Open2Test Test Automation Framework Introduction – TestPartner

Version 1.0

September 2009

Table of Contents

1. Preface

Automation testing is an emerging field that draws maximum benefits with minimum effort. The benefit of automation testing is its ability to increase the efficiency of resources, increase test coverage, and increase the quality and reliability of the software.

While there are several frameworks that provide support for automated software testing, this document introduces one particularly effective type: the **Open Source Test Automation Framework**.

In the Open Source Test Automation Framework, the discrete functional business events that make up an application are described using keywords. This approach fosters code reusability, optimum utilization of the tool, and greater productivity.

2. Framework Overview

2.1. Introduction to Framework

Automation testing requires a well-defined approach, based on a comprehensive framework, in order to get maximum benefits.

A **framework** is a hierarchical directory that encapsulates shared resources, such as a dynamic shared library, image files, localized strings, header files, and reference documentation in a single package.

There are various frameworks available for automation, such as:

- Test Script Modularity Framework
- Test Library Architecture Framework
- Data-Driven Automation Framework
- Hybrid Automation Framework
- Keyword-Driven Automation Framework

Keyword-based test design and test automation is based on the idea that the discrete functional business events that make up any application can be described using short text descriptions (keywords). By designing keywords that describe those discrete functional business events, testers begin to build a common library of keywords that can be used to create test scripts.

The Open Source Test Automation Framework is a keyword-driven automation framework that works with MicroFocus TestPartner (TP). This framework allows testers to develop test cases using Microsoft Excel and a list of keywords. When the test is executed, the framework processes the Excel workbook and calls functions associated with the keywords entered in the Excel spreadsheet. These keyword functions in turn perform specific actions against the application under test (AUT). The framework interprets the keyword and executes a set of statements in the program.

With this framework in place, applications can be automated without starting from scratch. Testers simply use the application-independent keywords and create extra application-specific keywords.

2.2. Framework Features

In addition to standard features such as performing operations and verifications on the objects, the framework includes other sophisticated features, such as:

1. **Use of variables:** Variables can be defined and used across the generated test script. This can be used to capture runtime values, which can be reused as input elsewhere during test execution.
2. **Conditional checking:** Conditional constructs such as ‘if’ can be implemented using keywords to handle different flows based on various conditions.
3. **Data-driven testing:** This framework supports data-driven testing by importing data from an external data sheet.

4. **Reports:** Customized report messages can also be used to perform effective analysis on execution reports. These reports can be customized to display the pass or fail condition of any function, even during the verification of any checkpoints.
5. **Calling functions and reusable actions:** Common functions or actions can be triggered through keywords. This framework supports a functional decomposition approach. This increases the reusability of functions, which in turn reduces the unnecessary repetition of steps.
6. **Keyboard inputs:** This plug-in is included to perform keyboard actions on the specified test object.
7. **Date/time functions:** The date/time function plug-in is included to perform different operations on date/time.
8. **Exception handling:** Runtime errors can be effectively handled and reported using this framework.

2.3. Framework Benefits

Reusability:

The Open Source Test Automation Framework is an application-independent framework that deals with all possible actions and verifications that can be performed on an object. Therefore, the code for the same object can be used across different applications.

Duplication of work is minimized at every level. For instance, a user might have to perform a certain action on an object of a similar class (e.g., clicking a button) repeatedly. This can be in the same test case or in a different application altogether. In both cases, the same code can be reused.

Optimum utilization of the tool:

The framework has the advantage of using keywords as the input for triggering an action. This well-built framework uses the features of the tool effectively. For instance, TestPartner uses a shared object repository where all the objects required can be added and reused across the scripts for an application under test (AUT).

Less effort:

The effort involved in coding and reviewing is minimal when compared to other frameworks, since a good percentage of coding is done within the framework. The tester simply has to enter the keywords, reducing the time required for coding. Recording is also not required, as the global repository is used. The amount of rework required for migrating from one application to another on the same platform is reduced because the code remains the same.

Increased quality:

The scripts will be of uniform quality since they make use of the same code.

Greater productivity:

The Open Source Test Automation Framework provides both qualitative and quantitative benefits for automation and is highly productive compared to any other framework. This framework also addresses the ongoing maintenance of the test scripts in a cost-effective manner.

Maintenance:

Simple modifications to the application can be easily handled in the code. The changes will be done only in the external file containing the code and the scripts need not be changed. Hence, it is easy to maintain the scripts and provide a cost-effective solution for test automation.

No scripting skills required by the end user:

No coding skills are required to automate and review the scripts. The scripts are user-friendly. Scripts can be interpreted easily by a person who does not have complete knowledge of the tool.

Return on investment is high:

Although the initial effort for building the framework is high, in the long run the return on investment will be high because of the reusability and optimum utilization of the tool.

