HAAGA-HELIA
University of Applied Sciences

Automating UI Tests for a Web Application Using Test-Complete

Olga Shakurova

Bachelor's Thesis
Degree Programme in
Business Information
Technology
2015

HAAGA-HELIA
University of Applied Sciences

Abstract

Date 28 February 2015

Author(s) Olga Shakurova	
Degree programme Business Information Technology	
Report/thesis title Automating UI Tests for a Web Application Using TestComplete	**Number of pages and appendix pages** **35 + 3**

The goal of this thesis is to discover how to make UI test automation successful and reveal benefits it provides for software development. The study is based on the smoke test automation project. It was completed for the case company product using TestComplete.

Test Automation is an activity which aims to save testing time, increase coverage and hence promote efficient software testing. TestComplete is a popular commercial test automation tool for a wide range of application types, including web applications.

The system under test is a web-based document management system for complex industrial projects.

The objectives of the study are to provide some guiding material about automating user interface tests for a web application using TestComplete with clear examples, images, code blocks, and familiarize testers with problems encountered during the project implementation and ways to avoid them.

The study process started in September 2014 and ended in February 2015. The thesis is written based mainly on the author's personal experience as well as the Internet resources, such as the SmartBear official site, e-books, articles.

To introduce the subject of the paper, the theoretical background is presented first. It is then followed by the project description and findings. The theoretical part includes information about software testing and test automation in general, and then focuses on the TestComplete theoretical basis. Modular scripting and data-driven testing approaches were combined during the implementation process.

The thesis resulted in a step-by-step description of automating user interface tests with TestComplete. It can guide testers through the test automation process. It can also help quickly understand the basic concepts of the tool, the benefits TestComplete and test automation provide and possible problems that may arise during automation. It gives tips for creating more reliable test scripts. The tangible result of the thesis is the TestComplete project which enables to run tests automatically when the case system needs that.

Keywords
test automation, software testing, TestComplete

Table of contents

Terms

Data-driven Testing	A scripting technique that stores test input and expected results in a table or spreadsheet, so that a single control script can execute all of the tests in the table. Data-driven testing is often used to support the application of test execution tools such as capture/playback tools (ISTQB 2014)
Defect	A flaw in a component or system that can cause the component or system to fail to perform its required function, e.g. an incorrect statement or data definition (ISTQB 2014)
Distributed testing	Running tests that consist of several parts executed on different workstations and interacting with each other (SMARTBEAR 2014 a)
Failure	Deviation of the component or system from its expected delivery, service or result (ISTQB 2014)
Functional Testing	Testing based on an analysis of the specification of the functionality of a component or system (ISTQB 2014)
Load Testing	A type of performance testing conducted to evaluate the behavior of a component or system with increasing load, e.g. numbers of parallel users and/or numbers of transactions, to determine what load can be handled by the component or system (ISTQB 2014)
Pass	A test is deemed to pass if its actual result matches its expected result (ISTQB 2014)
Regression Testing	Testing of a previously tested program following modification to ensure that defects have not been introduced or uncovered in unchanged areas of the software, as a result of the changes made. It is performed when the software or its environment is changed (ISTQB 2014)
Test Automation	The use of software to perform or support test activities, e.g. test management, test design, test execution and results checking (ISTQB 2014)
Test Automation Tool	Software used for test automation
Test Case	A set of input values, execution preconditions, expected results and execution postconditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement (ISTQB 2014)
Test Script	Commonly used to refer to a test procedure specification, especially an automated one (ISTQB 2014)

1 Introduction

1.1 Background

Software is part and parcel of the modern life, it is tightly tied to human everyday activities, inter alia, business, science, engineering, law, medicine. Software failures can cost millions. Creating a high quality software product requires thorough testing. It is a labour-intensive task, and automation in many cases can ease a tester's workload.

This thesis topic came into existence from the need to automate a smoke test suit for a web application. The software is a document management system for complex industrial projects which has integrations with Microsoft Office and Autodesk AutoCAD.

A lot of software tools imitating users' actions have been created to help automate testing. Since the owner of the web application had TestComplete installed and licensed and was planning to use it as a test automation tool for other company products, a decision was made to proceed with TestComplete in this project as well.

According to Gennadiy Alpaev, today TestComplete is one of the most popular commercial tools. Besides a wide variety of applications, such as .NET, Android, Delphi, Java, iOS, Windows etc., it gives testers an ability to create automated tests for Web applications. (Alpaev 2013, 7.)

1.2 Goals

The main objectives of the thesis are to provide a guiding material about automating user interface tests for a web application using TestComplete, and familiarize testers with problems encountered during the project implementation and ways to avoid them. The process of creating automated test scripts with TestComplete, where the actual and expected outputs are compared, running them and deciding whether the tests have failed, will be described.

The purpose of the thesis is to discover how to make UI test automation successful and reveal benefits it provides for software development. The thesis can be used by testers to get oriented in TestComplete and learn basic things that should be considered during test automation.

1.3 Thesis structure

The document is divided into chapters. The second chapter introduces some basics of automated software testing. The third chapter describes the main concepts of the automation tool. It is divided into sections, each revealing a certain feature of the tool. The fourth chapter is devoted to the project implementation: tables, images and code blocks, which are included whenever needed, give a better understanding of the implementation process. The layout is performed according to Writing reports and theses at HAAGA-HELIA.

2 Software testing

Since this thesis focuses on automated software testing, it is important to know the theory of the subject. Software testing is a huge field of knowledge. Different techniques, methods, types of Software testing have been contrived and classified.

Covering the vast testing theory is out of scope of this paper. A lot of weighty books have been written on the subject. This chapter briefly presents some basics of software testing in general and automated testing in particular.

2.1 Manual vs. automated testing

In many cases software tests are carried out manually. Manual testing has a number of strengths:

- Manual testing can be useful when a test case needs to be run only once or twice. As a rule, creating an automated test is more time-consuming i.e. expensive than running it once manually (Marick 1998, 3)
- Human testers are good in finding the strangest scenarios and hence finding defects by using the system in an unexpected way
- Human testers can observe, which may be useful if the goal is user-friendliness, improved customer experience (Apica 2014) or overall design

According to Hayes, test automation is a long term strategic activity. The goal is to increase test coverage, not to cut testing costs. Investments in planning, training, development are necessary before any benefits emerge. The lack of professionalism or expertise in testing cannot be compensated by any automation tool. (Hayes 2004, 22-25.)

The basic benefits of automated testing are:

- Automated tests are faster and more precise as compared to time consuming and error-prone manual tests
- Automated tests can be run over and over again at any time
- Test Automation is essential when it comes to heavy user workloads. It would be hardly possible to make, for example, 350 testers use the product simultaneously and, of course, not cost-effective. Some tests such as load tests need to be automated (Marick 1998, 2.)

Considering the above, automation is not a pure replacement of manual testing. The two approaches can be used complementarily.

2.2 Levels of testing

Starting software testing with testing the entire system would be a fatal mistake that could lead to the product failure. As Prasad (2009, 75) assures, a practical software testing approach is to divide a testing process into different levels. A division offered by Tugberk Ugurlu et al. (2013) gives the best fit to the context of test automation. The authors advise to keep the slanted pyramid shown in Figure 1 in mind when writing and automating tests.

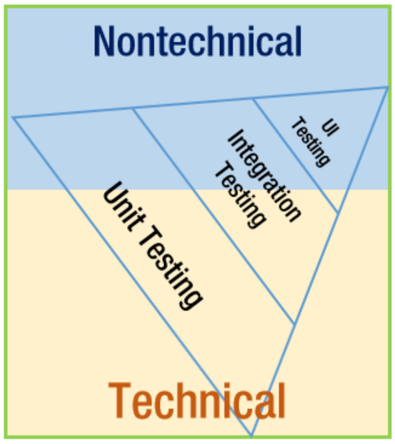


Figure 1. A healthy testing pyramid (Ugurlu, Zeitler & Kheyrollahi, 2013, 451)

A unit is usually the smallest block of code which performs a function. Unit testing is by its nature mostly automated (Laukkanen 2006, 6). Unit testing is usually done by developers themselves to test the code they write. Each unit must be tested in isolation. If it is not possible to test unit separately an additional piece of code may be written (Prasad 2009, 75).

Integration testing guarantees that the unit can work not just individually. It is about testing how different parts of the system interact with each other. Tugberk Ugurlu et al. (2013, 450) state that unit tests are quite easy to automate, whereas integration tests need a bit more setup because they are run in an environment.

Unit and Integration testing do not make user interface testing unnecessary. UI testing is about testing software at the user interface level. It assesses whether user interface is correctly bound to the system, i.e. whether it returns the right output for user input. Graphical user interface tests are rather hard to automate successfully (Tugberk Ugurlu & al. 2013, 451; Laukkanen 2006, 7), i.e. make them reliable and maintainable.

The pyramid (Figure 1) shows that unit tests should dominate. The number of UI tests is supposed to be the least. Nontechnical members of the team can define a big part of UI and integration tests (Ugurlu & al. 2013, 451).

A defect can be discovered at any level of testing but the sooner the better. It may happen during the development process or on the day the product is delivered to the customer. Of course, in some cases the latter can be catastrophic, but quite often defects continue to be detected even in post-release. Fixing defects found in earlier phases is generally considered to cost less.

This paper focuses on automating user interface tests, namely smoke test. According to Andreas Spillner et al. (2014, 144), a smoke test traditionally verifies the minimum reliability for the test software, concentrates on checking its main functionality, and the output of the test is not evaluated in detail.

3 TestComplete

This chapter introduces TestComplete and its core features. The chapter does not compare TestComplete to other automation tools nor tries to cover everything about the tool. It mainly focuses on the features needed for the project implementation. The content is based on the extensive official TestComplete documentation, other online sources and author's personal experience.

3.1 Overview

As mentioned above, the development environment, which was chosen for the project implementation is TestComplete, a popular commercial test automation tool for a wide range of application types owned and developed by SmartBear Software.

