompts to add desired reference.
    5. You should now see the reference listed in the “References” object.
    6. Repeat these steps for each project the reference is required for.

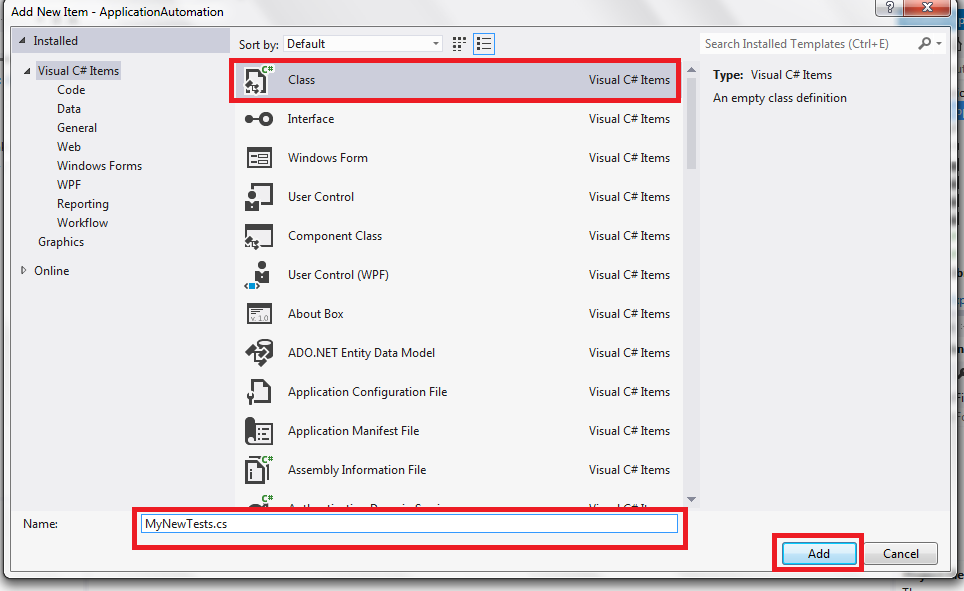
## Create a New Test

* + 1. Open AutomationFramework solution with Visual Studio
    2. Choose the appropriate project
       1. DataDog
    3. Do the following:
       1. Right Click on the project
       2. Click “Add”
       3. Click “New Item”



* + - * 1. Choose “Class”
        2. Give your class a new name

*Example shows “MyNewTests.cs”*

* + - * 1. Click “Add”**
        2. Once your new test class is in your project, there are a few things to add

Add three new “using” statements

using NUnit.Framework;

using Library;

using ApplicationAutomation.test\_modules;

Add “[TestFixture]” as seen below

[TestFixture]

Change your class to be “public” and add “ : SuperTest” to the end

public class MyNewTests : SuperTest

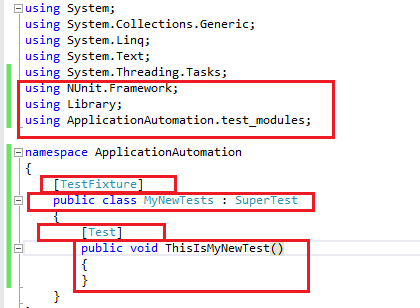
Add “[Test]” as seen below

[Test]

Add your new test (make sure it is “public”)

public void ThisIsMyNewTest()

Add brackets as well “{“ and “}”



## Build and Run Tests

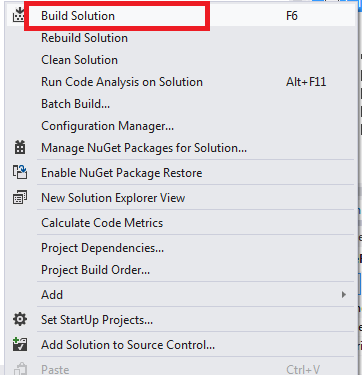
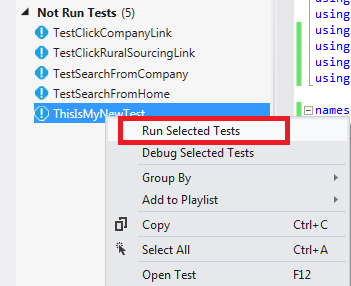
Once you have coded your new test, now it is time to build and run your test. The following steps will show you how to do this.

* + - 1. Verify that your “Test Explorer” window is open
         1. If you do not have the “Test Explorer” window, do the following:

Click on the “TEST” menu

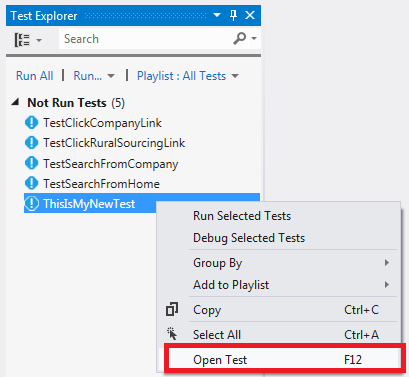
Choose “Windows”

Click “Test Explorer”

* + - 1. Right click on the “AutomationFramework” solution and click “Build Solution”  
         
      2. After you have built the solution, go to the “Test Explorer” window and find a list of available tests to run.
         1. We can see in the image below the test just created called “ThisIsMyNewTest”  
            