3. Framework Architecture

3.1. Framework Architecture

Architecture forms the foundation of any software application. It should be robust enough to handle the desired functions efficiently and effectively. In this approach, the goal is to develop an application-independent reusable keyword framework that can be used directly across any application without spending any extra time on it.

In order to make all the components of the system work in sync, it is important to define the components and functionalities, as well as the binding relationship between them.

An Automation Framework Architecture comprises the following components:

- **Framework**

The framework consists of the following sub-components, namely:

  - Function Library
  - Common Functions

- **Abstract Layer**

The abstract layer consists of the following sub-components, namely:

  - The Object Repository
  - Driver Script
  - Keywords

- **External Data**

External data consists of the following sub-components, namely:

  - Data Sheets
  - Global Variables

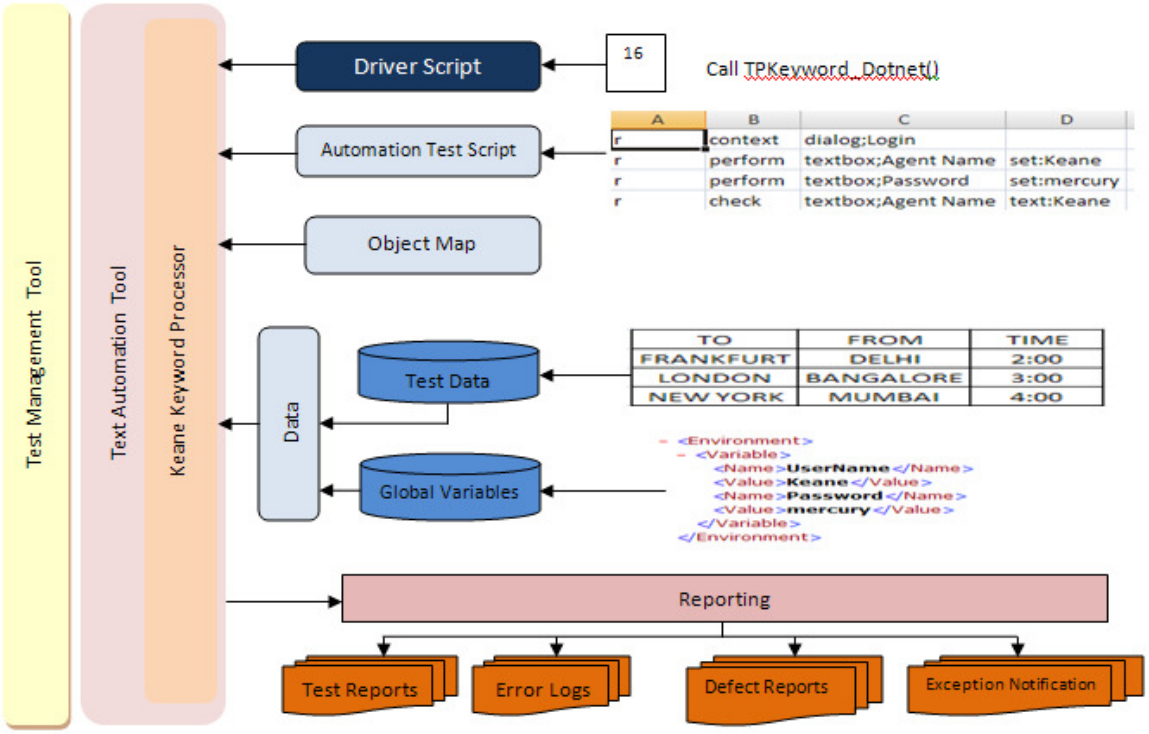


Figure 1: Framework Architecture

3.1.1. Driver Script

The driver script (DS) drives the script execution. It is the few lines of script in the TestPartner main window that will invoke the process of synchronizing the keywords with the framework and the object repository. This script calls the functions from the function library, which reads the keywords, objects, and parameters and performs appropriate actions as per the functions in the function library. By incorporating the DS with the framework in a separate function, the effort required to develop the DS is reduced. A call to the function will act as the DS.

3.1.2. Shared Module

The shared module forms the backbone of the automation framework. All the coding logic is in the form of a user-defined VBA. All these functions are stored in the shared module. This is the place where most of the scripts reside and where customization can be performed. The shared module is the only component in the framework that has to be changed in case the application is migrating from one platform to the other. This addition and deletion of functions makes the framework flexible enough to use it for any other application.

3.1.3. Common Function

The common functions (CF) are the functions that are reusable across all platforms. These functions are application-independent, and they do not depend on the technology that has been used to develop the application. Separating the common functions from the shared module ensures maximum utilization of reusable scripts, and in turn reduces the maintenance effort of scripts. Some of the common functions include conditional functions, loop functions, etc.

