

Using profiles in selenium

Custom Firefox profile for Selenium

Why create a new Firefox profile

A profile in Firefox is a collection of bookmarks, browser settings, extensions, passwords, and history; in short, all of your personal settings. When you want to run automation reliably on a Firefox browser:

- You should be consistent with the profile you use on all development and test execution machines. If you used different profiles everywhere, the SSL certificates you accepted or the plug-ins you installed would be different and that would make the tests behave differently on the machines.
- There are several times when you need something special in your profile to make test execution reliable. The most common example is a SSL certificate settings or browser plug-ins that handle self-signed certs. It makes a lot of sense to create a profile that handles these special test needs and packaging and deploying it along with the test execution code.
- You should use a very lightweight profile with just the settings and plug-ins you need for the execution. Each time Selenium starts a new session driving a Firefox instance, it copies the entire profile in some temporary directory and if the profile is big, it makes it, not only slow but unreliable as well.

Creating a new profile using the Profile Manager

Here are the steps:

- Close the application and make sure that it is not running in the background.
- Start the Profile Manager
 - Windows
 - Open the Windows “Start” menu, select “Run” (on Windows Vista, use “Start Search” or enable the Run box, as described here) then type and enter one of the following:
`firefox.exe -P`

or run the Profile Manager

```
"C:\Program Files\Mozilla Firefox\firefox.exe" -  
profilemanager
```

- o Linux

- Open the terminal and execute cd firefox program directory. If you don't know where firefox is installed, you can always do

```
locate firefox
```

- Then execute:

```
./firefox -profilemanager
```

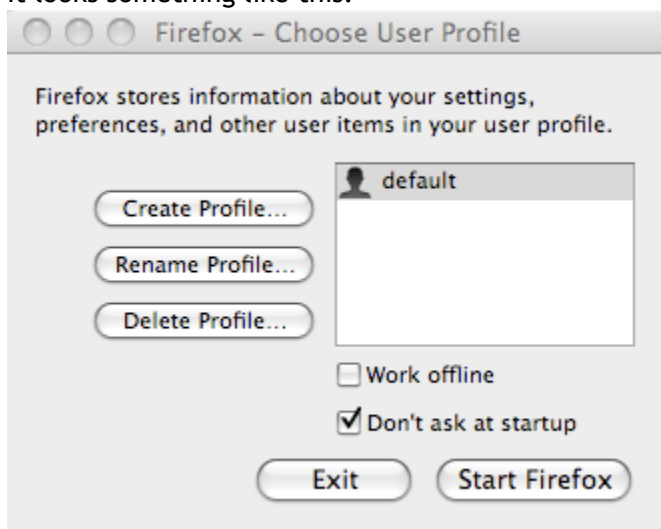
- o Mac OS X

- Assuming the program is installed in the "Applications" folder, launch Terminal ("Applications -> Utilities -> Terminal")

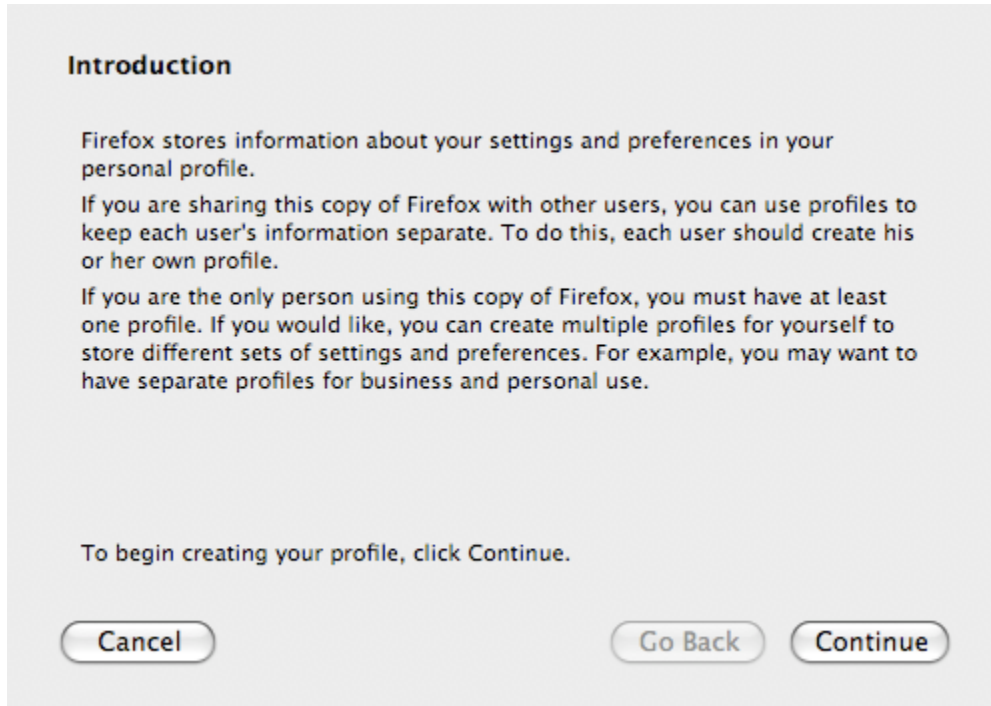
- Enter the command starting with / after the prompt in Terminal:
`/Applications/Firefox.app/Contents/MacOS/firefox-bin -
profilemanager`

Adding -bin after the application name is necessary on some systems and application versions. If you have problems starting the Profile Manager with one of the above commands, try again using just the application name.

