GUI Test Automation

How-To Tips

www. routinebot.com
AKS-Labs

GUI Test Automation - How-To Tips

It's often said that "*First Impression is the last impression*" and software applications are no exception to that rule. There is little doubt that the user interface (aka GUI) is the heart and soul of an application and is one of the major factors which influences user's impression.

Therefore, it's absolutely crucial that the GUI for an application be thoroughly tested. While GUI test automation seems to be the most obvious way forward that comes to mind, it's easier said than done. There are a numerous challenges in GUI Test Automation – multiple browsers, specific browser versions, cross-site references, proprietary JavaScript frameworks and several others.

This "How-To" series is aimed at providing test engineers and QA professionals with valuable tips and tricks for GUI Test Automation.

How do I test the impact of Back-End Configuration Changes on my application GUI?

Many web applications have back-end configuration parameters which can significantly alter that look and feel as well as the overall navigation flow. The best way to automate the testing of such scenarios is to use the record and replay approach while passing the back-end values as configurable parameters, either directly through the database or by storing the values temporarily in a data file.

This approach is especially helpful as it allows validating the application UI for all possible data sets of back-end configuration parameters. GUI Automation testing tools such as RoutineBot provide advanced recording capabilities which facilitate testing the impact of such back-end configuration changes.

How do I ensure that my GUI test automation scripts are automatically run when the application is re-built?

Conducting regression tests is one of the major objectives of performing GUI test automation. It's crucial to ensure that new code changes do not break the existing functionality. The best way to ensure this is to tie the execution of your automated GUI scripts to the build process. Whenever a developer checks-in any code changes, it triggers a build and deployment, followed by automatic execution of automated GUI regression scripts.

This helps ensure that no GUI bugs have been introduced as a result of the latest changes checked-in by the development team. A number of leading build tools such as Maven and Cruise Control support dynamic triggers for re-compiling and re-deploying your application followed with automatic execution of GUI test cases. For example, it's a good practice to attach a tool such as RoutineBot to your automated build and regression test cycle to ensure that there are no UI bugs as a result of the latest code changes.

How do I test applications which access multiple servers/ sites?

It's common to have web applications which access multiple servers or sites for specific tasks. E.g. many applications delegate the authentication process to third-party applications which may be running on a different server/ website. It is impossible to test such scenarios with basic GUI automation testing tools such as JMeter and Selenium due to the inherent limitation of cross-site scripting (XSS) security checks.

The trick to testing such applications is to run your test client as HTA Applications in IE. Since HTA apps run within a trusted security context, they bypass XSS limitations. For Safari, run your test client directly using a file:// URL so that it bypasses the built-in browser security policy. For Firefox, use Selenium-RC (Remote Control), an extension which smartly bypasses XSS limitation.

How do I conduct load and performance test for my application GUI?

While there are a plethora of tools available to conduct GUI automated unit tests for applications, a significant proportion of these do not provide any support for conducting load and performance tests. It's possible that your application GUI passes automated unit tests, but may crash or lead to unexpected results under a heavy-load scenario in live deployment. Therefore, it's crucial to test your application GUI for stress and performance tests before its deployed live.

If you know a bit of programming, writing load and performance tests is pretty easy. To generate load, create multiple threads, each one navigating through the GUI flow of your application. To generate variable load, it's recommended to introduce random delays on multiple threads in order to simulate real-time traffic.

Apache JMeter is a highly effective tool for conducting automated load tests for any application's GUI. It's open-source, written in Java and provides excellent support for testing static as well as dynamic resources.

How do I automate testing of scenarios which involve AJAX?

While AJAX is a developer's dream come true and improves the overall user experience of an application, it's nothing short of a nightmare from a test automation perspective. Since it involves synchronization issues and represents a challenge for deterministic testing, it must be handled carefully.

If you know how to parse XML and JOSN, half the battle to automate the testing of AJAX applications is already won. As a next step, implement a fake server which would simulate the responses coming from your database. The calls are then re-directed to your fake server but your application would behave the same as if it were interacting with a real server.

While leading testing tools such as JMeter and Selenium offer basic AJAX support, it is strongly recommended that you test applications with other tools such as RoutineBot and HTMLUnit which offer advanced AJAX capabilities.

How do I conduct GUI automation tests for my application on multiple browsers?

Browser incompatibilities are the root cause of nearly 80% GUI bugs. Therefore, it's important to do automated GUI testing across multiple browsers. While one approach is to do such tests one browser at a time, it's a time and effort consuming process. A better solution is to use any of the open-source/ commercial browser emulators which have the capability of hosting your application on multiple browsers (and multiple versions).

The USP of such tools is that they can be combined with any GUI automated testing tool such as RoutineBot and provide comprehensive GUI test coverage for an application across multiple browsers and their specific versions.

How do I test heterogeneous applications written in multiple languages like JavaScript & Pascal?

