Chapter 10. Frameworks

In this chapter, we will cover:

- Introduction to test automation frameworks
- Designing a test automation framework
- Building a test controller
- Building a reusable component (action)
- Building an event handler
- Building a test reporter

Introduction to test automation frameworks

This section intends to cover:

- Definition of a test automation framework
- Advantages of using a test automation framework
- Types of test automation frameworks (modular-driven, data-driven, keyword-driven, model-driven, and hybrid)
- Designing a test automation framework

Definition of a test automation framework

Test automation framework (often referred to in the testing industry as testware automation frameworks) is an integrated software solution that defines the rules for the development, maintenance, and execution of **automated test assets**. Typically, such a solution also defines how test results are reported, how runtime errors are handled, and how test data is managed. A test automation framework comprises function libraries, data sources (for example, Excel or XML files and DB), object models (such as stored in an object repository), and it may also include additional reusable external modules (for example, DLL files, COM objects, configuration scripts, and so on).

Advantages of using a test automation framework

A well-designed test automation framework typically contributes towards lower reduce development and maintenance costs of automated test assets. This means that, ideally, a framework should use generic and agnostic design patterns to provide a solution that is relevant, effective, maintainable, efficient, manageable, portable, reliable, and diagnosable.

The preceding list is not meant to be exhaustive, but provides the main features that are expected from a robust automation framework.

Another aspect of test automation that is quite often neglected, but one that can make a great contribution to design patterns, is the **command wrapper** (used when implementing test procedures and functions). This will also be explained later in this section.

Types of test automation frameworks

There are several generic types of test automation frameworks' design patterns:

- Modular-driven framework
- Data-driven framework

- Keyword-driven framework
- Model-driven framework (for example, **Action Based Testing** (**ABT**) or as termed in UFT, **Business Process Testing** (**BPT**))
- Hybrid approach framework

Note

Hybrid approach is a term used by the authors that refers to test automation frameworks that are a combination of the other design patterns, usually such that, they implement a blend of all, or part of, the design feature of the other patterns.

Selecting a framework type

As far as which framework type to select is concerned, it depends on the requirements we have to meet and the level of automation maturity required. No single pattern type is a silver bullet, and a more complex framework should not be used if the additional complexity is not expected to yield significant added value. We shall now define and explain the different patterns mentioned earlier.

Modular-driven framework

A modular-driven pattern uses classes and objects to encapsulate all the entities involved in the automation project. Basically, a modular-driven pattern can be considered a particular case of the hybrid approach. This is because the utilization of classes and objects is typically also accompanied by the usage of the features of the keyword-driven pattern (in more complex implementations) and the data-driven pattern.

Using classes (as described in detail in the previous chapters) allows for a better organized code base, typically resulting in a more concise, clearer, as well as more flexible, extensible, and, maintainable code. This also enables us to use object-oriented design patterns to maximize reusability and enhance the performance of our test suites. For example, a typical modular-driven pattern has a well-defined structure (or template) to implement runnable processes using the command wrapper pattern (refer to Chapter 7, *Using Classes*). This, in turn, simplifies the way data is loaded, saved, and shared; the way events are reported to the log file; how the flow is controlled; and finally, the way exceptions are handled.

Data-driven frameworks

A data-driven pattern supports test iterations and flow branching according to external input data. UFT can, by default, offer such a framework out-of-the-box; all we have to do is define the parameters in the DataTable, retrieve the values of these in our code where appropriate, and set the test or Action iterations (datasheet rows range) we wish to execute (refer to Chapter 1, *Data-driven Tests*).

Keyword-driven frameworks

A keyword-driven pattern is also a data-driven pattern, but with another level of abstraction; commands are encapsulated as data entities (keywords) and mapped to actual functions implemented in code.

These keywords are listed in their planned order of execution in some kind of data source (for instance, Excel worksheet, XML file, or DB). Typically, this data source would also include the corresponding parameter values for each operation. A central mechanism, often referred to as a controller or parser, reads the sequence of keywords and invokes the procedures associated with them.

Central to a keyword-driven pattern is the desire to provide nontechnical test engineers, who do not possess coding skills, with the ability to implement **automated test scenarios** (also referred to as test scripts) using a high-level structured language composed with words that represent underlying coded processes of varying complexity.

This pattern is very popular, and its main advantages are:

- Automation test scenarios are represented by steps composed of basic building blocks (procedures) and parameters.
- Apart from more complex cases, which are difficult to capture with a simple step-by-step representation, coding is essentially not required to automate the tests.
- Different levels of granularity are supported. For example, a keyword may represent a single test object method (for instance, `PressOK`) or a procedure (for example, `Login`). The latter is basically ABT or BPT, as the functionality offered by UFT requires connectivity with HP ALM/QC.

A keyword-driven pattern also has some challenges that require our attention. For instance, it typically carries the additional cost of developing a **User Interface** (**UI**) to manage, build, and validate the data entered by the designer of the tests while defining automated test scenarios.

Hybrid frameworks

A hybrid pattern is one that combines some or all the features of the mentioned framework types. It is typically implemented with a strong emphasis on design patterns, to provide a solution that yields high scores in (source Hybrid (keyword/data-driven) frameworks, ANZTB Conference, 2010 by Jonathon Wright):

- Maintainability: significantly reduces the test maintenance effort
- Reusability: due to modularity of test cases and library functions
- Manageability: effective test design, execution, and traceability
- Accessibility: to design, develop & modify tests whilst executing
- Availability: scheduled execution can run unattended on a 24/7 basis
- Reliability: to advanced error handling and scenario recovery
- Flexibility: framework independent of system or environment under test
- Measurability: customizable reporting of test results ensure quality