# About the Selenium WebDriver Engine

This topic introduces the Selenium WebDriver engine added in version 9.8.

Sections include:



## Introduction

Selenium makes direct calls to the browser using each browser's native support for automation. It offers:

- Closer fidelity to user actions in recorded scenarios
- Support for Safari
- Support for HTML 5

New browser scenarios use Selenium by default. Any existing browser scenarios (recorded before WebDriver support) can be configured for playback in Selenium (as described in Reconfiguring Legacy Browser Recordings to Play Back in Selenium).

The Selenium engine does not automatically use X virtual framebuffer (Xvfb), but it can be configured manually. Xvfb enables you, for example, to run a Selenium browser scenario in an automated soatestcli job on Linux without a display. For details, see Running Selenium WebDriver Browser Scenarios Using soatestcli on Linux.

## Migration Notes

- Legacy browser recordings might contain actions that are not supported by the Selenium engine. See Reconfiguring Legacy Browser Recordings to Play Back in Selenium for details.
- Since the Selenium WebDriver engine uses the 32-bit version of Internet Explorer, any custom Internet Explorer Executable Path settings (e.g., from **Preferences> Browser> IE Executable Path**) will not be applied when the Selenium engine is used.
- "Wait for Asynchronous Requests" wait conditions are now deprecated. New wait conditions cannot be added, but existing conditions will continue to function until you convert the associated scenarios for Selenium playback. You can replace these wait conditions with "Wait for Interval without Traffic" wait conditions. Be sure to make the interval long enough that the asynchronous request will have occurred.
- __No_Name_:# window format identifiers may be used in legacy browser recordings to identify windows, but this format is not supported in the Selenium WebDriver framework. To migrate these recordings, re-run them, then open them and use the drop-down menu to specify the correct window. You can use the window name or window index to identify a window.

## Deprecated Commands (and Alternatives, when Available)

- **Fireevent:** No longer needed because Selenium WebDriver uses native events to emulate user behavior.
- **Keydown:** Only supports text inputs with the values of "Shift", "Control" or "Alt." Other key modifiers are not needed due to Selenium WebDriver's native emulation of user behavior.
- **Keypress:** No longer needed because Selenium WebDriver uses native events to emulate user behavior.
- **Keyup:** Only supports text inputs with the values of "Shift", "Control" or "Alt." Other key modifiers are not should not be needed due to Selenium WebDriver's native emulation of userbehavior.
- **Mousedown:** No longer needed because Selenium WebDriver uses native events to emulate user behavior.
- **Mousemove:** No longer needed because Selenium WebDriver uses native events to emulate user behavior.
- **Mouseover:** No longer needed because Selenium WebDriver uses native events to emulate user behavior.
- **Mouseup:** No longer needed because Selenium WebDriver uses native events to emulate user behavior.
- **New Browser:** It is no longer possible to spawn a second browser during playback.
- **Other:** It is no longer possible to write custom commands using Other and UserCustomizableOptions.js.
- **Type (Without Focus):** No longer needed because Selenium WebDriver uses native events to emulate user behavior.
- **Type Password (Without Focus):** No longer needed because Selenium WebDriver uses native events to emulate user behavior.

### Migrating Actions that Interact with Hidden Elements

The legacy engine can interact with hidden elements. However, since the Selenium engine tries to simulate real users, it interacts only with visible elements. Thus, if your legacy scenario interacts with hidden elements, you might need to add intermediary steps to reveal a hidden element before you can successfully run that scenario with Selenium.The most common way of doing this is to click an element that makes the hidden element become visible.

An error message such as `Unable to perform user action: Element is not currently visible` might indicate that migration is required (it could also indicate that the application is not behaving as expected).

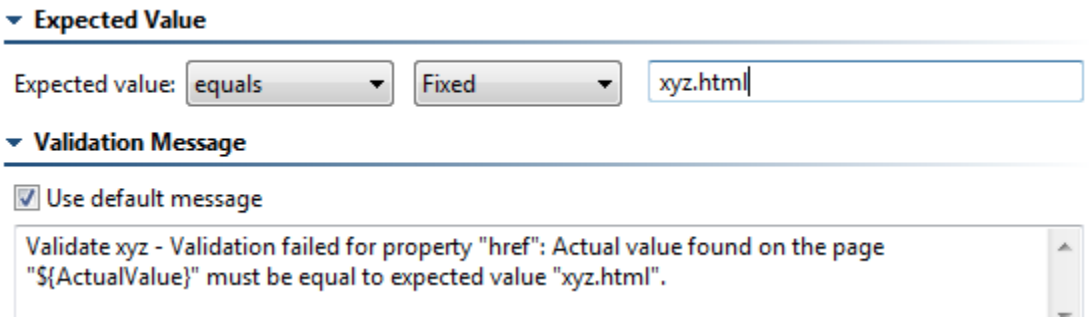
## Migrating Validations that Use Relative URLs

