StormRunner Functional 1.4 Field Enablement

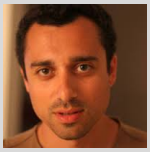
Some of the use-cases you're about to execute require SRF Client Id and Client Secret.

If you haven't created a pair yet, refer to the online documentation to learn how to generate them (it is recommended to store them for future use).
Manage Remote Access:
https://admhelp.microfocus.com/srf/en/1.40/Content/Config.htm#Generate

Manage Tunnels: https://admhelp.microfocus.com/srf/en/1.40/Content/Config.htm#Tunneling

Use Case #1 – Native Cloud Dev, advanced scripting & add parameters

Persona: Dani Hovav, **Software Engineer**
8.5y of experience as a Software Engineer
Works at Bosem
Lives in Holon, Israel

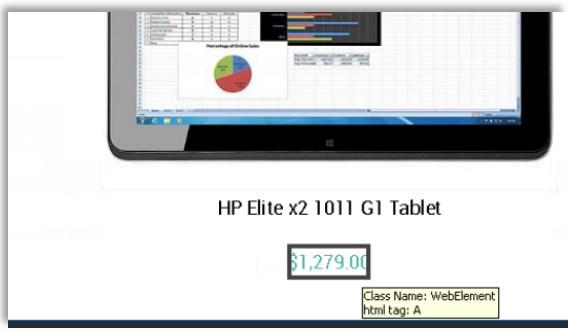


Preparations / Prerequisites

- N/A

Customer Scenario

1. Start by recording a new web script in SRF (Assets→ Record).
2. Go to Advantage (http://tinyurl.com/hpe-shop)
3. Press on tablets
4. Now, record a new checkpoint by pressing on the button
5. Notice that once you are in the checkpoint mode the object is being highlighted
6. Once you pressed the object SRF adds 2 default checkpoints:
   expect(<object>.toEqual(<tag>); // validate the tag
   expect(<object>.innerText()).toEqual(<inner text>) // validate the inner text
7. Mouse over and press on the "$1279.00" link under the 1st tablet:



8. Continue recording your flow
9. Once you finish save the script and give it a name
10. Once saved, it is added to the assets. Press on it to see a preview of it
11. On the left part you can see the parameters support
12. Press on the "Open in Player" button
13. Search for the checkpoint and change the variable name to a meaningful name:

var $127900 = browser.$(Web.Element({innerText: '$1,279.00 ', tagName: 'A'}));
Change to:
var link = browser.$(Web.Element({innerText: '$1,279.00 ', tagName: 'A'}));
And the corresponding checkpoints:
expect(link.tagName()).toEqual('A'); // validate it is a link
expect(link.innerText()).toEqual('$1,279.00 ') // validate the inner text

Now delete the tagName checkpoint and change the innerText to check that it contains the text and not equal (we are using the toContain):
expect(link.innerText()).toContain('1,279.00');

14. Now we need to define parameters. Go back to editing you script
15. In SRF, we are using environment variable in order to let SRF replace the values in runtime
16. Locate the browser.navigate and replace the value with the following:
    browser.navigate(process.env.url);
    Save you script
17. Now, in the script details we'll need to define the key and value to populate the parameter:

18. Note: the name of the parameter must be exactly with what you defined in the script
19. Press on Add (the green V)
20. Notice that now in the asset list there is a "P" annotation on the script to illustrate that the script is define with parameters. Later we'll be able to override the parameters in the Jenkins plugin
21. Now, go to the automation and examine the script again. Notice that you can see the parameter and only change his value
22. Run the test
23. A preferred way to work with parameters is to define a default in case no value was sent. This is in order to protect the user from any "human-errors". In order to do that you'll need to manually define the default.
24. Go back to editing you script
25. Before the 1st describe section define the following:
    var url = process.env.url;

    And now to protect yourself with a default value:
    if (url===undefined)
    url="http://www.advantageonlineshopping.com/";
26. That's it.
27. The last option you can do is to download the asset to your local machine and to continue working on it with your existing IDEs
28. Go to the script preview and press the button to download it

Use Case #2 – Multiple Parallel Cloud Executions

Persona: Patrick Montgomery, **DevTest Engineer**
18y of experience in testing
Works at Solid Future
Lives in Orlando, FL, USA

Preparations / Prerequisites

- Recorded/Uploaded Scripts

Customer Scenario

1. Go to the Automation tab and create a new test
2. Give it a name and a description
3. Add a tag to it
4. Press on Script
5. Add 4 different scripts. Each script should represent a different business flow (login, add to cart, checkout, logout)
6. Go to the environment and select 3 different environments
7. Run the test
8. Now, SRF will run the scripts in parallel on the different environments that you defined
9. Go to Results
10. Press on the row of your test
11. Notice in the table that the 1st environments are already running (this is based on the license) and the rest are in status pending until an available slot becomes (this is done automatically by SRF)
12. Let the test finish