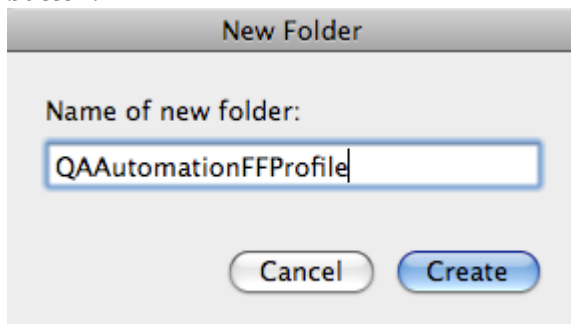
It looks something like this:



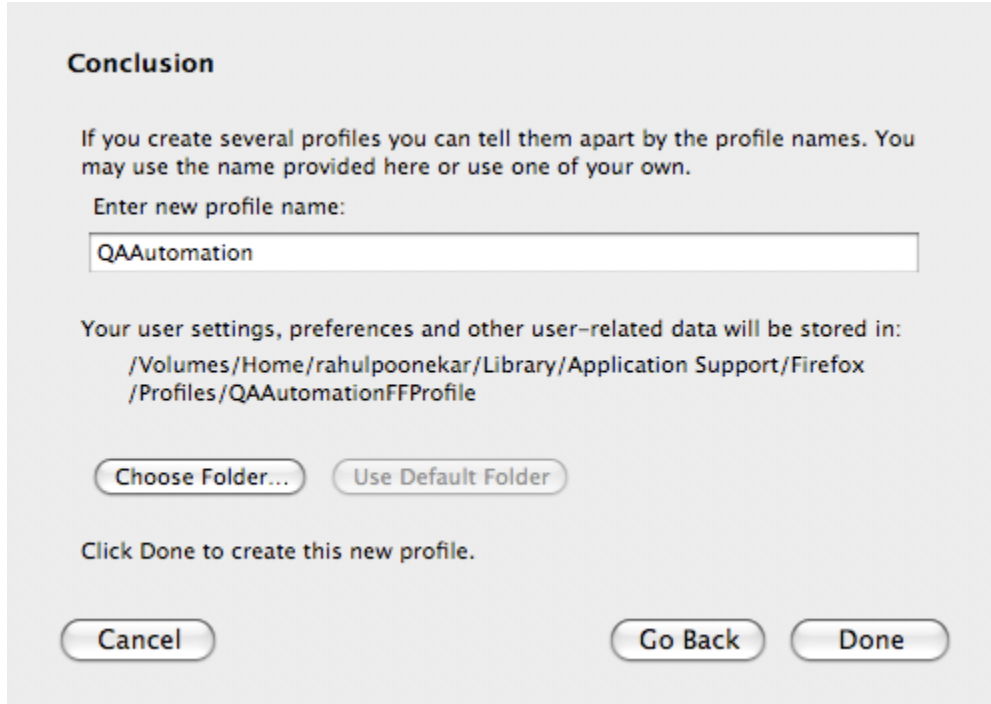
- Click the “Create Profile” button and click “Continue”



- Set the folder name. Here I am setting it as “QAAutomationFFProfile”. Click the “Create” button.

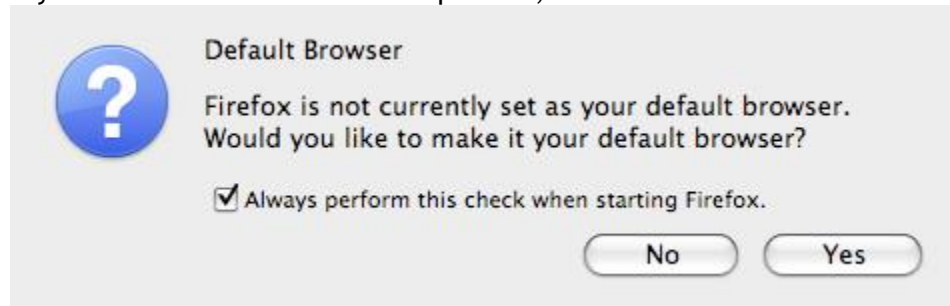


- Now set this new profile name. I am setting it as “QAAutomation”. Click “Done”.