For validations on `href` or `src` attributes, the legacy driver extracts a URL's relative path (if a relative path was given in a locator). However, the Selenium engine extracts the absolute path instead of the relative path. For example, assume that you have browser contents with `<a href="xyz.html">` on a domain `http://localhost:8080`. The validations would be:

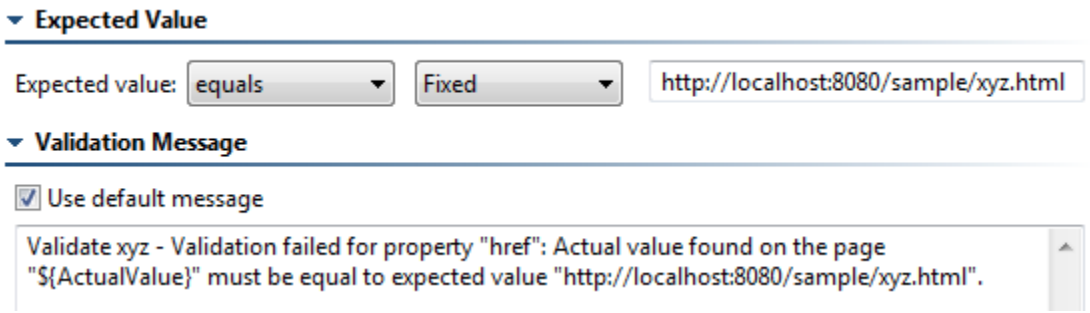
- Legacy: `xyz.html` (the literal href attribute)
- Selenium: `http://localhost:8080/sample/xyz.html` (the absolute path)

An error message such as `Validation failed for property "href": Actual value found on the page "http://localhost:8080/sample/xyz.html" must be equal to expected value "xyz.html"` indicates that migration is required.

To perform the migration, simply update the expected value. For example, if your legacy scenario uses




you could update it to




## Deprecation and Migration of Script Dialog Actions

The following actions that handle dialogs are still allowed, but cannot be created during recording:

- Answeronnextprompt
- Assertalert
- Assertprompt
- Assertconfirmation
- Choosecancelonnextconfirmation

New recordings will use equivalent Selenium commands (e.g., Accept Script Dialog, Dismiss Script Dialog, Type into Script Dialog).

### Migrating Actions for Playback on IE, Chrome, Firefox

Wait for Script Dialog cannot be applied if legacy script dialog actions (e.g., Assertalert, Assertconfirmation, Assertprompt) are used and the Selenium engine is selected for playback. Due to differences between the legacy engine and the Selenium engine, when these legacy actions are played with Selenium, a dialog that has a delayed opening will always fail with a "No alert present" error (regardless of whether a Wait for Script Dialog wait condition has been added to the test).

If you are using the Selenium engine and the newer script dialog actions (Accept Script Dialog, Dismiss Script Dialog), then Wait for Script Dialog will behave as expected.

You can migrate to the newer script dialog actions as follows:

- **Update Alerts:** Replace Assertalert with Accept Script Dialog.
- **Update Confirms:** Legacy scenarios will have Assertconfirmation user actions; they might also have Choosecancelonnextconfirmation actions (if the user is expected to press Cancel). To migrate these scenarios, remove the Choosecancelonnextconfirmation user action, and replace the Assertconfirmation user action with Accept Script Dialog or Dismiss Script Dialog (depending on whether the user is supposed to press OK or Cancel in the dialog).
- **Update Prompts:** Legacy scenarios will have Assertprompt user actions; they might also have Answeronnextprompt actions (if the user is expected to input text). To migrate these scenarios, replace Assertprompt with Accept Script Dialog or Dismiss Script Dialog (depending on whether the user clicks OK or Cancel in the dialog). If the legacy scenario contained Answeronnextprompt, that user action should be removed, and a Type into Script Dialog should be added immediately before the Accept/Dismiss Script Dialog user action.
- **Update Attached/Actions:** Any Browser Validation tools or Wait for Asynchronous Requests attached to an action triggering the alert(e.g. Click 'alert') should be re-attached to the Accept Script Dialog/Dismiss Script Dialog following that action.