TestComplete provides a TestComplete platform and technology modules (Figure 2). The Platform enables creating, maintaining, and executing automated tests, and modules make testing possible across desktop, web and mobile. Each module, i.e. Desktop, Web and Mobile, includes features that allow creating automated tests on the specified platform. (SMARTBEAR 2014b.)

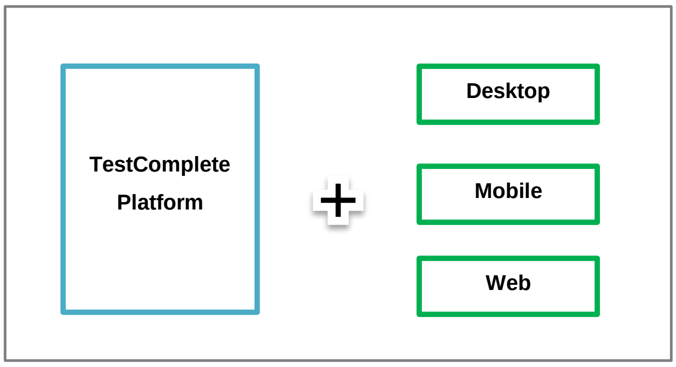


Figure 2. TestComplete Platform and Modules (SMARTBEAR 2014b)

TestComplete focuses on both, functional and unit testing. It can be used in regression testing as well as in other kinds of testing types: data-driven, distributed, load, even manual testing etc. (SMARTBEAR 2014c.)

In TestComplete it is possible to create keyword-driven tests and scripts. The former consist of keywords, each of which corresponds to some action, such as open file, set text, click button etc. The latter are usually functions, written in one of the scripting languages TestComplete supports.

Keyword-driven tests and scripts can be recorded with recording tool or created from scratch. Creating keyword tests does not require any programming skills and is quite easy. Scripting requires some basic programming knowledge but allows additional flexibility in tests. TestComplete supports VBScript, JScript, DelphiScript, C++Script and C#Script. (SMARTBEAR 2014d.)

3.2 Web browser configuration

A special Web Browser configuration is needed to make test scripts runnable with TestComplete. SmartBear Software offers a detailed instruction for the main modern Web Browsers: Google Chrome, Internet Explorer, Mozilla Firefox, Opera, on their web site (http://support.smartbear.com/viewarticle/56974). Setting Web Browser correctly enables recording test scripts and, which is more important for the project, playing them back. A tester can spend a lot of time trying to figure out why the existing scripts are not played back, and then find out that something in the Web browser configuration has been changed.

3.3 TestComplete project

Test scripts are created in a TestComplete project. A tester should specify a type of the application under test, choose settings for a Test Vizualizer and a scripting language during the creation process.

TestComplete Test Visualizer catches information about test actions. Depending on the chosen settings, this information can contain screen shots or screen shots along with test object data they contain. It helps to ’visualize’, or better understand, the actions performed during the test, which is useful, especially for beginners. It is also possible to disable the Visualizer. This is what Alpaev (2013, 116,168) recommends to do as creating screen shots is time consuming, and it is not necessary to capture all screen shots. In most cases it is redundant.

There is an option to post image on error in the project editor. If an error occurs a screen shot is sent to the test log. This property was enabled during the project implementation.

Choosing a scripting language is an important matter if test scripts are planned to be created. It will not be possible to change it in the future. If someone would want to change the selected project's scripting language, only redoing the project from scratch would work (Alpaev 2013, 13).

3.4 JScript

One of the scripting languages suggested by TestComplete is JScript. A brief description of the language is briefly given in this section as it was used during the project implementation phase. The name is quite similar to JavaScript, the most widely used programming language on the planet. But is JScript actually JavaScript?
ECMA International - the European association for standardizing information and communication systems - delivers a standardized, international programming language based on core JavaScript. This standardized version of JavaScript, called ECMAScript, acts the same way in all applications that support the standard. The open standard language can be used for developing implementations of JavaScript. The ECMAScript standard is documented in the ECMA-262 specification. The ECMA-262 standard is also approved by the International Organization for Standardization as ISO-16262 (Mozilla Development Network).

Microsoft implemented its own version of the language called JScript. The company develops it separately from the ECMAScript standard. JScript and JavaScript are often confused with one another and produce many problems on the browser side (Düüna 2012, 9). However, Stephen Chapman (2013) assures that in spite of some differences, the two languages can be considered to be equivalent to one another.

3.5 Name mapping

TestComplete operates on the object level; it reads the object properties from the application. When the test is run, TestComplete uses the identification information about the objects, which is stored in the project, to find them on the web page. Every project has a NameMapping.tcNM file which contains a name mapping scheme aka Name Mapping repository. All objects data, such as, objects properties, names, types, identifiers etc., is saved there and organized into an object hierarchy, namely Mapped Objects tree (appendix 1). The data is automatically added to the repository during test recording or can be added manually when a test is being created from scratch. The fact that the object information and actual tests are separated from each other makes tests more stable and easier to maintain, as no modifications in the tests are needed in case the application is changed.

A special property, Extended Find, was often enabled during the project implementation. This feature can be applied to mapped objects and is used if an object has a dynamic parent which is not mapped. It enforces TestComplete to look for a certain object all levels down in the object hierarchy. (SMARTBEAR 2014e.)

3.6 Aliases

Besides a Mapped Objects tree, there is an Aliases tree in the Name Mapping repository (appendix 1). Each mapped object has a corresponding alias, which is just a descriptive name. It is used in scripts to refer to the mapped object.

The Mapped Object tree represents the exact hierarchy of objects in applications. It is necessary for finding objects on the web page. As for Aliases, it is reasonable in many cases to shorten the Aliases hierarchy by dragging them up the tree. An object can then be referred in a script without excluded aliases.

A descriptive name and a shortened hierarchy make scripts more readable. (SMART-BEAR 2014f.)

3.7 Object Browser

Object Browser is an essential TestComplete tool (appendix 2). It shows all the processes and objects visible for TestComplete, which are workable from within TestComplete. The Sys node is the root object. Object properties and methods are displayed for a selected item in the Object Browser window (Alpaev 2013, 29). Object Browser is a handy tool for adding Objects to the Name Mapping Repository.

3.8 Object Spy

As Gennadiy Alpaev (2013, 32) truly states, locating the necessary element in Object Browser may be quite challenging. To simplify the task, the Object Spy utility can be used (appendix 3). Using the Object Spy, any object on the screen that is visible for TestComplete can be selected, its properties and methods can be viewed, and it can be easily found in the Object Browser object tree. It can also help to add the object to the Name Mapping repository or check if it is already there (SMARTBEAR 2014g).

3.9 Data-driven testing

A lot of test data has to be inputted while performing functional testing, many data combinations must to be checked in order to verify that the system works correctly and gets the right output. One of the possibilities is to include the test data into scripts. In this case, if the data has to be changed, the test script must be updated. This is not always an easy task especially if a tester does not have programming skills or the test script is long and complex. Pekka Laukkanen assures that updating scripts, with the test data embedded, may become a real problem, and a small change in the system under test may require changing all scripts (Laukkanen 2006, 23,24).

To avoid this, TestComplete supports the approach known as data-driven testing, which implicates storing and reading test data from external sources. An external source may be an Excel file, a CSV file, a database etc. TestComplete provides a special object, DDT, for accessing them. (Alpaev 2013, 203.) Separating test data from the code makes test scripts easier to modify and maintain.

3.10 Test management

Project editor is a TestComplete project management tool for creating and choosing test items to be executed, viewing and modifying project variables, logs and properties.
Test Item is a TestComplete project element, which is used to manage tests launches. Each test item has properties which provide easier management, such as, the Enable checkbox, Count, Timeout etc. The Enabled checkbox allows excluding test items from being run. The Count column allows launching the test several times if needed, the Timeout column sets the expiration time when the test will be stopped and a corresponding message will be sent to the test log. (Alpaev 2013, 102.)

By the end of the test run, the findings should be reported to the project team and management. Reporting can be also automated by TestComplete. After a test is executed, a detailed test log of all performed actions is generated. It provides a list of passed and failed tests with the description of each operation and reasons of possible failures. It is also possible to export results and view them in a web browser. (SMARTBEAR 2014h.) This does not require TestComplete and gives an opportunity of sending results to anyone interested or store them separately with other test results.

4 Project implementation

This chapter describes the implementation process of the smoke test automation for a Web application by describing the work phases, used techniques and sample test cases. The problems encountered during the project implementation and important issues a tester should take into consideration while automating tests are then presented in the Summary section.

4.1 Why automate?

Before getting a comprehensive view of the testing automation procedure, it is important to understand why it was considered to be necessary. Not all tests should be automated. Test automation can be considered not cost effective. For example, automating some test and running it only once will cost more than simply running it manually. However, smoke test is something that is usually automated. (Marick 1998, 2,4.) As Andreas Lundgren (2008, 15) writes, the purpose of automating functional testing is to replace repetitive and often error-prone manual testing. As a rule, smoke test is run often. It aims to verify the basic functionality of the software and validate code changes.

Some applications, such as weather-mapping system or one that relies on real-time data, are unstable by design. In this case, automation will be difficult because the expected results are not known. (Hayes 2004, 15.)

Since the application under test is quite stable in design, the expected outputs are always known. Smoke test is run on a regular basis and involves a lot of user’s input, which can be inaccurate. Moreover, running smoke test manually is a time-consuming task; it takes approximately two working days to complete the tests.

These facts formed an opinion that smoke test automation was reasonable and that it could be cost-effective for the case company.

4.2 Methods

Different automation approaches exist, such as linear scripting, modular scripting, data-driven, keyword-driven, record and playback testing. Some of them have been already mentioned in this paper. The following methods were combined during the implementation process: modular scripting and data-driven testing.

The data-driven approach and some of its benefits have been discussed in paragraph 3.9. Test data is separated from scripts, which makes test easier to edit and maintain. One script can run many similar test cases.

As opposed to linear scripting, where non-structured scripts directly interact with an application under test, modular scripting consists of modules or functions corresponding to application functionality, for example, navigating or logging into the application. Modules organize a test library, and test scripts can be constructed by combining these modules with each other. In case of modularity creating of new tests is faster as it is possible to reuse scripts. (Klärck.)