- Create your profile:
 - Start the firefox session using the firefox executable with the `-P "QAAutomation"` option. For example on Mac it would be:
`/Applications/Firefox.app/Contents/MacOS/firefox-bin -P "QAAutomation"`

- If you see the “Default Browser” question, click Yes.



- This will create all the files for your new profile in the folder you specified earlier.

Make it part of your selenium session

In Selenium2 - WebDriver

You can now add the path to your profile that you would use while instantiating the FirefoxDriver:

```
File profileDirectory = new File(profileDirectory);
FirefoxProfile profile = new FirefoxProfile(profileDirectory);
WebDriver webDriver = new FirefoxDriver(profile);
```

How to use firefox profile while working with remote webdriver

```
static void Main(string[] args)
{
    FirefoxProfile profile = new FirefoxProfile();
    //profile.SetPreference("webdriver.log", @"C:\FirefoxDriver.txt");
    profile.SetPreference("plugins.hide_infobar_for_missing_plugin", true);

    var capabilities = DesiredCapabilities.Firefox();
    capabilities.IsJavaScriptEnabled = true;
    capabilities.SetCapability(FirefoxDriver.ProfileCapabilityName,
    profile.ToBase64String());

    IWebDriver driver = new RemoteWebDriver(capabilities) { Url = "http://google.com"
};

    Console.ReadLine();
    driver.Quit();
}
```

How to update the profile

Installing an extension/plugin

1. Download firebug-1.7.3-fx.xpi to your disk in advance
2. Use code like:

```
File ext = new File("firebug-1.7.3-fx.xpi");
FirefoxProfile profile = new FirefoxProfile();
if (ext.exists()) {
    profile.addExtension(ext);
    profile.setPreference("extensions.firebug.currentVersion", "1.7.3"); //avoid startup screen
    profile.setPreference("extensions.firebug.console.enableSites", true); //enable console
}
```

```
//some more prefs:
profile.setPreference( "app.update.enabled", false);
profile.setPreference( "browser.tabs.autoHide", true);

//use this profile:
DesiredCapabilities capability = DesiredCapabilities.firefox();
capability.setCapability(FirefoxDriver.PROFILE, profile); //FirefoxDriver.PROFILE =
"firefox_profile";

//run driver with this profile this profile:
RemoteWebDriver driver = new RemoteWebDriver(capability);
```

Google chrome profiles

Capabilities (aka ChromeOptions)

Capabilities are options that you can use to customize and configure a ChromeDriver session. This page documents all ChromeDriver supported capabilities and how to use them.

There are two ways to specify capabilities. The first is to use the ChromeOptions class. Alternatively, if your client library does not have a ChromeOptions class (like the selenium ruby client), you can specify the capabilities directly as part of the DesiredCapabilities.

Using the ChromeOptions class

You can create an instance of ChromeOptions, which has convenient methods for setting ChromeDriver-specific capabilities. You can pass the ChromeOptions object directly into the ChromeDriver constructor:

```
ChromeOptions options = new ChromeOptions();
options.addExtensions(new File("/path/to/extension.crx"));
ChromeDriver driver = new ChromeDriver(options);
```

Alternatively, you can add the options to an already existing DesiredCapabilities object, which is useful when need to specify other WebDriver capabilities not specific to ChromeDriver.

```
DesiredCapabilities capabilities = DesiredCapabilities.chrome();
// Add the WebDriver proxy capability.
Proxy proxy = new Proxy();
proxy.setHttpProxy("myhttpproxy:3337");
capabilities.setCapability("proxy", proxy);

// Add ChromeDriver-specific capabilities through ChromeOptions.
ChromeOptions options = new ChromeOptions();
options.addExtensions(new File("/path/to/extension.crx"));
capabilities.setCapability(ChromeOptions.CAPABILITY, options);
ChromeDriver driver = new ChromeDriver(capabilities);
```

Using DesiredCapabilities directly

The ChromeOptions class uses DesiredCapabilities underneath. To use DesiredCapabilities directly, you need to know the name of the capability and the type of value it takes. See the full list further below.