### Migrating Actions for Playback on Safari

A Browser Validation tool attached to the action triggering the alert might sometimes work incorrectly during Selenium playback due to a missing wait condition (since Wait for Script Dialog cannot be applied if legacy script dialog actions are used and the Selenium engine is selected for playback). In this case, move the Browser Validation tool to the next action.

# Selenium Known Bugs and Issues

## Compatibility Issues

- Chrome 28+ and Selenium's chromedriver is not compatible with RHEL/!CentOS 6.x. CentOS 7.0 (64-bit) works—although there is a warning reported. This warning can be disabled from the command line with `setsebool -P unconfined_chrome_sandbox_transition 0`

## Selenium Script Dialog Issues

- Selenium WebDriver randomly fails to accept the script dialogs. This occurs when a test tries to open a dialog and the "Wait for Script Dialog" is reporting error that script dialog does not exist during the specified timeout (by default 10 seconds). The issue can be solved by increasing the wait time.
    - https://groups.google.com/forum/#!topic/selenium-developer-activity/It-RyMNBrpw
- In Chrome, Selenium cannot interact with onBeforeUnload dialogs that open due to a Navigate, Go Back, or Go Forward action.
    - https://code.google.com/p/chromedriver/issues/detail?id=29
- In Chrome and Internet Explorer, Selenium cannot interact with onBeforeUnload dialogs that open when calling "Close" on the only browser window.
    - Chrome: https://code.google.com/p/chromedriver/issues/detail?id=901
    - Internet Explorer: https://code.google.com/p/selenium/issues/detail?id=7895
- Selenium WebDriver fails to accept an alert that is opened on a new window.
    - https://code.google.com/p/selenium/issues/detail?id=7807
- Modal dialogs (those opened using the JavaScript function "showModalDialog") are not currently supported in SOAtest's Selenium WebDriver framework for Chrome or Safari. Web applications that rely on this functionality should use Parasoft's native engine or use Internet Explorer or Firefox.

## Other Issues

- For the "click" user action, key modifiers ("Shift," "Ctrl," and "Alt") are supported only in Chrome due to an issue in Selenium WebDriver. See https://code.google.com/p/selenium/issues/detail?id=4385.
- When multiple windows are opened by one user action, the Selenium WebDriver can return a wrong order of window handlers for Chrome and Internet Explorer. This can happen only in the case of unnamed windows. The workaround is to use named windows.
- Cannot click on area elements in Safari or Internet Explorer. See https://code.google.com/p/selenium/issues/detail?id=2354.
- In Internet Explorer, if an application overrides the default "fireEvent" implementation on an element with a new implementation, Selenium will trigger the new implementation when performing actions on that element. This differs from default browser behavior, where the default "fireEvent" implementation is called even when the "fireEvent" method is changed by a web application.
- Selenium does not support playing back scenarios in headless mode. If headless mode is selected, the scenarios will still play back in visible mode.

# Safari-Specific Issues

## Unsupported Commands

The following commands are not supported in the Selenium WebDriver engine in Safari. This is due to a limitation in the SafariDriver (https://code.google.com/p/selenium/issues/detail?id=4136)

- DoubleClick
- Keydown
- Keyup
- Dragdrop
- Go Back
- Go Forward

## Unsupported Functionality

Due to Selenium limitations, the following SOAtest functionality is impacted:

- Traffic is not recorded when playing back scenarios.
- NTLM/Digest/Basic authentication is not supported.
- Configure and Validate for Load Test are not supported.
- Attaching tools to HTTP traffic is not supported.
- The "Wait for Interval without Traffic" wait condition will always succeed after the configured interval has elapsed, regardless of whether traffic is being sent by the browser or server.
- The "Wait for Asynchronous Requests" wait condition does not work.
- Sites that use untrusted certificates are not supported.