After learning the TestComplete basic theory, creating keyword tests and scripts, recording and creating tests from scratch, consulting with experienced TestComplete users, a decision was made to create test scripts from scratch using JScript. This choice was considered to be the most rational, because as it was already mentioned, scripting can help to create flexible tests. Using recorder seemed, at first, to be the easiest way to create scripts. In fact, recording gives a brilliant opportunity for beginners to study scripting tests with TestComplete. But, as Al-Zain, Eleyan & Garfield (2012, 2) assure, creating and running tests with the recorder tool, is weak when it comes to dynamic web pages. Almost all recorded tests fail to run during playback. TestComplete engine is not able recognize objects as the object identification information changes each time the page is loaded. The records need further editing in order to keep them maintainable. A tester has to understand each line of the code in the script, edit mapping if not done before the recording or done incorrectly, then edit the script. It is very time-consuming because it requires too much editing. It is much easier to utilize the API of TestComplete and create own test scripts from scratch.

JScript was chosen as the scripting language for this project mainly due to the popularity of JavaSctript. And as Alpaev mentions (2013, 14), it is a powerful and flexible language with a compact syntax.

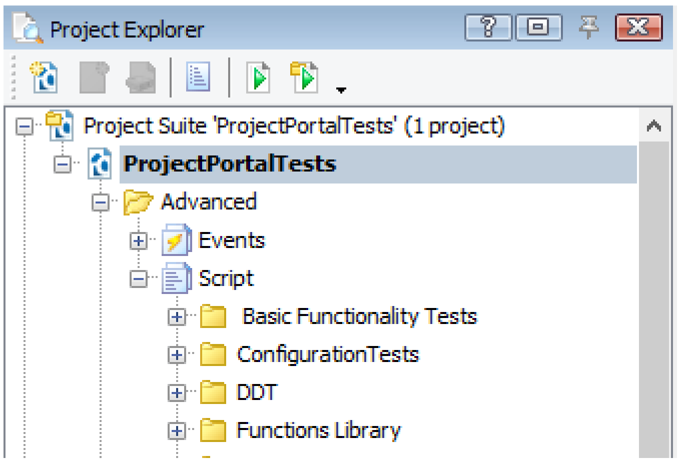
4.3 Software

Concerning the tasks set in the thesis, the following software was used:

- Microsoft Windows 8.1 Enterprise x64
- SmartBear TestComplete 10.20
- Mozilla Firefox 29

4.4 Project structure

In order to attain thesis objectives, the case Web application smoke test was automated. Test scripts were created in TestComplete, the ProjectPortalTests project. Figure 3 covers the final structure of the project. Basic Functionality Tests and Configuration Tests are folders that contain automated test cases. The Functions Library folder includes the routines they use. They can be also used by other functions of the same level. The DDT folder consists of functions, which are used for data retrieval.
The final structure looks quite simple. However, it required time and effort to make it logically and functionally applicable.



Project Suite 'ProjectPortalTests' (1 project)
ProjectPortalTests
Advanced
Events
Script
Basic Functionality Tests
ConfigurationTests
DDT
Functions Library

Figure 3. Project structure

4.5 Sample test cases

In order to understand how to automate tests with TestComplete, a process of creating first test scripts presented in Table 1 will be described.

Table1. Sample test cases

No	User	Operation	Expected output
1.	admin	Log on to web application with an admin user.	The user is successfully logged in.
2.	admin	Create a workspace by clicking the 'Create' button. Enter the following:	The workspace properties view is brought up.
		Template selection: Default project,	
		Workspace name: First project,	
		Code: FP,	
		Type: *Delivery* and	
		Status: *Pilot*.	
		Save the project by clicking Create.	Values are saved properly.

4.6 Name mapping

The first step was name mapping. During the project implementation adding an object to the Mapping repository typically consisted of the following steps:

1. Open a web page with an object that is to be found during the test run in the Browser
2. Click *Object Spy* button in TestComplete
3. Drag the target to point to the object -> the Object Spy window opens
4. Press *Highlight object in the Object tree* button
5. Right click object in Object Browser ->Choose *…Map object*
6. Choose unique properties for identifying the object

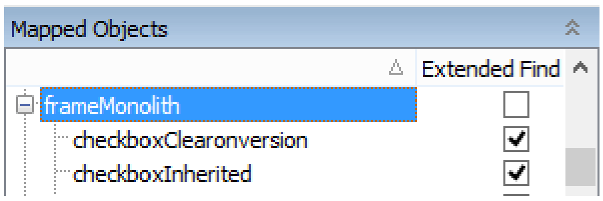
A list of objects and identification properties that were necessary for automating the first sample test case is presented in Table 2.

Table 2. Mapped objects for login

Alias name	Properties (name: value)
browser	ObjectType: Browser
pageProjectPortal	ObjectType: Page URL: *application URL*
formLoginform	ObjectType: Form IdStr: loginform
passwordbox	ObjectType: PasswordBox ObjectIdentifier: pwd*
tbUsername	ObjectType: Textbox ObjectIdentifier: uid
submitbtLogin	ObjectType: SubmitButton ObjectIdentifier: action_Login

There are some techniques that were used to map the objects during the project implementation. One of them is used, for example, to map the passwordbox object in Table 2. TestComplete can handle wildcarding; it supports standard * and ? wildcards. Wildcarding is used for mapping objects with a dynamically generated identifier i.e. every time the test is run, that particular property is going to get a new value. The passwordbox object's property ObjectIdentifier is generated into something like pwd49545445, where the pwd part is constant and the number is dynamic. Using the asterisk wildcard in pwd* enforces TestComplete to pay attention only to the constant part while searching for the password box.

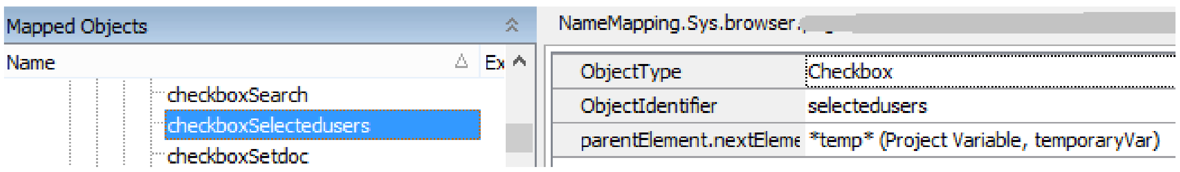
As mentioned above, the Extended find feature is used when mapping objects whose direct parent is not stable. In this case, the object is dragged up the Mapped Objects tree to the most stable parent and the Extended Find checkbox is checked. In the example shown in Figure 4, the checkboxes are moved up next to the frameMonolith object as they can appear in different places but always within this frame. The search will be performed all levels down the stable parent.


Mapped Objects

	Extended Find
frameMonolith	☐
checkboxClearonversion	✓
checkboxInherited	✓

Figure 4. Extended Find

In some cases, an object's only unique property is not known and its value can be figured out only when the test is being run. Then, a project variable can be used as the property value. It can be, for example, manually created in the Project Editor, on the Variables page. In Figure 5, the initial value of the temporaryVar project variable is assigned to one of the checkboxSelectedUsers object identifiers.


Mapped Objects

Name	Ex
checkboxSearch	
checkboxSelectedusers	
checkboxSetdoc	

NameMapping.Sys.browser.[redacted]

ObjectType	Checkbox
ObjectIdentifier	selectedusers
parentElement.nextEleme	*temp* (Project Variable, temporaryVar)

Figure 5. Project variable temporaryVar

The necessary property value is assigned during the test run and the needed checkbox can be found and checked as Figure 6 demonstrates.

```
      function addUserToWs(username)
      {
         …
         Project.Variables.temporaryVar = username;
         frame.chkSelectedUsers.ClickChecked(true);
         …
      }
```

Figure 6. Assigning value to a project variable

4.7 Functions library

After the name mapping for the sample test cases was done, TestComplete could perform the required actions on the objects from scripts. Creating the functions, presented below, was the next step, which later allowed their repeated usage in the actual test scripts. The function that launches Firefox is shown in Figure 7.

```
    function launch(url)
    {
       while (Sys.WaitBrowser().Exists)
         Sys.WaitBrowser().Close();
      Browsers.Item(btFirefox).Run(url);
    }
```

Figure 7. Launch

The Sys.WaitBrowser() method enforces TestComplete to wait until some browser process exists in the system. If the process is found, the method returns the respective Browser object i.e. a running web browser. The Exists property checks if a browser was found. (SMARTBEAR 2014i.) To close the browser, the Close() method is called. It is emplaced within a while loop to close all the instances of running browsers. To launch a browser, the Browsers.Item(btFirefox).Run(url) method is used. The url parameter enforces a specified tested web page to be opened.

A user's logging in to the application using the appropriate information is shown in Figure 8.

```
    function login(username, password)
    {
      var page = Aliases.browser.ProjectPortal;
      var loginForm = page.formLoginform

      if(loginForm.Exists)
        {
          loginForm.tbUsername.SetText(username);
          loginForm.passwordbox.SetText(password);
          loginForm.submitbtLogin.Click();
          page.Wait();
          if(page.FindChild(["ObjectType", "contentText"],["Panel", "Wrong username or
    password, please try again."], 10).Exists)
            Log.Message("Login has failed (wrong username or password).");
         else if(page.FindChild(["ObjectType", "contentText"],["Link", "Workspaces"], 10
      ).Exists)
            Log.Message(username + " has been successfully logged in.");
          else
```

```
            Log.Error("Login has failed.");
        }
      else
          Log.Message("formLoginform not found");
    }
```

Figure 8. Login

The Aliases object provides access to the Aliases tree and its objects in the Name Mapping repository.

As it logically implies from the names, the SetText() action sets the specified text to a control, while Click() simulates a left mouse button click.

After a specified button is clicked, the Wait() method is used that instructs TestComplete to wait until the page is loaded and TestComplete is able to find the necessary objects on the page and continue executing the test.