Java

```
Map<String, Object> chromeOptions = new Map<String, Object>();
chromeOptions.put("binary", "/usr/lib/chromium-browser/chromium-browser");
DesiredCapabilities capabilities = DesiredCapabilities.chrome();
capabilities.setCapability(ChromeOptions.CAPABILITY, chromeOptions);
WebDriver driver = new ChromeDriver(capabilities);
```

Ruby

```
caps = Selenium::WebDriver::Remote::Capabilities.chrome("chromeOptions" =>
{"args" => [ "--disable-web-security" ]})
driver = Selenium::WebDriver.for :remote, url: 'http://localhost:4444/wd/hub'
desired_capabilities: caps
```

Common use cases

Use custom profile (also called user data directory)

By default, ChromeDriver will create a new temporary profile for each session. At times you may want to set special preferences or just use a custom profile altogether. If the former, you can use the 'chrome.prefs' capability (described later below) to specify preferences that will be applied after Chrome starts. If the latter, you can use the 'user-data-dir' Chrome command-line switch to tell Chrome which profile to use:

```
ChromeOptions options = new ChromeOptions();
options.addArguments("user-data-dir=/path/to/your/custom/profile");
```

You can create your own custom profile by just running Chrome (on the command-line or through ChromeDriver) with the 'user-data-dir' switch set to some new directory. If the path doesn't exist, Chrome will create a new profile in the specified location. You can then modify the profile settings as desired, and ChromeDriver can use the profile in the future. Goto `chrome://version` in the browser to see what profile Chrome is using.

Start Chrome maximized

```
ChromeOptions options = new ChromeOptions();
options.addArguments("start-maximized");
```

Using a Chrome executable in a non-standard location

```
ChromeOptions options = new ChromeOptions();
options.setBinary("/path/to/other/chrome/binary");
```

Set a Chrome preference

```
ChromeOptions options = new ChromeOptions();
Map<String, Object> prefs = new HashMap<String, Object>();
```

```
prefs.put("profile.default_content_settings.popups", 0);
options.setExperimentalOptions("prefs", prefs);
```

List of recognized capabilities

This is a list of all the WebDriver-standard capabilities that ChromeDriver supports:

- [proxy](#)
- [loggingPrefs](#)

This is a list of all the Chrome-specific desired capabilities, which all are under the chromeOptions dictionary. If possible, use the ChromeOptions class instead of specifying these directly.

Name	Type	Default	Description
args	list of strings		List of command-line arguments to use when starting Chrome. Arguments with an associated value should be separated by a '=' sign (e.g., ['start-maximized', 'user-data-dir=/tmp/temp_profile']). See here for a list of Chrome arguments.
binary	string		Path to the Chrome executable to use (on Mac OS X, this should be the actual binary, not just the app. e.g., '/Applications/Google Chrome.app/Contents/MacOS/Google Chrome')
extensions	list of strings		A list of Chrome extensions to install on startup. Each item in the list should be a base-64 encoded packed Chrome extension (.crx)
localState	dictionary		A dictionary with each entry consisting of the name of the preference and its value. These preferences are applied to the Local State file in the user data folder.
prefs	dictionary		A dictionary with each entry consisting of the name of the preference and its value. These preferences are only applied to the user profile in use. See the 'Preferences' file in Chrome's user data directory for examples.
detach	boolean	false	If false, Chrome will be quit when ChromeDriver is killed, regardless of whether the session is quit. If true, Chrome will only be quit if the session is quit (or closed). Note, if true, and the session is not quit, ChromeDriver cannot clean up the temporary user data directory that the running Chrome instance is using.
debuggerAddress	string		An address of a Chrome debugger server to connect to, in the form of <hostname/ip:port>, e.g. '127.0.0.1:38947'

excludeSwitches	list of strings		List of Chrome command line switches to exclude that ChromeDriver by default passes when starting Chrome. Do not prefix switches with --.
minidumpPath	string		Directory to store Chrome minidumps


This is a list of all the Chrome-specific returned capabilities (i.e., what ChromeDriver returns when you create a new session)

Name	Type	Description
chrome.chromedriverVersion	string	version of ChromeDriver
userDataDir	string	path to user data directory that Chrome is using; note, this is inside a 'chrome' dictionary

ChromeDriver server command line arguments

Run chromedriver with -h or --help to see command line arguments for your version.

Create a new browser user profile

1. Exit Google Chrome completely.
2. Enter the keyboard shortcut **Windows key**  +E to open Windows Explorer.
3. In the Windows Explorer window that appears enter the following in the address bar.
 - o **Windows XP:** %USERPROFILE%\Local Settings\Application Data\Google\Chrome\User Data\
 - o **Windows Vista/ Windows 7/ Windows 8:** %LOCALAPPDATA%\Google\Chrome\User Data\
4. Locate the folder called "Default" in the directory window that opens and rename it as "Backup default."
5. Try opening Google Chrome again. A new "Default" folder is automatically created as you start using the browser.

If you wish, you can transfer information from your old user profile to your new one. However, **this action is not recommended**, since a part of your old profile may be corrupt. With that in mind, to transfer your old bookmarks, copy the "Bookmarks.bak" file from the "Backup default" folder to your new "Default" folder. Once moved, rename the file from "Bookmarks.bak" to "Bookmarks" to complete the migration. All other browser data will remain in the "Backup default" folder, but you won't be able to transfer it to your new profile.