## Other Issues

- For Safari 7+, the SafariDriver does not automatically install the browser extension on each run because keychain authorization is required to install extensions. The workaround is to manually install the extension.
    - https://code.google.com/p/selenium/issues/detail?id=8514
    - https://code.google.com/p/selenium/source/detail?r=047ab5f8f98d
- For Safari 7.10 (released mid-September 2014), it's necessary to go into Safari preferences and enable the WebDriver plugin.
    - https://code.google.com/p/selenium/issues/detail?id=7933
- For Safari, wait for page load timeout is supported only for navigation action. The WebDriver pageloadTimeout for Safari is not supported.
    - https://code.google.com/p/selenium/issues/detail?id=413
    - https://code.google.com/p/selenium/issues/detail?id=687
- Alerts in Safari are always suppressed and are not reported as exceptions (users will never get 'unexpected script dialog'). Existing Accept Script Dialog/Dismiss Script Dialog and Type into Script Dialog tools will always succeed. The choice of Accept or Dismiss Script Dialog will have the intended effect on confirm dialogs and Type into Script Dialog will have its intended effect on prompt dialogs; however these dialogs are suppressed from view.
- Setting the value of file inputs is not supported.
    - https://code.google.com/p/selenium/issues/detail?id=4220
- If you focus on a text input element (using JavaScript), change its value, then focus on another element, the "change" event is not fired.
    - https://code.google.com/p/selenium/issues/detail?id=4061
- Windows opened with window.open() will fail to open when playing back a scenario in Safari if "Block pop-up windows" is enabled. You can disable "Block pop-up windows" to work around this issue.
    - https://code.google.com/p/selenium/issues/detail?id=3693
- If Safari is set to request the same URI with a fragment, it will cause WebDriver to hang.
    - https://code.google.com/p/selenium/issues/detail?id=7176
- Rich text editors are not supported. This is also true of the legacy engine.
    - https://code.google.com/p/selenium/issues/detail?id=4467
- When calling switchTo().window(), the Safari browser will actually switch to the window in the GUI, which may be disorienting.

# Using Selenium WebDriver for Legacy Browser Recordings

There are two options for running legacy browser recordings (created prior to 9.8) using Selenium WebDriver:

- Reconfigure them to use Selenium WebDriver—this involves changing the test suite configuration.
- Running them with Selenium WebDriver on an ad-hoc basis—this does not require changes to the test suite configuration; instead, the playback engine setting is overridden at the Test Configuration level.

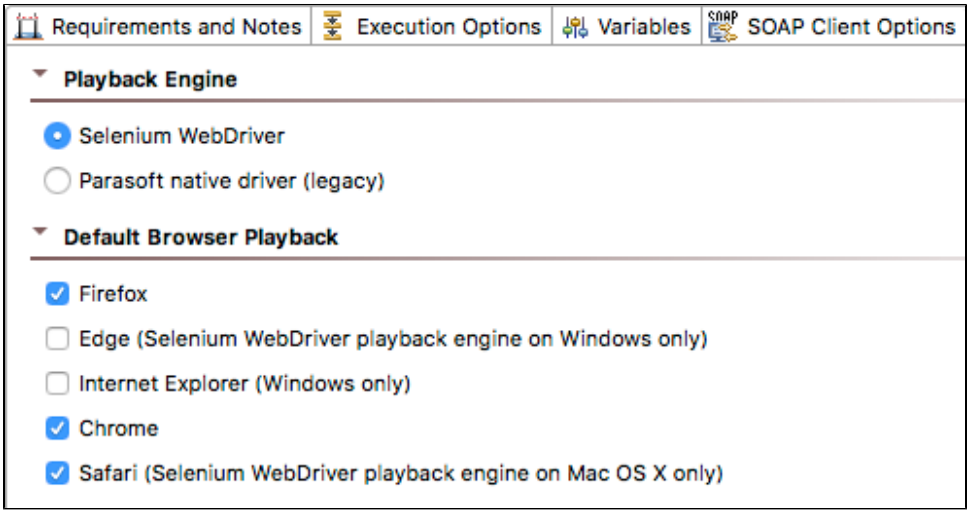
Note that legacy scenarios might contain actions that are deprecated; if so, a warning dialog will open when you select **Use Selenium for Playback**. If you see this dialog, review the scenario and modify the actions as needed. See Selenium Known Bugs and Issues for details. Additionally, if any legacy scenario fires a beforeunload event, you will need to add new user actions (e.g., Accept Script Dialog, Dismiss Script Dialog) to handle the dialogs that appear.