The FindChild method then searches for a child object with the specified properties (””ObjectType”, ”contentText”) that have the specified values (”Panel”, ”Wrong username…”) in the object hierarchy, and returns it if the object is found. Ten is depth, a number that tells down to what level of child objects of the *page* object the method should search for a specified object. It's zero by default meaning that the search should be done only in child, not in grandchild etc. objects (SMARTBEAR 2014j).

TestComplete also provides a test log and allows posting there different types of messages: events, ordinary messages, warnings, errors etc., using the Log object and corresponding methods. In the above example, Log.Error() and Log.Message() methods are used to indicate test pass or failure.

The function presented in Figure 9 demonstrates a process of creating a workspace.

```
function createWorkspace(workspace)
 {
    var page = Aliases.browser.pageProjectPortal;
    var wsForm = page.formMetaform;
    var star = "*"
     var nameTemplateString =
  star.concat(workspace.templateSelection, star);

  /*go to workspaces tab*/
  var btnWorkspaces = page.btnWorkspaces;
  btnWorkspaces.Click();
  page.Wait();

  /*click Create*/
  var btnCreateWorkspace = page.btnCreateWorkspace;
  btnCreateWorkspace.Click();
```

```
        page.Wait();

          if(workspace.templateSelection)
          {
            if(wsForm.WaitAliasChild("slWorkspaceTemplate",
            10000).Exists)
              wsForm.slWorkspaceTemplate.ClickItem(nameTemplateString);
            else
              Log.Error("The select list for workspace template has not
              appeared.");
          }
          if(workspace.includedItems)
          {
            if(wsForm.WaitAliasChild("slIncludeIntoWorkspace",
            10000).Exists)
              wsForm.slIncludeIntoWorkspace.
              ClickItem(workspace.includedItems);
            else
              Log.Error("The select list for included items has not
              appeared.");
          }
          if(workspace.name)
          {
            if(wsForm.WaitAliasChild("tbWorkspaceName", 10000).Exists)
              wsForm.tbWorkspaceName.SetText(workspace.name);
            else
              Log.Error("The text box for workspace name has not
              appeared.");
          }
          …
          wsForm.btCreateWorkspace.Click();
          page.Wait();
      }
```
Figure 9. Create workspace

The WaitAliasChild() method is used to pause the test execution until the object with property values of some alias, for example the slWorkspaceTemplate select list, appears on the page or until the time limit of 10000 milliseconds is reached. The method does not post an error message to the test log if case the object does not appear. The Exists property helps to detect whether the object is found in the system or not. (SMARTBEAR 2014k.)

The checkWorkspaceProps function presented in Figure 10 checks if the object properties on a web page have specified values. This is done to verify that the values of the workspace properties were saved properly.

```
function checkWorspaceProps(workspace)
{
    var page = Aliases.browser.pageProjectPortal
    var frame = page.frameMonolith;
    var returnBool = true;

    frame.linkEditWs.Click();
    if(workspace.name)
    {
      if(!aqObject.CheckProperty(frame.tbEditWsName, "Text",
      cmpEqual, workspace.name))
        returnBool =  false;
      }
    if(workspace.code)
    {
      if(!aqObject.CheckProperty(frame.tbEditWsCode, "Text",
      cmpEqual, workspace.code))
```

```
          returnBool =  false;
        }
      if(workspace.type)
      {
        if(!aqObject.CheckProperty(frame.slEditWsType, "wText",
        cmpEqual, workspace.type))
          returnBool =  false;
      }
      if(workspace.status)
      {
        if(!aqObject.CheckProperty(frame.slEditWsStatus, "wText",
        cmpEqual, workspace.status))
          returnBool =  false;
      }
      frame.btEditWsCancel.Click();
      return returnBool;
   }
```

Figure 10. Check Workspace Properties

Property checkpoints help to verify that an object property has the expected value. A aqObject object and a CheckProperty method are used to perform a property check-point. The object just provides a range of methods for objects (SMARTBEAR 2014l), and its name can be actually skipped. In aqObject.CheckProperty(frame.tbEditWsName, "Text", cmpEqual, workspace.name), for example, frame.tbEditWsName is the object whose property is checked, “Text” is the name of the property to be checked, work-space.name is the expected property value and cmpEqual means that the method must check whether the expected property value equals the actual property value. Other test conditions, such as ‘contains’, ‘less than’ etc., were also used during the project implementation.

4.8 Input data for test scripts

To simplify tests maintenance by keeping input data separated from test scripts, the data-driven approach was used for storing input data in the project. Excel sheets were chosen as the data storage due to the format convenience, i.e. a table structure with rows and columns.

The following test data was needed for the sample test cases: url to test, username and password to login, workspace template, name, code, type, status. In order to execute the tests using data-driven approach, the Excel sheets and tables presented in Tables 3-5 were created in the testCasesConfig.xlsx file.

Table 3. Sheet UrlUnderTest

URL
application URL

The column name URL is placed into the first row to make test data easier to understand.

Table 4. Sheet 1

Username	Password
admin	*password*

Table 4 provides the input data for a user's logging in to the application.

Table 5. Sheet 2

Template selection	Included items	Name	Code	Type	Status	Description	Icon URL	Home page URL	Logo URL	Administrator	Start Date	End Date	Default properties
Default project		First workspace	FP	Delivery	Pilot								

Table 5 contains the data for creating a workspace. Only a few of the possible property values were needed for the sample test case. However, all properties were included in the table to make different combinations of data possible in further testing.

To read the data from Excel tables the functions shown in Figures 11, 12 were created in the DDT folder.

```
 function readLoginData(fileName, sheetName, aceDriver)
{
  var Driver = DDT.ExcelDriver(fileName, sheetName, aceDriver);
  var loginsData =[];
  while(!Driver.EOF())
  {
    var loginData = {
              user: Driver.Value(0),
              pwd:Driver.Value(1)
              };
    loginsData.push(loginData);
    Driver.Next();
  }
  // Closes the driver
  DDT.CloseDriver(Driver.Name);

 return loginsData;
}
```

Figure 11. Read Login Data from Excel

The ExcelDriver method provided by TestComplete helps to access data in Excel sheets. In the function above, it obtains values stored in the file filename, sheet sheetName. The aceDriver parameter specifies whether TestComplete should use the AceDriver for connection (true) or not (false). If it is false TestComplete uses the ODBC driver which has no support for Excel 2007 – 2013 (SMARTBEAR 2014m).

The tables for the sample test cases are single-row tables. In order to make the function more flexible and possible to reuse in other test cases, where the amount of rows in the tables may vary, the data from the file is read into an array. The while loop helps to iterate through the rows of the file and add new elements to the loginsData array.

The readWorkspaceData function in Figure 12 retrieves the data for creating workspaces and similarly pushes it into an array.

```
function readWorkspaceData(fileName, sheetName, aceDriver)
{
  var Driver = DDT.ExcelDriver(fileName, sheetName, aceDriver);
  var wssData =[];
  while(!Driver.EOF())
  {
    var wsData = {
              templateSelection:Driver.Value(0),
              includedItems:Driver.Value(1),
              name:Driver.Value(2),
              code:Driver.Value(3),
              type:Driver.Value(4),
              status:Driver.Value(5),
              description:Driver.Value(6),
              iconURL:Driver.Value(7),
              homePageURL:Driver.Value(8),
              logoURL:Driver.Value(9),
                 administrator:Driver.Value(10),
              startDate:Driver.Value(11),
              endDate:Driver.Value(12)
              };
    wssData.push(wsData);
    Driver.Next();
  }
  DDT.CloseDriver(Driver.Name);

  return wssData;
}
```

Figure 12. Read Workspace Data

4.9 Test scripts

In this section, the scripts that execute the specific actions within the web application are presented. Executing them in TestComplete fulfils the actions of the sample test cases presented in Table 1.They apply all the functions created earlier.

```
//USEUNIT Launch
//USEUNIT ReadLoginData
//USEUNIT Login
function adminLogin()
{
  var DriverUrl = DDT.ExcelDriver("../testCasesConfig.xlsx", "UrlUnderTest", true);
  launch(DriverUrl.Value(0));
  var loginsData = readLoginData("../testCasesConfig.xlsx", "1", true);
  login(loginsData[0].user, loginsData[0].pwd);
}
```

Figure 13. TC 1, Login as Administrator

In the above Figure 13, the web browser window is opened and navigated to a certain url, an admin user logs on to the web application, and the message is sent to the test log saying whether or not the login has succeeded. //USEUNIT helps to make references to other units.

```
//USEUNIT ReadWorkspaceData
//USEUNIT CreateWorkspace
//USEUNIT CheckWorspaceProps
function createFirstWs ()
{
  var workspaceData = readWorkspaceData("../testCasesConfig.xlsx", "2", true);
  createWorkspace(workspaceData[0]);
  if(checkWorspaceProps(workspaceData[0]))
      Log.Message("Workspace " + workspaceData[0].name + " has been
      successfully created and its property values checked.");
  else
      Log.Error("Workspace " + workspaceData[0].name + " has been
      created with errors.");
  }
```

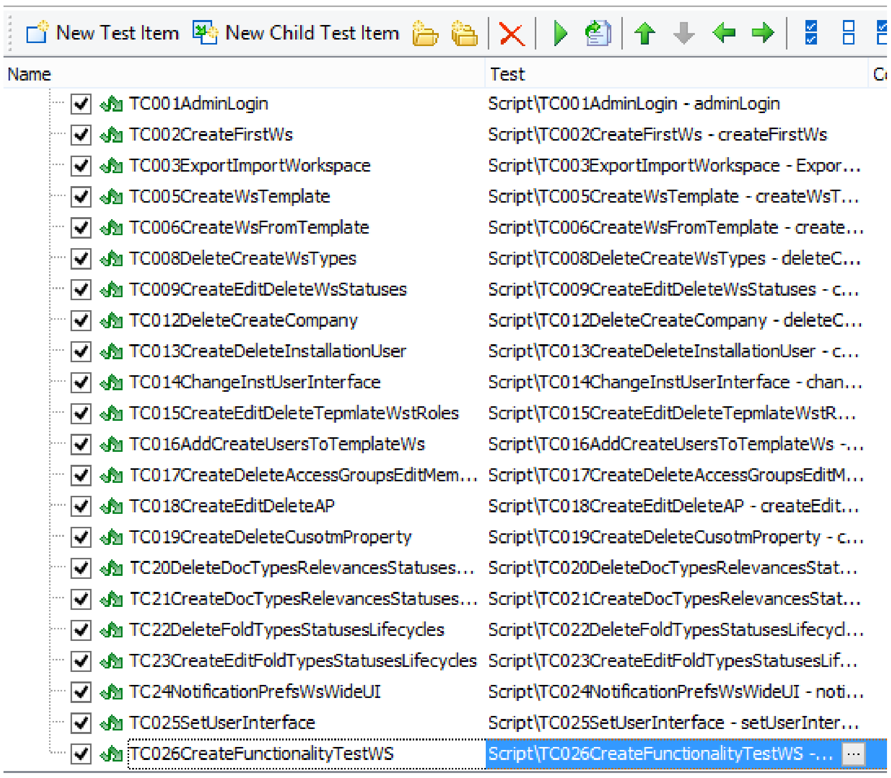
Figure 14. TC2, Create the First Workspace

The test script shown in Figure 14 creates the first workspace and verifies if the creation has been performed correctly.