Use Case #3 – Root Cause Analysis, Deep-Dive Analysis

Persona: Kirk Mitchell, **Test Engineer**
4y of experience as a TE
Works at Custom Lawn Care
Lives in Stoke,UK

Preparations / Prerequisites

- Completed results from previous use case

Customer Scenario

1. Go the results and press on the row of your results
2. In this view you see an overview of how your tests was executed and a cross-browser/device comparison in a high level
3. In case your test had any errors or warning you'll see them at the bottom part
4. Examine the left side of the view to see the meta-data of the test run
5. Press on one of the scripts to drill into the script level overview
6. Press on one of the steps
7. Examine its resources (script, snapshot, etc.)
8. Review a steps details when pressing the **i** icon
9. In order to let you test the behavior between the different environments you can move into a special "compare mode" by pressing on the  button

10. By default, the 2 first environments are selected. You can switch between the environments to compare by pressing on the circle below the environment
11. After pressing on the environments, you can now compare between the different resources.
12. Scroll between the different resources and the different steps
13. Try to find a specific environment that your AUT is performing slower than other, or failed script. Try to pinpoint the specific action that causes this

Use Case #4 – single test execution with Jenkins & SRF

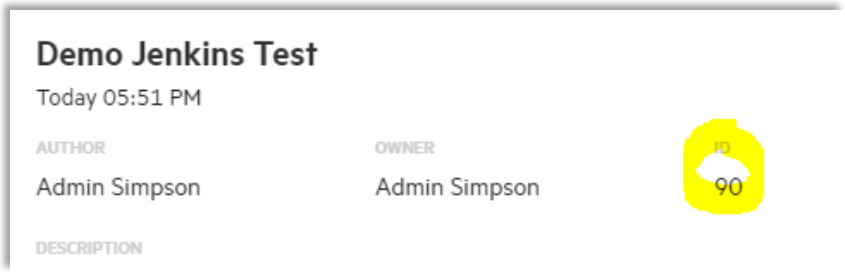
Persona: Will Nars, **Testing Guru**
24y of experience as a Guru
Works at WillNars.com
Lives in Morrilton, A, USA

Preparations / Prerequisites

- Jenkins installed
- HPE Application Automation Tools plugin (ver 5.3+) installed

Customer Scenario

1. For SRF we expanded the existing HPE Application Automation Tools Plugin by adding 2 new actions: "Execute test by SRF" and "Publish HP SRF tests result"
2. If you need help installing Jenkins please let us know or refer to Download Jenkins
3. In Jenkins go to Manage Jenkins > Configure System
4. Scroll down to SRF Common Setting
5. In the SRF server press the Add Server and enter: https://ftaas.saas.hpe.com:443
6. In case you are using proxy enter your proxy server
7. Enter your SRF Client Id and secret
8. Press Save
9. Make sure you have a test in SRF configured
10. You can find out the test ID by pressing on it and in the right side look at its ID:



11. In Jenkins create a new job by pressing the "New Item" in the left side of Jenkins
12. Give the item a name and select a Freestyle project and press OK
13. Go to the build tab and add a new build step and select "Execute test by SRF"
14. In the SRF Test Id enter the test ID from step #10
15. Press Save
16. Press Build Now
17. You will now see the test running in SRF

Functional Testing | StormRunner Functional | Hands-On Session #2 – Advanced

Use Case #5 – Define your continuous testing with Jenkins & SRF

Persona: Moe Yolantoino, **Testing Guru**
24y of experience as a Guru
Works at moeyolantonio.com
Lives in Morrilton, A, USA

Preparations / Prerequisites

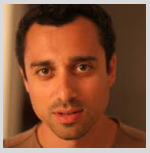
- Jenkins installed
- Plugin installed
- 2 tests defined in SRF

Customer Scenario

1. In SRF, define 2 tests that you want to run as part of your CT process.
2. Assign each test with the “sanity” tag
3. We will now use the tag to tell Jenkins to run the tests that are tagged with this tag.
4. In Jenkins create a new job by pressing the “New Item” in the left side of Jenkins
5. Give the item a name and select a Freestyle project and press OK
6. Go to the build tab and add a new build step and select “Execute test by SRF”
7. In the SRF Test tags enter the tag from the previous step
8. You can also pass Jenkins the Build and release number. It is best to use Jenkins runtime variable or any parameter that is part of the pipeline.
9. Last option will be to send a parameter as part of the Jenkins job.
10. Simply press on Add Parameter
11. And fill in the key and value of your defined parameters.
12. Now, add a post –build action so you can see you SRF results in Jenkins
13. Add a new Post-Build action
14. Select “Publish HPE SRF Tests Results”
15. Press Save
16. Press Build Now
17. While running you can examine the progress of you tests in the Jenkins console
18. Press on the build number in Jenkins
19. Press Console Output
20. Examine the progress
21. Once the Jenkins job is finished you will be able to see the results in Jenkins

Use Case #6 – Remote Execution Selenium Java

Persona: Dani Hovav, **Software Engineer in Test**
8.5y of experience as a SEIT
Works at Bosem
Lives in Holon, Israel