## Reconfiguring Legacy Browser Recordings to Play Back in Selenium

To reconfigure legacy web scenarios to play back using the Selenium WebDriver engine:

1. Open the test suite for the scenario you want to convert.

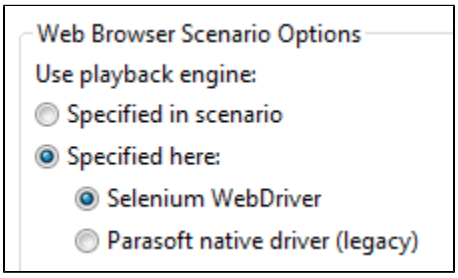
2. Open the **Browser Playback Options** tab and enable the **Selenium WebDriver** Playback Engine option.




## Running a Test Configuration that Uses Selenium

By default, Test Configurations are set to play web scenarios using the playback engine specified at the test suite level. This allows you to use a single Test Configuration to execute a mixture of tests configured for Selenium and tests configured for the legacy engine.

However, there are some cases where you might want to override the test suite's playback engine setting—for example, if you want to see how your legacy test scenarios work with Selenium before you reconfigure them for Selenium. In these situations, you can run any Test Configuration configured to use the Selenium WebDriver playback engine. This option is set in the **Execution> Functional** tab.

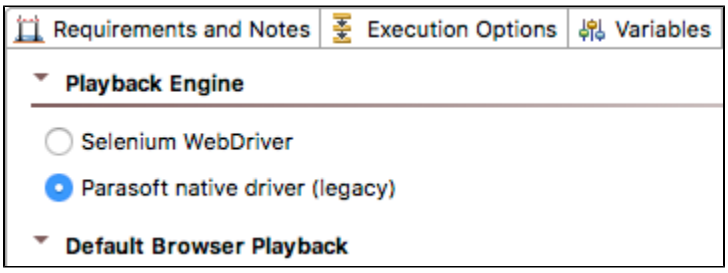



Any tests run with this configuration will use Selenium WebDriver for playback—regardless of what engine is configured at the test scenario level.

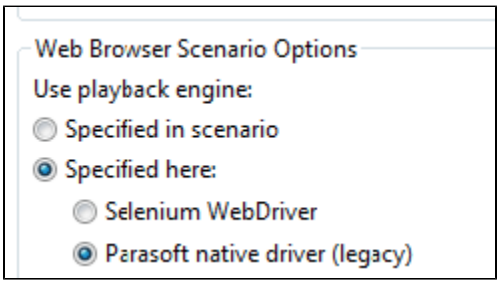
# Using the Legacy Native Driver

If you prefer to use the legacy native driver for a specific scenario:

1. Open the test suite for the scenario you want to use the legacy engine.
2. Open the **Browser Playback Options** tab and enable the **Parasoft native driver (legacy)** Playback Engine option.



Alternatively, you can create and apply a Test Configuration that runs web scenarios with the native driver—regardless of what engine is configured at the test scenario level.  This option is set in the **Execution> Functional** tab.




# Manually Upgrading to a New Version of Selenium WebDriver

You can upgrade SOAtest's Selenium WebDriver version before it is available in a new SOAtest release, but note that using a newer version of Selenium WebDriver than the version shipped with SOAtest is not officially supported or tested by Parasoft. As a result, you may encounter incompatibilities when running the newer version of Selenium WebDriver.

To upgrade, you need to:

1. Upgrade the Selenium client (which also upgrades WebDriver support for Firefox playback). This step is required, no matter what browser(s) you want WebDriver to use for playback. For details, see Upgrading Selenium Client Libraries (Includes Updating WebDriver Support for Firefox).
2. Upgrade WebDriver's playback support for the desired browser(s)—Chrome, Internet Explorer, and/or Safari. See the following sections:


- Upgrading Selenium Client Libraries (Includes Updating WebDriver Support for Firefox)
- Upgrading WebDriver Support for Chrome (ChromeDriver)
- Upgrading WebDriver Support for Internet Explorer (InternetExplorerDriver)
- Upgrading WebDriver Support for Safari (SafariDriver)
- Upgrading WebDriver Support for Microsoft Edge (EdgeDriver)


## Upgrading Selenium Client Libraries (Includes Updating WebDriver Support for Firefox)

The following is always the first step for updating Selenium WebDriver—no matter what browser you want it to use for playback. WebDriver support for Firefox will be upgraded as part of this process.