4.10 Running tests

By the end of the automation process, a list of test scripts was created. In order to manage the ProjectPortalTests project, the TestComplete project editor was used. New test items were created, added to the test list and selected to be run.

The test items for running test scripts are shown in Figure 15. Their names and test scripts names are identical which appears to be logical and helps to avoid misunderstanding.



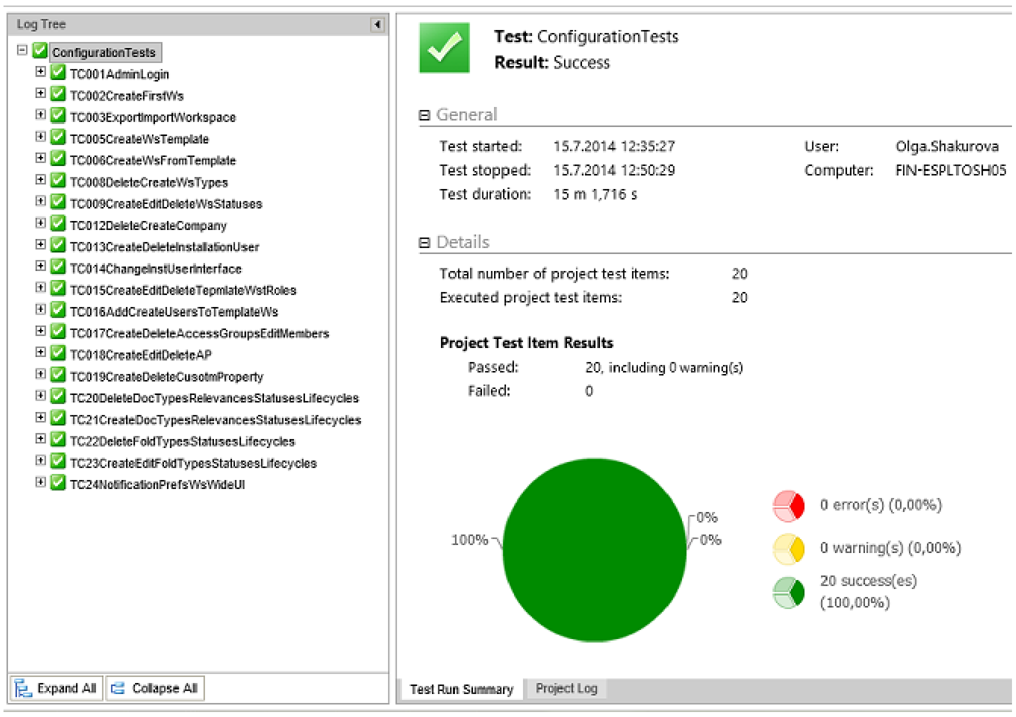
Name	Test
✓ TC001AdminLogin	Script\TC001AdminLogin - adminLogin
✓ TC002CreateFirstWs	Script\TC002CreateFirstWs - createFirstWs
✓ TC003ExportImportWorkspace	Script\TC003ExportImportWorkspace - Expor…
✓ TC005CreateWsTemplate	Script\TC005CreateWsTemplate - createWsT…
✓ TC006CreateWsFromTemplate	Script\TC006CreateWsFromTemplate - create…
✓ TC008DeleteCreateWsTypes	Script\TC008DeleteCreateWsTypes - deleteC…
✓ TC009CreateEditDeleteWsStatuses	Script\TC009CreateEditDeleteWsStatuses - c…
✓ TC012DeleteCreateCompany	Script\TC012DeleteCreateCompany - deleteC…
✓ TC013CreateDeleteInstallationUser	Script\TC013CreateDeleteInstallationUser - c…
✓ TC014ChangeInstUserInterface	Script\TC014ChangeInstUserInterface - chan…
✓ TC015CreateEditDeleteTepmlateWstRoles	Script\TC015CreateEditDeleteTepmlateWstR…
✓ TC016AddCreateUsersToTemplateWs	Script\TC016AddCreateUsersToTemplateWs -…
✓ TC017CreateDeleteAccessGroupsEditMem…	Script\TC017CreateDeleteAccessGroupsEditM…
✓ TC018CreateEditDeleteAP	Script\TC018CreateEditDeleteAP - createEdit…
✓ TC019CreateDeleteCusotmProperty	Script\TC019CreateDeleteCusotmProperty - c…
✓ TC20DeleteDocTypesRelevancesStatuses…	Script\TC020DeleteDocTypesRelevancesStat…
✓ TC21CreateDocTypesRelevancesStatuses…	Script\TC021CreateDocTypesRelevancesStat…
✓ TC22DeleteFoldTypesStatusesLifecycles	Script\TC022DeleteFoldTypesStatusesLifecycl…
✓ TC23CreateEditFoldTypesStatusesLifecycles	Script\TC023CreateEditFoldTypesStatusesLif…
✓ TC24NotificationPrefsWsWideUI	Script\TC024NotificationPrefsWsWideUI - noti…
✓ TC025SetUserInterface	Script\TC025SetUserInterface - setUserInter…
✓ TC026CreateFunctionalityTestWS	Script\TC026CreateFunctionalityTestWS -…

Figure 15. Project Test Items

There are different ways to run tests. It is possible to run them from command line, as a part of VisualStudio or MSBuild projects etc.(SMARTBEAR 2014n). The easiest way is to launch them from the TestComplete IDE by selecting the Run button on the editor’s toolbar.

4.11 Test results

By the end of the test run, a detailed test log and a summary were generated. The Test Run Summary (Figure 16) briefly presents the information on the run: the test group name and status, general information, like test time, duration time etc. The Details section gives an overview of results with a chart that visually demonstrates the number of passed and failed test items.


Log Tree

- ConfigurationTests
  - TC001AdminLogin
  - TC002CreateFirstWs
  - TC003ExportImportWorkspace
  - TC005CreateWsTemplate
  - TC006CreateWsFromTemplate
  - TC008DeleteCreateWsTypes
  - TC009CreateEditDeleteWsStatuses
  - TC012DeleteCreateCompany
  - TC013CreateDeleteInstallationUser
  - TC014ChangeInstUserInterface
  - TC015CreateEditDeleteTepmlateWstRoles
  - TC016AddCreateUsersToTemplateWs
  - TC017CreateDeleteAccessGroupsEditMembers
  - TC018CreateEditDeleteAP
  - TC019CreateDeleteCusotmProperty
  - TC20DeleteDocTypesRelevancesStatusesLifecycles
  - TC21CreateDocTypesRelevancesStatusesLifecycles
  - TC22DeleteFoldTypesStatusesLifecycles
  - TC23CreateEditFoldTypesStatusesLifecycles
  - TC24NotificationPrefsWsWideUI

Expand All | Collapse All

Test: ConfigurationTests
Result: Success

General

Test started:	15.7.2014 12:35:27	User:	Olga.Shakurova
Test stopped:	15.7.2014 12:50:29	Computer:	FIN-ESPLTOSH05
Test duration:	15 m 1,716 s		

Details

Total number of project test items:	20
Executed project test items:	20

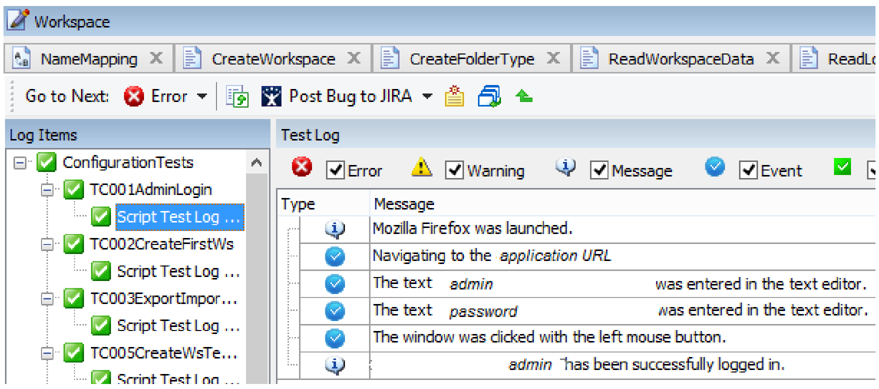
Project Test Item Results

Passed:	20, including 0 warning(s)
Failed:	0

Test Run Summary | Project Log

Figure 16. Test Run Summary

It is also possible to view the details of each test item as shown in Figure 17, as well as to filter the messages by selecting the check boxes of certain types.



Log Items

- ConfigurationTests
  - TC001AdminLogin
    - Script Test Log ...
  - TC002CreateFirstWs
    - Script Test Log ...
  - TC003ExportImpor...
    - Script Test Log ...
  - TC005CreateWsTe...
    - Script Test Log ...

