ITAC

(Interactive Test Automation cartridge)

Technical Spec v1.0

|  |  |
| --- | --- |
| **Authors** | Abhishek Seelamanthula  Madhukar Gangadhar |
| **Contributors** | Praneeth Molakala  Rahul Babu Challa  Siri Cheemaladinne |
| **Interactive COP Reviewers** | Javeed Shaik  Bhawana Jha  Praneeth Molakala |
| **Asset Approvers** | Srivathsan Ramanujam  Sonal Naik |

**Contents:**

1. **Introduction of ITAC…………………………………………………………………… 3**
2. **Installation & Application usage …….………………….…………………………... 3**
3. **Back end Configuration……………………………………………………………….….. 4**
4. **Additional Comments……………………………………………………………………. 11**
5. Introduction:

ITAC application is designed for the automation of Mobile & Web browser applications encompassing below validations:

* 1. Finding the broken links throughout the site
  2. Fetching all the digital assets from the complete site and checking for special characters restricted in asset name if any & it also provides the asset sizes.
  3. It can also compare the assets between 2 environments and shows the missing assets.
  4. Getting the response time for a webpage.
  5. Compare the pages of 2 environments to validate the UI.
  6. Regression suite for project specific functionality checks.

1. Installation & Application usage of ITAC:

For the installation & usage of the ITAC please refer to ITAC-EndUserGuide attached below:



Also, refer to following documents for Emulator & Real Device set up in Windows & MAC:

If user wants to use the application without any code tweaks, then download the zip file from End Application folder in source code available in Inner Source.

**For Reference:**

Tasks List:

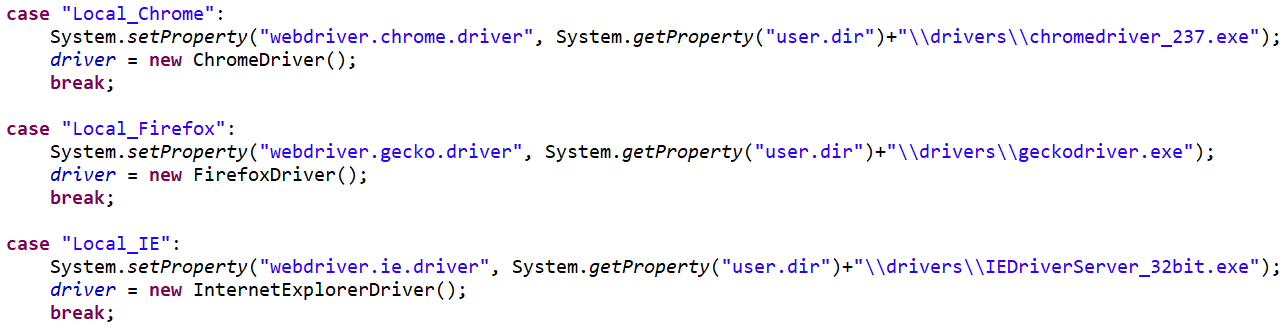
1. **Broken Links Task – BL**
2. **Special Characters – SC**
3. **Missing Assets Task – MA**
4. **Display Asset Sizes – DA**
5. **Page Comparison – PC**
6. **Response Time – RT**
7. Back End Configuration:

This is a maven-based project and build using TestNG framework. After importing the project into the workspace, the folder structure of the project goes as below:

1. All tasks related class files are placed under a package named **site\_monitoring** (\src\test\java\site\_monitoring).
2. Drivers are placed in driver’s folder in main folder.
3. Input Data folder stores the Test input data.
4. Results folder contains HTML results files & CSV files.
5. Screenshots folder has the screenshots of Page Comparison tasks.
6. testConfig.properties file which stores the action values that are performed on the application.
7. sampleTemplate.xlsx, is the sheet which will download when user click on download template button on the application.