Syntax for function

fun <fun name>(arg1,arg2)

<Statements>

end

For example if u want to call mul(3,6)

Call mul(3,6)

function mul(a,b)

mul=a*b

end

3.1.4. Object Map

TestPartner learns the interface of an application by learning the application's objects, their corresponding property values, and the object descriptions. **The object map** is the place where TestPartner recognizes and stores information such as properties, values, etc.

With this object map, the tool identifies the application and executes the business flows as per the functions in the test script. If any of the object's property values in the application differ from the property values stored in the object repository, the tool will not identify the object and the script will fail. Therefore, it is necessary to change the property value in the object map if it differs from the property value in the application.

3.1.5. Keywords

Keywords trigger specific functions in the framework to perform a specific operation on the desired object in the application. These keywords are fed in the data table of TestPartner or as an input Excel sheet, which will be fed in by the driver script. The data table records contain the keywords that describe the desired actions. They also provide additional data needed as input into the application, the benchmark information to verify the state of components, and the application in general.

Ideally, the keyword (vocabulary) should be written in such a way that it is recognizable and suitable for manual testing. Achieving this crossover capability allows us to create one test case for both automated and manual test cases. It is this feature that makes the keyword-driven framework powerful.

3.1.6. External Test Data

External test data is given as inputs to the test scripts to perform the same operations on the application using different set of data. This spreadsheet holds multiple combinations of inputs to be fed to test the application. External test data can also be given as an input sheet during checking operations. The best practice here is to keep the data sheet in a common place, preferably in the test management tool.

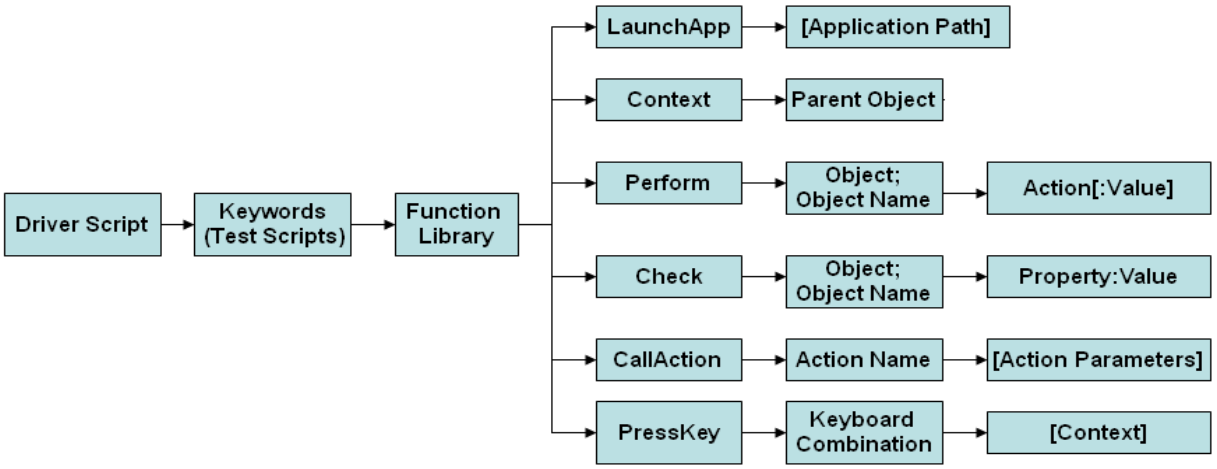
3.1.7. Global Variables

In TestPartner we make use of a concept called Dictionary Objects to store variables across the entire script. The Dictionary object is a data structure that can contain sets of pairs, where each pair consists of an item, which can be any data type, and a key, which is a unique string value that identifies the item.

3.1.8. Reporting

After execution of the test script, it is necessary to obtain the results of the execution. Apart from the test execution report generated by the TestPartner tool, testers can customize reports in an external spreadsheet format. This provides report details for which test scripts have failed or passed while running a test suite. The reports detail exception cases handled, defects found, and errors logged. These functions are customized in the driver script. This helps in performing effective analysis on the execution report.

4. Flow Diagram of the Open Source Test Automation Framework



Note :

The values within [ ] are optional, usage depends on the scenario

Figure 2: Flow Diagram

At the start of the test execution, the driver script is invoked. The function library contains all the code for the actions identified. The driver script traverses through keywords, and each keyword triggers the function library to perform the desired action on the application.

5. Conclusion

Analysis is an important and time-consuming phase of automation testing. However, in the long run, the time spent will be useful during the regression phase. In order to keep up with the pace of product development and delivery, it is essential to implement effective, reusable test automation. The Open Source Test Automation Framework provides a way to drive productivity and foster code reuse — ultimately enhancing the quality of resulting software.