Test Log

☑ Error ☑ Warning ☑ Message ☑ Event

Type	Message
Message	Mozilla Firefox was launched.
Event	Navigating to the *application URL*
Event	The text *admin* was entered in the text editor.
Event	The text *password* was entered in the text editor.
Event	The window was clicked with the left mouse button.
Message	*admin* "has been successfully logged in.

Figure 17. Test Item Log

4.12 Summary

TestComplete provides handy tools like Object Spy, name mapping, Object Browser etc. to create, run automated tests and get comprehensive test reports. However, it is not that straightforward when it comes to automation. The problems described in this section were all encountered during the project implementation.

Name mapping is the essential part of the automation process. In order to identify objects, unique, unchangeable properties should be used for searching. Despite the possibilities

TestComplete gives to make name mapping easier, it is sometimes quite tricky to find such a search criteria. It may be not just ordinary ObjectIdentifier and idStr etc. but something like parentElement.NextElementSibling.contentText or parentElement.parentElement.contentText. The latter property, for example, was used for mapping as it was the only unique property of the object in the list of similar objects. In some cases, when finding a unique identifier is problematic, it is possible to ask a developer to give an object a unique property, for example, id. Each case is specific and there is no universal approach in solving this kind of issues.

The fact that a test script has been run successfully once does not always mean that it is written well enough and the automation process has been finished. The test may pass several times and then fails. This has happened during the project implementation.

Every automation tool has peculiar properties and it is better to take them into account during the automation process. TestComplete is not an exception. In the sample test case the Wait() method is called. As it is stated on the SmartBear official site, The Wait() method checks whether the loading process has been finished and the following commands are executed after the page has been completely loaded. It is then mentioned that it works for simple pages but for complex pages, including dynamic pages, the method may return a value before the page is loaded completely. (SMARTBEAR 2014o.) TestComplete then starts accessing onscreen objects even though they are not yet loaded. This causes an error and the test run fails. The sample Login test case passed using this method many times (Figure 18) but then a series of failures followed, because TestComplete was looking for the Workspaces link when it was not yet loaded.

```
…
page.formLoginform.submitbtLogin.Click();

page.Wait();

    if(page.FindChild(["ObjectType", "contentText"],["Panel", "Wrong username or
password, please try again."], 10 ).Exists)
      Log.Message("Login has failed (wrong username or password).");
    else if(page.FindChild(["ObjectType", "contentText"],["Link", "Workspaces"], 10
).Exists)
      Log.Message(username + " has been successfully logged in.");
    else
      Log.Error("Login has failed.");
…
```
Figure 18. Wait method

Fortunately, there are a few workarounds that can help to avoid failures. The Refresh method can be called to force TestComplete update the objects tree (Figure 19). It is recommended to apply the method only to a restricted object hierarchy branch, otherwise the refreshing can take a great deal of time.

```
…
page.formLoginform.submitbtLogin.Click();
page.Wait();
Sys.Refresh();
…
```

Figure 19. Refresh Method

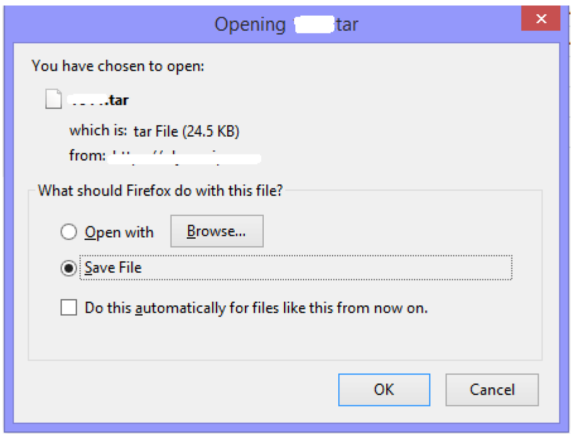
It is also possible to use the WaitAliasChild method, which was mentioned above, or similar WaitChild method instead of FindChild to make TestComplete wait for an object's appearance - these methods refresh the objects tree implicitly. Both workarounds helped and the test script ended successfully.

The test may keep on failing during playback. The reason is that the data provider cannot retrieve all data from Excel spreadsheet. It turns out that data in every column, except for the column name, needs to be of the same type, for example, text only or numbers only. Otherwise, data may be treated incorrectly and hence test fails. (SMARTBEAR 2014p.)

The scenario in which the test passes and then fails may also happen, for example, if the code in the test script enforces TestComplete to look for objects that for some reasons have been deleted from the system. For example, in the sample test case workspace of type Delivery is to be created. The type should exist in the system before the test is run.

An unexpected failure can occur because the Web browser configuration has been changed. As it has been mentioned above, wrong configuration can cause incorrect playback.

Some specific issues must also be considered. Mozilla Firefox was used in this project. As the Mozilla Support website informs, when a link to download a file is clicked, the Internet Media type defines what Firefox will do. There may be some plugin installed that will automatically handle the download, or a dialog asking whether to open the file or save it may appear (Figure 20).


Opening [redacted] tar

You have chosen to open:

[redacted].tar

which is: tar File (24.5 KB)

from: [redacted]

What should Firefox do with this file?

- ( ) Open with Browse...
- (●) Save File
- [ ] Do this automatically for files like this from now on.

Figure 20. Opening file dialog

The checkbox "Do this automatically for files like this from now on" in the dialog may be checked and the dialog will not appear afterwards for that type of file. In Figure 21 the Export button is clicked to download a file.

```
…
frame.linkExportWs.Click();
browser.WaitAliasChild("browserWin", 1000);
…
```

Figure 21. Download File

The browser object then waits for the browserWin object i. e. the dialog window to appear However, as it has become clear, the object may not appear at all because of the individual browser settings. As a result, the download operation and hence the test fail. The problem could be solved programmatically but it is hard to predict all of the possible scenarios.

To avoid these kinds of failures, it is important to have documented instructions on how to prepare the system and the browser before a test round.

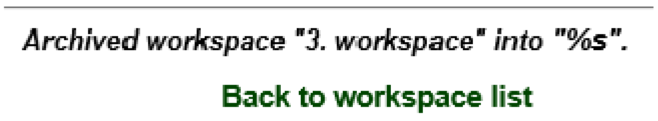
Test scripts should be simple, well designed and maintainable. If maintaining automated tests takes more time than manual testing there is scarcely any point in automation. However, maintenance can be hardly escaped. User interfaces change even in stable applications. In case of manual testing people can handle these changes without any problems. But automated test scripts can fail because of the slightest change. The code above initiates choosing the product basic interface from the select list (Figure 22).

```
frame.slTemplateset.ClickItem("Product name basic user-interface");
```

Figure 22. Click Item with a Product Name

During the project implementation the product name was slightly changed and this caused a test failure. That is why it is also necessary to be informed about such changes in advance. The easiest way is to get the information from the development team. The team may be asked to post corresponding messages to the change log in the application version control system.

A tester should also keep in mind that automated tests are not likely to find unexpected defects. Running tests the same way and expecting to get a certain output makes it impossible to detect when something else goes wrong. A message shown after a workspace has been archived is presented in Figure 23.



Archived workspace "3. workspace" into "%s".

Back to workspace list

Figure 23. Archive workspace

The generated path to the archived workspace is evidently wrong. If the code in the test script will not check it, which may happen, the defect will not be caught by TestComplete. But the issue is not critical as it does not crucially affect the basic functionality of the product and the expected output of the test case “successful archiving” is accomplished. After all, it is not possible to check everything and even a human tester may not notice this kind of issues.

Nevertheless, a tester should be careful in choosing what exactly should be checked in the automated test. This may require the correction of flaws in existing manual tests (something like “successful archiving” in the expected output would not be sufficient), enough experience with the application and a good knowledge of the tool.

5 Discussion

The goal of this thesis was to reveal the strengths and weaknesses of test automation on the example of the smoke test automation project for one specific company. The automation was done using TestComplete, a popular automation tool. The system under test is a web-based document management system for complex industrial projects. The thesis introduced the basic principles of the tool and demonstrated how to use them by describing the implementation process. The problems encountered during the project implementation and possible ways to avoid them were then presented.

This chapter summarizes the main results of the study. The future suggestions for further development are also discussed. The assessment of the author's professional progress is done as well at the end of the chapter.

5.1 Conclusions

The tangible result of the thesis is a project which enables to run tests automatically. The problems encountered during the project implementation have revealed that test automation should be done very thoroughly in order to avoid them.

The process should be carefully planned. The manual tests intended for automation may require adjustment that should be controlled or done by the members of the team experienced in the application. TestComplete project should be well-structured. The code needs to be logically organized to allow easy interaction between modules, further usage in different test cases and thus constant test coverage improvement.

The examples with the Wait() method which does not work for dynamic web pages, or the ExcelDriver object which may treat data of different types incorrectly have revealed that TestComplete, just like any other Software tool, has distinctive features a tester should be aware of. Flaws in the test script code or external data sources may result in unjustified test failures. Thus ignorance can considerably slow down the automation process, make it more difficult. Ideally, script developers should be experts in the automation tool.

One of the key requirements in test automation is creating scripts easy to maintain. The thesis has set out that TestComplete provides name mapping techniques, such as Extended find, Wildcarding etc. which enable object search on dynamic web pages. Data-Driven approach separates test data from the code. Undoubtedly, this has a significant positive effect on maintainability. Still it has become clear that it is not enough to make tests runnable. Successful test automation requires support from management to ensure, inter alia, the timely awareness of the changes performed in the system that can affect maintainability. Having documented instructions on how to prepare the system and the browser before a test round is of great importance.

It has also been pointed out that automated tests are not likely to find unexpected defects.

However, the automation of the smoke test has provided essential benefit to testing. It has given an opportunity to run the tests on new versions of the product without great efforts and hence make sure that the changes made in the code do not affect the basic functionality of the system. This fully corresponds to the purpose of the smoke testing.

Test Automation has given an increased, compared to the manual testing, confidence that the tests are run correctly according to the smoke test instructions.