Preparations / Prerequisites

- Selenium Script ready and running locally (**SeleniumRemote.java** under https://github.com/Rishon73/Enablements/tree/master/2018-03-bootcamp/SRF/scripts)
- SRF Client ID & Secret

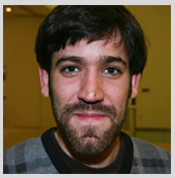
Customer Scenario

1. Open your IDE of your choice
2. Import the SeleniumRemote.java file into your existing Selenium project
3. The script is already configured with all the needed capabilities to run your script
4. In each line make sure you fill in the missing items:
5. Line 28 – your script will run on Chrome. If you want to change it you can change the DesiredCapabilities.XXX() to the browser you want
6. Line 30 – fill in your SRF Client ID
7. Line 31 – fill in your SRF client secret
8. Line 32 – fill in your SRF results name. Highly recommended to give this a unique name as this will be the name of the results in SRF UI
9. Line 34 – set the browser version that you want. Note: this is not mandatory (SRF will give you the latest in case not specified)
10. Line 35 – the operating system. Note: this is not mandatory. Default will be Windows 7

    Run the test and go to SRF to see the results

Use Case #7 – Compiling and Uploading a Jar to SRF

Persona: Raul Powell, **Testing Consultant**
11y of experience in consulting
Works at raulp.es
Lives in Reus, Spain



Preparations / Prerequisites

- Selenium script ready and running remotely against SRF
- Maven installed
- Descriptor file

Customer Scenario

1. Make sure that your project is a maven project.
2. It is recommended that you copy & paste the original remote execution file in the same location for reference
3. In your project, add the descriptor.json file under the resources folder. Where the runnableClass is the path the name of the class that you want to run:
4.
```
[
  {
    "runnableClass": "com.hpe.srf.SeleniumUpload",
    "framework": "junit"
  }
]
```

In your Selenium script

5. Replace your Client ID and Secret with the following:

```
capabilities.setCapabilit("SRF_CLIENT_ID", System.getenv("SRF_CLIENT_ID"));
capabilities.setCapability("SRF_CLIENT_SECRET", System.getenv("SRF_CLIENT_SECRET"));
```

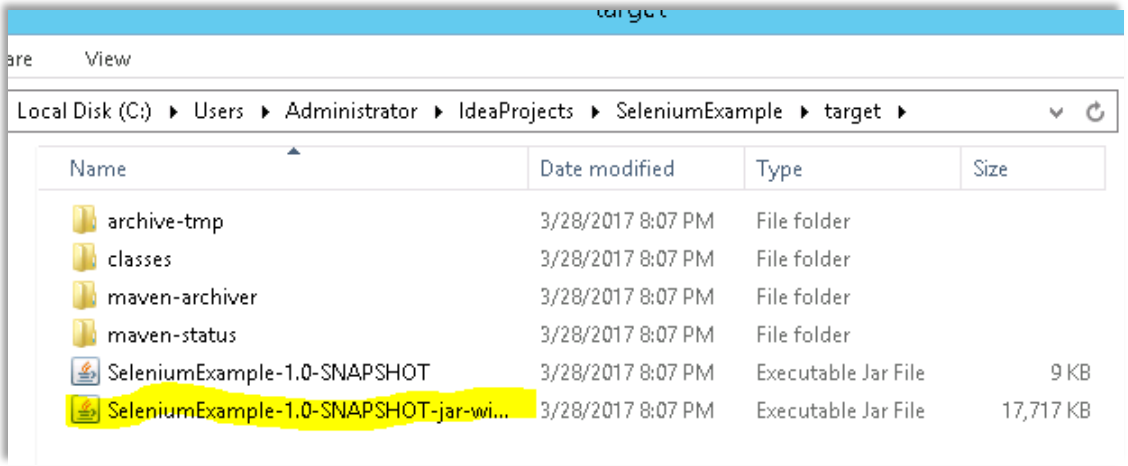
And the RemoteWebDriver location:

```
driver = new RemoteWebDriver(new URL(System.getenv("SELENIUM_ADDRESS")), capabilities);
```

6. Compile your code including dependencies and make sure that the hierarchy remains as is
7. The best way will be to use the maven plugin that complies the jar with dependencies:
8.
```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-assembly-plugin</artifactId>
    <version>2.6</version>
    <configuration>
        <descriptorRefs>
            <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
    </configuration>
    <executions>
        <execution>
            <id>make-assembly</id>
            <phase>compile</phase>
            <goals>
                <goal>single</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Note: there are several plugins that allows you to compile a jar with dependencies. The example above is just one.

9. You should run a maven clean install command (either from your IDE or from a command line)
10. After compiling you should find the file under your /target folder:



11. Go to SRF assets and press Upload
12. In the dropdown select Selenium and select the generated .jar file
13. Go to the automation and create a new test
14. Add the uploaded jar as the script and select and environment
15. Run the test
16. Check the results