Automating the GUI tests for a heterogeneous application is a tedious task. Automated Tools such as RoutineBot which support multiple scripting languages provide the ideal platform for testing such applications. RoutineBot, for instance, supports Pascal, JavaScript and Basic scripting languages. Further, such tools offer a massive advantage of being technology agnostic and work just as well with a variety of UI technologies such as Windows Forms, Microsoft Silverlight, Flex etc.

How do I invoke other Windows applications as part of my automated GUI tests?

Several web applications use other Windows applications (e.g. opening a document in Microsoft Word or generating a calendar invite for an Outlook event). Therefore, to test such scenarios, it is required that the GUI automation test scripts provide the capability to invoke these Windows applications.

While it's not impossible to develop such code to invoke external Windows applications, it's not advisable to re-invent the wheel. Instead, it's preferable to adopt automated GUI testing tools which provide this capability. For example, RoutineBot provides a ShellExecute function which provides the capability to automatically invoke a program associated with any file extension.

How do I take snapshots while the automated GUI tests are running?

It's a common practice to take screen snapshots while testing an application. When a bug is reported, snapshots help bring the developer and testers on the same page and bridge the communication gap. So, how do you take snapshots of your application while it's running the test cases in your GUI automation test suite?

One way is to take screenshots manually by pressing Alt+PrtSrc every time when you need a snapshot. However, this approach has several disadvantages. First, it's a tedious activity to capture the screen and save it to an image file every time. Secondly, this approach becomes infeasible for load tests. And lastly, taking manual snapshots defeats the sole purpose of adopting *automated* UI testing.

Thankfully, several tools offer the built-in capability to generate snapshots while executing automated UI tests. For example, RoutineBot provides a flexible *TakeSnapShot* command which lets you take a screenshot of the whole screen or the active window while running your automated GUI tests.

How do I induce random delays in my automated GUI tests?

Inducing random delays in firing HTTP requests and fetching response is a common way for simulating real-time traffic in test scenarios. Such delays can be coded into your test cases by calling the sleep() and wait() functions. However, manually inducing such delays can often lead to synchronization issues. Therefore, using automated testing tools which offer such built-in sleep/ wait capabilities is much better than coding such delays.

Moreover, the biggest advantage of wait periods induced through tools (such as RoutineBot Wait and sleep commands) is that it can be induced for any specific event such as mouse over, image loading etc. without requiring mammoth code changes.

My application uses a lot of sophisticated images and heavy-duty graphics. How do I write test cases which can accurate match the image patterns in my test cases to the objects which appear on the screen?

It's a nightmare to automate the GUI testing for high-end graphic applications. Most testing tools use an approximate match approach to detect image patterns; therefore they can lead to skewed test results when subject to high-resolution images as input. Coding this functionality within your test case is possible but requires specialized knowledge of graphic libraries and involves substantial development cost.

The ideal solution for such scenarios is to use a tool which provides support for high-end images and sophisticated graphics. The OpenCV (Open Source Computer Vision) library is widely gaining traction as the leading repository of algorithms for computer applications. Tools like RoutineBot leverage the high-end graphic capabilities of OpenCV in order to conduct pixel-by-pixel matches for accurate comparison results.

How do I invoke test GUI automation scripts and access graphics from multiple external projects for my test scripts?

There are two ways to use the same GUI automation scripts and graphics across multiple projects. First, copy the automation scripts and graphics into each individual project. Second, keep the scripts and graphics as part of one project and then link these as dependencies to any other project which needs them.

The first approach has several limitations – it creates multiple redundant copies of the scripts and graphics, and makes it difficult to keep the scripts synced across multiple projects. Therefore, it's much better to reuse the GUI automation scripts by linking them as dependencies across multiple projects.

This keeps the code clean and ensures that any changes to common scripts and graphics are immediately available across all projects.

How do I schedule the automated GUI tests to run or repeat at a specific time?

Most web applications have varying traffic patterns (peaks, average, and low) at different times during the day. While conducting automated GUI tests for your application, it's often desirable to conduct dry-runs which replicate the traffic patterns in real world. This approach is especially useful for testers when they are trying to replicate a production issue.

The best approach to mimic the traffic pattern for production environment is to use a scheduler service (such as Windows scheduler) to fire your GUI test automation scripts at pre-defined time intervals. Further, using such scheduler services also allows testing the application repeatedly at specific time intervals during the day.

How do I keep track of unexpected UI bugs during GUI Automation Load Tests?

While conducting GUI Automation Load Tests is a must in order to identify any unexpected UI bugs and crashes under varying loads, keeping track of such issues is far from easy. The inherent nature of load tests (high traffic and quick execution speed) makes it a daunting task to capture GUI bugs. Therefore, it's recommended to log all UI failures (Error Windows, Warning Pop-ups, Crash Dialogs, Exceptions stack traces, Windows fatal error dialogues) to an external log file. Further, it's a good practice to log the loading time to identify any potential performance or UI resource crunch issues.

The external log file must be used for reconciliation while replaying the recording of your application's automated GUI test cases.