The testing time has significantly decreased. The time spent on the configuration tests has changed from 2,5 hours to 15 minutes (Figure 16).

Moreover, the data driven approach and the project code structure make the creation of new automated tests easier. For example, it is now possible to create many workspaces and try different inputs just by creating a short script and populating the Excel table presented in Table 5. All this clearly makes testing more efficient.

5.2 Future development

The scope of this thesis is rather narrow. The project was implemented using certain methods, such as creating test scripts from scratch in JScript , Name mapping, Data-Driven approach via Excel spread sheets. It was developed for a certain type of application, namely Web application. However, TestComplete provides a wide range of possibilities and tools how test automation can be done. It is a huge field to be investigated. It would be interesting to continue with TestComplete and look for the ways to improve the project as well as to automate tests for desktop or mobile applications. This would give a deeper understanding of the tool and, of course, affect the final project results.

5.3 Self-assessment

Before the project implementation, I didn’t have neither theoretical nor practical knowledge about test automation and test automation tools. The need to automate a smoke test with TestComplete for a specific company pushed me to research them and apply new knowledge in practice. It gave me invaluable experience which helped to understand the core concepts of test automation and realize its complexity and importance for software development. I acquired much knowledge about TestComplete, one of the most popular automation tools.

The project implementation was labour-intensive and absorbing. Several full project test rounds and numerous test items runs were executed which resulted in finding defects in the system under test as well as in the project itself. The project defects were then fixed. They significantly helped in writing this paper.

References

Alpaev G. 2013. TestComplete Cookbook: Over 110 practical recipes teaching you to master TestComplete – one of the most popular tools for testing automation. Packt Publishing. Birmingham.

Al-Zain, S., Eleyan, D., Garfield. J. 2012. Automated User Interface Testing for Web Applications and TestComplete. In: CUBE 2012, Proceedings of the CUBE International Information Technology Conference, 3 – 5 September 2012, Pune, India. URL: http://eprints.worc.ac.uk/3190. Accessed: 20 February 2015.

Apica 2014. Automated Testing vs Manual Testing: Which Should You Use, and When? URL: https://www.apicasystem.com/blog/2014/11/07/automated-testing-vs-manual-testing. Accessed: 20 February 2015.

Chapman, S. 2013. A Brief History of Javascript. URL: http://javascript.about.com/od/reference/a/history.htm. Accessed: 5 May 2013.

Düüna, K. 2012. Analysis of Node.js platform web application security: Master's thesis. Tallinn University of Technology. Tallinn. URL: http://www.google.ru/url?sa=t&rct=j&q=analysis%20of%20node.js%20platform%20web%20application%20security%3A%20&source=web&cd=1&ved=0CC4QFjAA&url=http%3A%2F%2Flab.cs.ttu.ee%2Fdl93&ei=o8VeUdDxIYmh4gS-joGADQ&usg=AFQjCNE0MMCy8ZYdpi5O0gzr-2Qy5e2phg&bvm=bv.44770516,d.bGE&cad=rjt. Accessed: 5 May 2013.

Hayes, L.G. 2004. The Automated Testing Handbook. 2nd ed. Software Testing Institute. URL: http://books.google.fi/books?id=-jangThcGIkC&printsec=frontcover#v=onepage&q&f=false. Accessed: 20 February 2015.

International Software Testing Qualifications Board (ISTQB) 2014. Glossary: Standard Glossary of Terms used in Software Testing. v.2.4. URL: http://www.software-tester.ch/PDF-Files/ISTQB%20Glossary%20of%20Testing%20Terms%202.4.pdf. Accessed: 25 February 2015.

Klärck P. Introduction to Test Automation. URL: http://www.slideshare.net/pekkaklarck/introduction-to-test-automation?related=3. Accessed: 11 March 2015.

Laukkanen, P. 2006. Data-Driven and Keyword-Driven Test Automation Frameworks: Master’s thesis. Helsinki University of Technology. Espoo. URL: http://eliga.fi/Thesis-Pekka-Laukkanen.pdf. Accessed: 20 February 2015.

Lundgren, A. 2008. Abstraction Levels of Automated Test Scripts: Master’s thesis. Lund University. Stockholm. URL: http://fileadmin.cs.lth.se/cs/Education/Examensarbete/Rapporter/2008/Rapport_2008-11.pdf. Accessed: 20 February 2015.

Marick, B. 1998. When Should a Test Be Automated? URL: http://www.stickyminds.com/article/when-should-test-be-automated. Accessed: 20 February 2015.

Mozilla Support 1998–2015. Change what Firefox does when you click on or download a file. URL: https://support.mozilla.org/en-US/kb/change-firefox-behavior-when-open-file Accessed: 20 February 2015.

Prasad, K.V.K.K. 2009. Software Testing Tools: Covering WinRunner, Silk Test, LoadRunner, Jmeter, TestDirector and QTP with case studies. Dreamtech Press. New Delhi. URL: https://books.google.fi/books?id=DuinhInx0moC&pg=PA2&dq=Prasad+K.V.K.K.+2009.+Software+Testing+Tools:&hl=en&sa=X&ei=pUnnVNnCF-HkyAPQxI-DACg&ved=0CDMQ6AEwAA#v=onepage&q=Prasad%20K.V.K.K.%202009.%20Software%20Testing%20Tools%3A&f=false. Accessed: 20 February 2015.

SMARTBEAR 2014a. Distributed Testing - Overview. URL: http://support.smartbear.com/viewarticle/55788. Accessed: 25 February 2015.

SMARTBEAR 2014b. About TestComplete Platform and Modules. URL: http://support.smartbear.com/viewarticle/55808. Accessed: 21 February 2015.

SMARTBEAR 2014c. About TestComplete. URL: http://support.smartbear.com/viewarticle/ 55657. Accessed: 21 February 2015.

SMARTBEAR 2014d. Test Types. URL: http://support.smartbear.com/viewarticle/ 58033. Accessed: 21 February 2015.

SMARTBEAR 2014e. About Name Mapping. URL: http://support.smartbear.com/viewarticle/55236. Accessed: 24 February 2015.

SMARTBEAR 2014f. Changing Mapped Objects' Hierarchy. URL: http://support.smartbear.com/viewarticle/57334. Accessed: 24 February 2015.

SMARTBEAR 2014g. About Object Spy. URL: http://support.smartbear.com/viewarticle/55432. Accessed: 21 February 2015.

SMARTBEAR 2014h. About Test Log. URL: http://support.smartbear.com/viewarticle/55125. Accessed: 21 February 2015.

SMARTBEAR 2014i. WaitBrowser Method. URL: http://support.smartbear.com/viewarticle/61205. Accessed: 21 February 2015.

SMARTBEAR 2014j. FindChild Method. URL: http://support.smartbear.com/viewarticle/55112. Accessed: 21 February 2015.

SMARTBEAR 2014k. FindAliasChild Method. URL: http://support.smartbear.com/viewarticle/55413. Accessed: 21 February 2015.

SMARTBEAR 2014l. aqObject Object. URL: http://support.smartbear.com/viewarticle/55360. Accessed: 21 February 2015.

SMARTBEAR 2014m. ExcelDriver Method. URL: http://support.smartbear.com/viewarticle/55314. Accessed: 21 February 2015.

SMARTBEAR 2014n. Running Tests. URL: http://support.smartbear.com/viewarticle/56462. Accessed: 21 February 2015.

SMARTBEAR 2014o. Waiting For Web Pages. URL: http://support.smartbear.com/viewarticle/57082. Accessed: 21 February 2015.

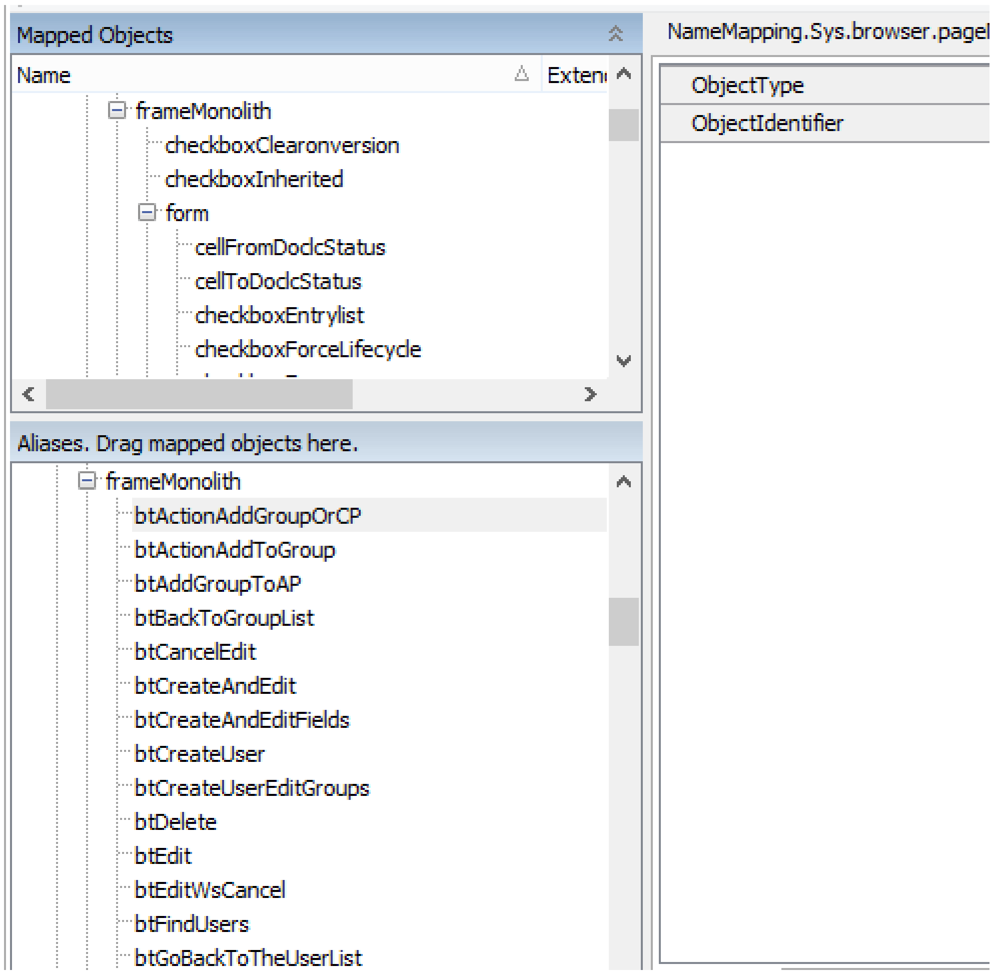
SMARTBEAR 2014p. Using Excel Files as Data Storages. URL: http://support.smartbear.com/viewarticle/56928. Accessed: 21 February 2015.