      3. To run the test, right-click on the desired test and select “Run Selected Test”  
         
      4. Success!
         1. Fix any bugs that may have been found if the test is unsuccessful

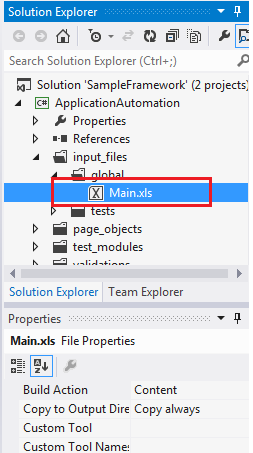
## Edit a Test

* + 1. Right click on the test you want to edit and click on “Open Test”



* + 1. Once you open the test you will see the code and can navigate to the code in the DataDog project to modify the code as needed
    2. After you have edited the test case, refer to “3.3 Build and Run Tests” to build and run the modified test case

## Modify Excel

* + - 1. Within the project is an “input\_files” folder that houses all the external Excel spreadsheets used within the DataDog project. The Excel files should be named the same as the test.
      2. Find the Excel spreadsheet you want to modify and right click on it and click “Check Out” *(if the Excel spreadsheet is tied to a source control)*
      3. Once the Excel spreadsheet has been checked out, you can double click on the spreadsheet and open to modify. *(if the Excel spreadsheet is tied to a source control)*
      4. After you have modified the data, click save and close the Excel spreadsheet
      5. Within Visual Studio, find the Excel spreadsheet you just modified and right click on it. Click “Check In”. This checks the modified Excel spreadsheet you just modified back into source control. *(if the Excel spreadsheet is tied to a source control)*  
           
         ******

# Test Specifics

The tests are grouped by the section and function each test evaluates. Each test has an Excel data file that contains specifications and verification data for the test.



## FileLoad1 These tests evaluate the ability for MicroStrategy to automatically process or reject files dropped into the “DropZone”. Tests included in this group are:

* + Test\_FileLoad\_497 – Test Case 497
  + Test\_FileLoad\_498 – Test Case 498
  + Test\_FileLoad\_501 – Test Case 501
  + Test\_FileLoad\_565 – Test Case 565
  + Test\_FileLoad\_654 – Test Case 654

## POLA These tests evaluate the PO Line Analysis (POLA) report to verify the correct information is displayed and calculated. The tests included in this group are:

* + Test\_POLA\_2895 – Test Case 2895
  + Test\_POLA\_ReportCompleteness – Test Case 95
  + Test\_POLA\_WasteItemsClassifiedAsProcedure – Test Case 2138

## ProcedureInference These tests evaluate the displayed information in the PO Line Analysis report is correct for certain procedure types. Tests included in this group are:

* + Test\_ProcedureInference\_31 – Test Case 31

## RevisionOutliers These tests evaluate specific logic applying to POs. The PO Number is specified in each test’s Test-Specific Input file. The data the reports are verified against is called “Data\_RevisionOutliers.xls” and is used for all tests except “Test\_RevisionOutliers\_WP22 – Test Case 529”. This group includes the following tests:

* + Test\_RevisionOutliers\_WP15 – Test Case 520
  + Test\_RevisionOutliers\_WP16 – Test Case 524
  + Test\_RevisionOutliers\_WP17 – Test Case 531
  + Test\_RevisionOutliers\_WP18 – Test Case 530
  + Test\_RevisionOutliers\_WP19 – Test Case 526
  + Test\_RevisionOutliers\_WP20 – Test Case 527
  + Test\_RevisionOutliers\_WP21 – Test Case 528
  + Test\_RevisionOutliers\_WP22 – Test Case 529
  + Test\_RevisionOutliers\_WP23 – Test Case 528
  + Test\_RevisionOutliers\_WP24 – Test Case 521

## SavingsOpportunityAnalysis These tests evaluate the data displayed and calculated in the Detail SOA Summary and Quick SOA Summary reports. Tests included in this group are:

* + Test\_SOA\_DetailSOASummaryReport\_VerifyFilteredByWaste – Test Case 2285
  + Test\_SOA\_QuickSOASummaryReport\_ReportCompleteness – Test Case 97
  + Test\_SOA\_QuickSOASummaryReport\_VerifyFilteredByWaste – Test Case 2143
  + Test\_SOA\_TotalComparisonSOA – Test Case 91

## VendorPriceFileAudit These tests compare the values in the Vendor Price File Audit to the values in the POLA report for the same PO Files. Tests included in this group are:

* + Test\_VendorPriceFileAudit\_FileStatisticalReport – Test Case 4023

## WorkQueueReports These tests compare the values and calculations in the Work Queue reports to the values in the POLA reports for the same PO Files. Tests included in this group are:

* + Test\_WorkQueue\_ExceptionTypes – Test Case 45
  + Test\_WorkQueue\_FieldRemoval – Test Case 47
  + Test\_WorkQueue\_ItemDescription – Test Case 49
  + Test\_WorkQueue\_NeutralizingABenchmark – Test Case 90
  + Test\_WorkQueue\_ReportCompleteness – Test Case 96
  + Test\_WorkQueue\_ResetChanges – Test Case 46
  + Test\_WorkQueue\_UpdateItemNumber – Test Case 52
  + Test\_WorkQueue\_UpdateManualItemNumber – Test Case 51
  + Test\_WorkQueue\_UpdateSupplierReorderNumberSame – Test Case 48
  + Test\_WorkQueue\_ZeroOutABenchmark – Test Case 89