1. Go to http://search.maven.org.
2. Search for and download the following files from maven.org (use the **jar** download link for each).
    - selenium-api-{x.y.z}.jar
    - selenium-chrome-driver-{x.y.z}.jar
    - selenium-firefox-driver-{x.y.z}.jar
    - selenium-ie-driver-{x.y.z}.jar
    - selenium-edge-driver- {x.y.z}.jar
    - selenium-java-{x.y.z}.jar
    - selenium-remote-driver-{x.y.z}.jar
    - selenium-safari-driver-{x.y.z}.jar
    - selenium-support-{x.y.z}.jar


3. Remove the version suffix (e.g., {x.y.x}) from each of the downloaded jar files. For example, `selenium-api-2.45.0.jar` should be renamed to `selenium-api.jar`.

4. In the SOAtest installation directory, run the `update.bat` script (for Windows) or `update` script (for Linux and Mac) with the `-patch` argument and the path to each of the jars you downloaded and renamed in the previous steps. Running this script will replace the version in the installation with the file referenced by the `patch` argument. It will also back up the replaced file in the installation by appending a `.bak` extension to the file.
    - For Windows: Run `update.bat -patch /path/to/[name_of_file].jar`
    - For Linux or Mac: Run `./update -patch /path/to/[name_of_file].jar`. Note that on some versions of Mac, you might need to manually copy the Selenium drivers.

For example, this is what the script output would look like on Windows when you run the script to upgrade the `selenium-java.jar` file and your file is downloaded to the `%DOWNLOADS%` directory:

```
c:\Program Files\Parasoft\SOAtest\9.9>update -patch %DOWNLOADS%\selenium-java.jar INFO: Patching file: C:
\Program Files\Parasoft\SOAtest\9.9\eclipse\plugins\com.parasoft.xtest.libs.web_9.9.0.20141024\root\lib-
java\org.seleniumhq.selenium\selenium-java.jar
Update completed successfully
```

After you have completed the above steps, you can confirm that Selenium WebDriver for Firefox was upgraded by going to the Firefox Add-Ons page and checking the version number.

## Upgrading WebDriver Support for Chrome (ChromeDriver)

To upgrade WebDriver's support for Chrome, update ChromeDriver as follows:

1. If you have not already done so, upgrade the Selenium client libraries as described in Upgrading Selenium Client Libraries (Includes Updating WebDriver Support for Firefox).
2. Go to the ChromeDriver downloads page at http://chromedriver.storage.googleapis.com/index.html.
3. Download the latest release (you can click the LATEST_RELEASE link to find out what version is the latest).
4. Download the ChromeDriver .zip file for your architecture:
    - For Windows 64-bit and 32-bit: chromedriver_win32.zip
    - For Mac 64-bit and 32-bit: chromedriver_mac32.zip
    - For Linux 64-bit: chromedriver_linux64.zip
    - For Linux 32-bit: chromedriver_linux32.zip
5. Extract the `chromedriver.exe` file from this .zip.
6. In the SOAtest installation directory, run the `update.bat` script (for Windows) or `update` script (for Linux and Mac) with the `-patch` argument and the path to the `chromedriver.exe` file. Running this script will replace the version in the installation with the file referenced by the `patch` argument. It will also back up the replaced file in the installation by appending a `.bak` extension to the file.
    - For Windows: Run `update.bat -patch /path/to/chromedriver.exe`
    - For Linux or Mac: Run `./update -patch /path/to/chromedriver`

## Upgrading WebDriver Support for Internet Explorer (InternetExplorerDriver)

To upgrade WebDriver's support for Internet Explorer, update InternetExplorerDriver as follows:

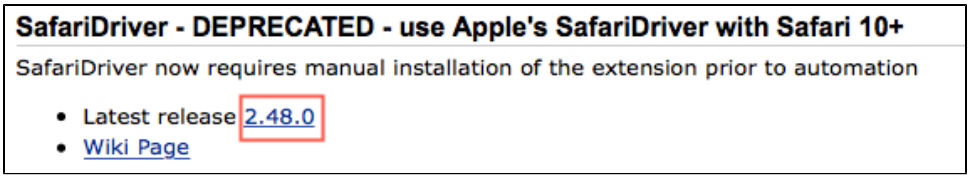
1. If you have not already done so, upgrade the Selenium client libraries as described in Upgrading Selenium Client Libraries (Includes Updating WebDriver Support for Firefox).
2. Go to the Selenium Download page (http://www.seleniumhq.org/download/), then under the **Internet Explorer Driver Server** section, download the **32 bit Windows IE** version—*even if you're running 64-bit Windows*.
3. Extract the `IEDriverServer.exe` file from the.zip.
4. In the SOAtest installation directory, run the `update.bat` script with the `-patch` argument and the path to the `IEDriverServer.exe` file you downloaded (e.g., `update.bat -patch /path/to/IEDriverServer.exe`). Running this script will replace the version in the installation with the file pointed to by the `patch` argument. It will also back up the replaced file in the installation by appending a `.bak` extension to the file.

SOAtest will now use the newer version of InternetExplorerDriver for playback.

## Upgrading WebDriver Support for Safari (SafariDriver)

Selenium's SafariDriver is deprecated in Safari 10 and later. Update to the latest version of Safari and use Apple's SafariDriver. If you cannot upgrade Safari, you can download and install the latest Safari WebDriver Extension as follows:

1. Open the Safari browser and go to the Selenium Download page (http://www.seleniumhq.org/download/)
2. Under the SafariDriver section, download the latest release of `SafariDriver.safariextz`.




3. Click on the downloaded file to install the Safari WebDriver extension.

## Upgrading WebDriver Support for Microsoft Edge (EdgeDriver)

1. If you have not already done so, upgrade the Selenium client libraries as described in Upgrading Selenium Client Libraries (Includes Updating WebDriver Support for Firefox).
2. Go to the Microsoft WebDriver download page (https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/) and download the driver for your version of Edge (see Browser Support for version support details).
3. Save the driver to the following directory:

   <SOATEST_INSTALL>\eclipse\plugins\com.parasoft.xtest.libs.web_<version>\root\browsers\webdriver\edge\x86\

# Running Selenium WebDriver Browser Scenarios Using soatestcli on Linux

When SOAtest runs a browser test, it starts the browser used for testing (e.g., Firefox or Chrome) as a separate process. On Linux, the browser process must connect to an X display. When running SOAtest from the UI, this works seamlessly because the browser uses the same display as SOAtest. When running soatestcli through an automated job or in a terminal not connected to a physical display, extra setup is required for browser scenarios that use the Selenium WebDriver playback engine.

## Start a Virtual X Server (Xvfb) to Create a Display

Install Xvfb on the machine where soatestcli will run. (Alternatively, you can use the Xvfb_Linux that is packaged with SOAtest, but you may need to experiment with options to get these working. See Getting Xvfb working independently of SOAtest for more information.)

The easiest way to use Xvfb is to start a single process that will be used by all SOAtest runs. To do this:

1. Log in to the machine where soatestcli will run (w.g., by ssh).
2. Run a command such as the following:
   ```
   $ nohup Xvfb :99 > /dev/null 2>&1 &
   ```

This starts Xvfb on display :99.  All logging information is discarded (sent to /dev/null). The "nohup" command ensures that the Xvfb process will continue running after you log out.

Note that each time you restart the machine, you will need to remember to start another Xvfb process before any soatestcli runs. If you want to avoid this, you can use scripting to start and stop Xvfb as needed.

## Set the DISPLAY Environment Variable

The browser uses the DISPLAY environment variable to determine what X display to use.

How to set the environment variable will depend on how you have configured your automated jobs to run. What is important is that the DISPLAY variable is set to the same value that you use when starting Xvfb. For the above example, that value is ":99".

If you have created a shell script—or if you need to run soatestcli on the server on an ad-hoc basis—then you can directly set the variable in the script.  For example, if using bash:

```
$ DISPLAY=:99 soatestcli ...options...
```

Or:

```
$ export DISPLAY=:99
$ soatestcli ...options...
```

If you are using Jenkins to run automated jobs, then you can set the DISPLAY variable for your jobs by configuring the environment of a Jenkins node. (Be aware that this will set the variable for *all* jobs that run on the node, not only your SOAtest jobs.) To do this:

1. Browse to the Jenkins web page.
2. Log in.
3. Click the Jenkins node on which soatestcli will run.
4. Click to configure the node.
5. In the "Node Properties" section, add an environment variable with the name "DISPLAY" and the desired value (e.g., ":99").