Spillner, A., Linz, T. Schaefer, H. 2014. Software Testing Foundation: A study guide for the Certified Tester Exam. 4th ed. Rocky Nook Inc. Santa-Barbara.

Ugurlu, T., Zeitler, A., Kheyrollahi, A. 2013. Pro ASP.NET Web API HTTP Web Services in ASP.NET. Apress.

Appendices

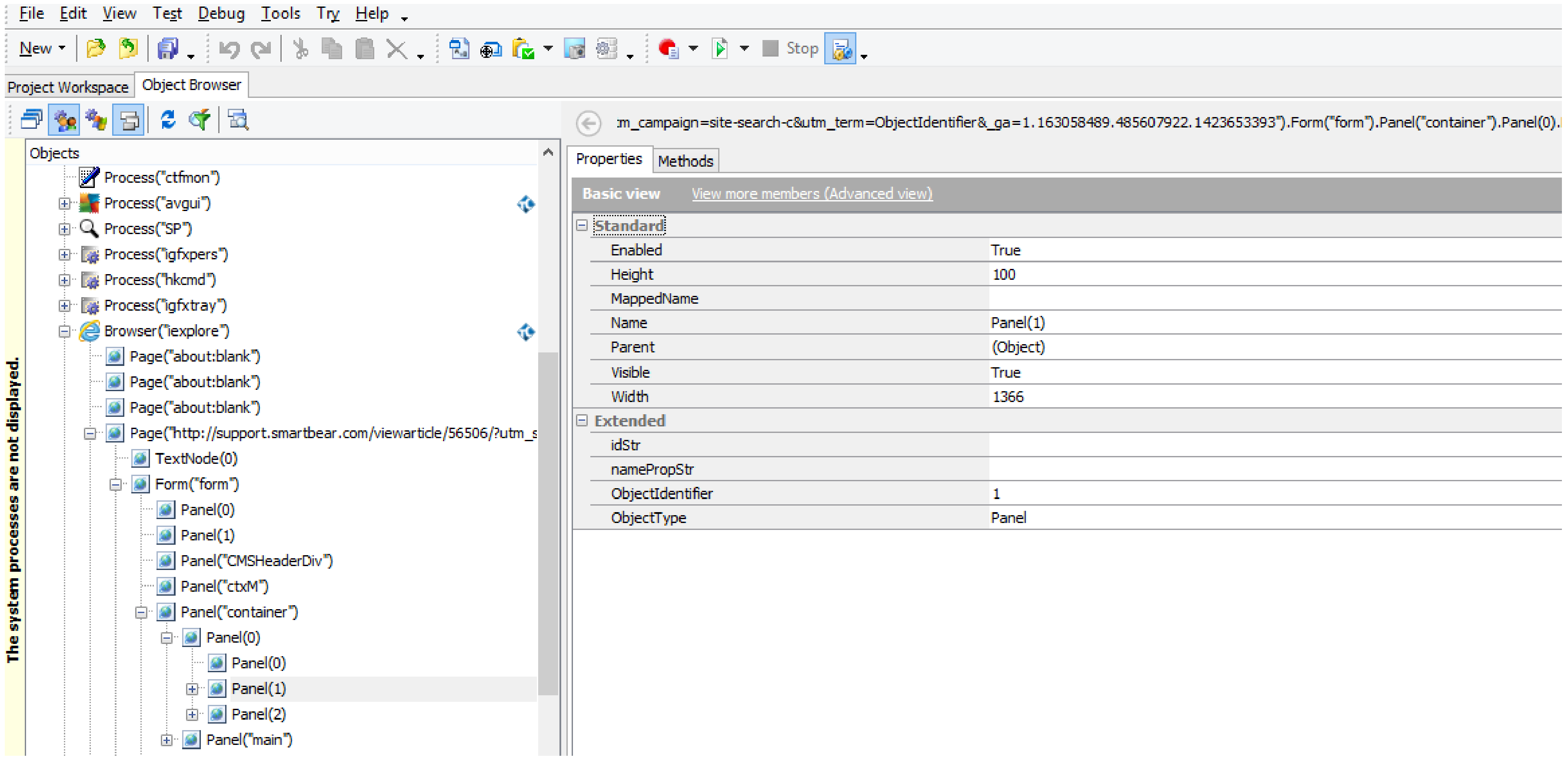
Appendix 1. Mapped Objects and Aliases trees


Mapped Objects
Name
Exten
NameMapping.Sys.browser.pagel
ObjectType
ObjectIdentifier
Aliases. Drag mapped objects here.

```
frameMonolith
    checkboxClearonversion
    checkboxInherited
    form
        cellFromDoclcStatus
        cellToDoclcStatus
        checkboxEntrylist
        checkboxForceLifecycle
```

```
frameMonolith
    btActionAddGroupOrCP
    btActionAddToGroup
    btAddGroupToAP
    btBackToGroupList
    btCancelEdit
    btCreateAndEdit
    btCreateAndEditFields
    btCreateUser
    btCreateUserEditGroups
    btDelete
    btEdit
    btEditWsCancel
    btFindUsers
    btGoBackToTheUserList
```

Appendix 2. Object Browser



The system processes are not displayed.

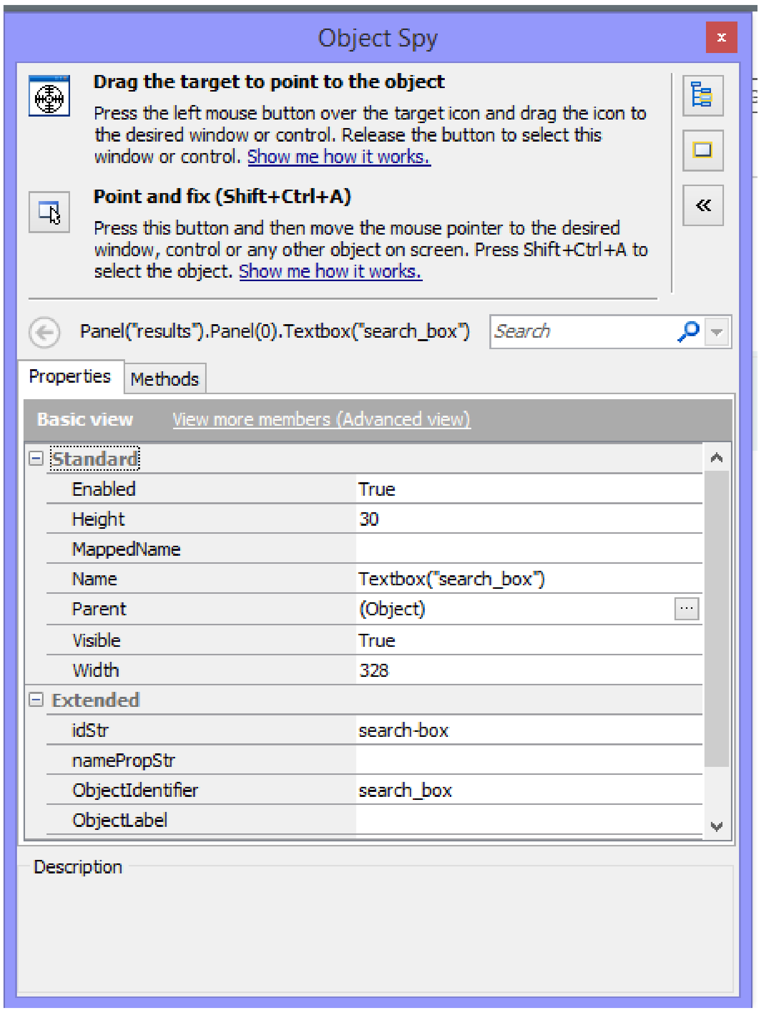
```
Process("ctfmon")
Process("avgui")
Process("SP")
Process("igfxpers")
Process("hkcmd")
Process("igfxtray")
Browser("iexplore")
  Page("about:blank")
  Page("about:blank")
  Page("about:blank")
  Page("http://support.smartbear.com/viewarticle/56506/?utm_s
    TextNode(0)
    Form("form")
      Panel(0)
      Panel(1)
      Panel("CMSHeaderDiv")
      Panel("ctxM")
      Panel("container")
        Panel(0)
          Panel(0)
          Panel(1)
          Panel(2)
        Panel("main")
```

m_campaign=site-search-c&utm_term=ObjectIdentifier&_ga=1.163058489.485607922.1423653393").Form("form").Panel("container").Panel(0).

Basic view   View more members (Advanced view)

Standard	
Enabled	True
Height	100
MappedName	
Name	Panel(1)
Parent	(Object)
Visible	True
Width	1366
Extended	
idStr	
namePropStr	
ObjectIdentifier	1
ObjectType	Panel

Appendix 3. Object Spy



Object Spy

Drag the target to point to the object

Press the left mouse button over the target icon and drag the icon to the desired window or control. Release the button to select this window or control. Show me how it works.

Point and fix (Shift+Ctrl+A)

Press this button and then move the mouse pointer to the desired window, control or any other object on screen. Press Shift+Ctrl+A to select the object. Show me how it works.

Panel("results").Panel(0).Textbox("search_box")

Standard	
Enabled	True
Height	30
MappedName	
Name	Textbox("search_box")
Parent	(Object)
Visible	True
Width	328
Extended	
idStr	search-box
namePropStr	
ObjectIdentifier	search_box
ObjectLabel	