**Recognizing dynamic content**

It can be complicated to distinguish content with progressively produced qualities utilizing Selenium. Dynamic content depends on dynamic identifiers.

These IDs are created again every time a component is shown, which makes it hard to address the web component in view of this trait. For instance, the IDs in web applications in view of the “Yahoo User Interface Library” look something like this “yui\_3\_8\_1\_1\_13679224741219\_543″.

This ID changes each time the website page reloads. For this challenges faced in selenium automation, you can’t utilize the ID locator to distinguish the component.

**How to settle it:**  
Ranorex accompanies an arrangement of [RanoreXPath](https://www.ranorex.com/help/latest/ranorexpath) weight protocols that naturally choose which trait to use for recognizing the objects relying on specific web libraries.

If you need to determine a particular issue while distinguishing web components, you can add a protocol to this set. When added to the protocol set, you can make your content free or code-based test situations and run them on your WebDriver endpoints. Accordingly, the object repository will be consequently loaded with strong repository things.

**Executing data-driven testing**

This is the technical challenges with Selenium WebDriver that it doesn’t have a built-in data-driven testing system. This implies you need to manually associate your mechanized tests to external data sources, read the information out of these sources and execute your test situation with the additional data in a case you need to execute the data-driven testing procedures.

**How to settle it?**  
Data-driven testing is already a vital part of Ranorex. With no extra arrangements, you can pick between a few sorts of information connectors, for example, basic information table, CSV document, SQL database and Excel record. Essentially utilize the information from these outer sources, to consequently execute your test cases with various data sets.

**Managing timing issues**

Another challenges faced in selenium automation testing, particularly when testing dynamic web applications, is taking care of timing issues – for instance, when an inquiry takes more time in giving the required outcomes. In Selenium, you need to do manually execution of a hold up component in code to conquer this issue.

**How to settle it?**  
These [selenium real time issues](http://testorigen.com/services/) can be solved as Ranorex consequently makes scan time-outs for everything in the object repository, giving an in-built safety net for conceivable planning issues. You can proofread the search time-out of a repository item in Ranorex Studio. In option to the consequently included search time-outs, you can expressly wait for a particular component to show up or vanish utilizing a Wait for Exist or a Wait for NotExist activity in the activities table.

**Reporting**

A vital part of any test automation environment is getting a detailed and effortlessly reasonable report for each test execution. When you utilize Selenium WebDriver, there are a few potential outcomes to accomplish a detailing system. All of which, must be executed in code utilizing third-party integrations and it is the one of the biggest among various selenium webdriver challenges.

**How to settle it?**  
Utilizing the Ranorex Selenium WebDriver integration, you don’t need to stress over revealing. When you execute a site test on a WebDriver endpoint, a point by point Ranorex report will be consequently created. This report gives you a thorough outline of the whole test execution stream. As it is a JUnit good report, you can without much of a stretch integrate it into any CI procedure.

Selenium is an outstanding open source testing tool, which gives a strong arrangement of tools that backings fast improvement of test automation for web, desktop based applications. But there are so many challenges faced in selenium automation during software testing. Above mentioned selenium automation challenges are faced most of the time as it is a limited automation tools.

**Challenges faced using selenium automation testing, and how to solve them**

Selenium at times fails to function correctly if a dynamic event or change takes place during the test cycle. A few common problems faced are listed below, along with how to mitigate them with selenium test automation.

1. **Dealing with pop-up windows:** Selenium can sometimes fail to record common popups in web apps. To handle any kind of alert popup, you can apply a getAlert function. Before actually running the script, you must import a package that can generate a WebDriver script for handling alerts. The efficient interface brings with it the following commands: *void dismiss(), void accept (), getText(), void sendKeys(String stringToSend)*. The first two basically click on the “cancel” and “OK” buttons respectively on a popup window.
2. **No event trigger from value changes:** Because Selenium does not initiate events with a change in values, one must do it oneself using fireEvent: selenium.FireEvent(cmbCategory, “onchange”);
3. **Timeout resulting from synchronization problems:**One should ideally use selenium.IsElementPresent(locator) to verify that the object is in a loop with Thread.Sleep
4. **Testing Flash apps:** To automate flash apps with Selenium, one can use Flex Monkium. The application source code must be compiled with the swc files generated by Flex Monkium. Then the app and the Selenium IDE are connected, and the tests can be recorded with IDE.
5. **Unexpected error launching Internet Explorer**. Browser zoom level should be set to 100% by default for the IE browser to overcome this error.
6. **Protected Mode** must be set to the same value error occurs when trying to run Selenium WebDriver on a fresh Windows machine. This issue can be fixed by using capabilities as below when launching IE.