**Task wise configurations follows:**

1. For Broken Links task BrokenLinks.java (**src\test\java\site\_monitoring\BrokenLinks.java**) class, will be executed and the methods in it follows:
   1. In the test annotation depending on the test input the execution starts.
   2. If the domain is given as test data then, given URL will be crawled completely throughout the site using Jsoup library in BrokenLinks\_ChildPages method in class file & then all the child links are stored in a list which is used further to get the status for each using *brokenLinks\_domain* method.
   3. If the input is given through excel with a list of URLs’, then all the links in each URL will be collected using finalLinksList method and then this list of links will be used for fetching the status.
   4. Global variables are used in most of the cases as, these needs to be stored the values for the HTML reports, once the variables are sent to HTML\_Reports.*BrokenLinks* method in **HTML\_Reports(\src\test\java\com\library\HTML\_Reports.java)** class, then all the variables are cleared at once.
2. For Special Characters, FindingSpecialCharacters.java (**src\test\java\site\_monitoring\** FindingSpecialCharacters**.java**), will be executed and the methods in it follows:
   1. In the test annotation depending on the test input the execution starts.
   2. If the domain is given as test data, then given URL will be crawled completely throughout the site using Jsoup library in GetAssetsList method in **WebCrawler** (**src\test\java\com\library\WebCrawler.java**) class file & then all the child links are stored in a list which is used further to get the asset list for each URL using getAllLinks method.
   3. If the input is given through excel with a list of URLs’, then all the assets in each URL will be collected using getAllLinks method.
   4. Then all the asset names will be verified with the given special characters in **validation** method.
   5. Global variables are cleared once it is sent to HTML\_Reports. *SpecialCharacters\_*HTMLReport method in **HTML\_Reports** class.
3. For MissingAssets FindingMissingAssets.java (**src\test\java\site\_monitoring\ FindingMissingAssets.java**), will be executed and the methods in it follows:
   1. In the test annotation depending on the test input the execution starts.
   2. If the domain is given as test data then, given URL will be crawled completely throughout the site using Jsoup library in GetAssetsList method in **WebCrawler** (**src\test\java\com\library\WebCrawler.java**) class file & then all the child links are stored in a list which is used further to get the asset list for each URL using getAllLinks method. This will be run on both environments Prod & test.
   3. If the input is given through excel with a list of URLs’, then all the assets in each URL will be collected using getAllLinks method. This will be run on both environments Prod & test.
   4. Assets in each URL are compared, and missing assets will be stored in array list.
   5. Global variables are cleared once it is sent to HTML\_Reports. *MissingAssets\_HTMLReport* method in **HTML\_Reports** class.
4. For Asset Sizes DisplayAssetSize.java (**src\test\java\site\_monitoring\ DisplayAssetSize.java**), will be executed and the methods in it follows:
   1. In the test annotation depending on the test input the execution starts.
   2. If the domain is given as test data then, given URL will be crawled completely throughout the site using Jsoup library in GetAssetsList method in **WebCrawler** (**src\test\java\com\library\WebCrawler.java**) class file & then all the child links are stored in a list which is used further to get the asset list for each URL using getAllLinks method.
   3. If the input is given through excel with a list of URLs’, then all the assets in each URL will be collected using getAllLinks method.
   4. Assets list is fetched with the complete path, so that they are executed for fetching the content size.
   5. Global variables are cleared once it is sent to HTML\_Reports. *AssetSizes\_HTMLReport* method in **HTML\_Reports** class.
5. For Page Comparison PageComparison.java (**src\test\java\site\_monitoring\** PageComparison**.java**), will be executed and the methods in it follows:
   1. In page comparison, depending on the browsers list, the runnable **Comparison.java** class will trigger to achieve parallel testing. For example, if 3 browsers are selected and then in the 3 threads runnable class will be executed parallelly.
   2. Using Jsoup, URL will be opened, and HTML will be parsed through it and fetch the class names using GetClassnames method of all the sections.
   3. Using class names will get the CSS selectors with GetSelectors method and copied into list which will then use for capturing screenshots of each section using CaptureScreenShot method.
   4. This will be executed for both Test & Prod, also, if the selectors list of Prod & Test is matched, only then the screenshot will be captured.
   5. As it only consists of div & sections tags, so it may require few tweaks if the site HTML structure varies.
   6. Global variables are cleared once it is sent to HTML\_Reports. *PageComparison\_HTMLReport* method in **HTML\_Reports** class.
6. For Response time, ResponseTime.java (**src\test\java\site\_monitoring\** ResponseTime**.java**), will be executed and the methods in it follows:
   1. URLs will be opened in the browser and capture the time to load the page.
   2. This time will be compared to the threshold value and segregate the pass & failures accordingly.
   3. Global variables are cleared.
7. In Regression pack SampleTest.java(\src\test\java\global\_suite\SampleTest.java) is a sample script & these can be replaced or can be added with any functionality checks script.
8. For the tasks, **PC, RT & RS** drivers and devices are involved and its configuration is as below.
9. GetDriver.java(\src\main\java\utilities\GetDriver.java) script is used to launch the browser wherever needed in all the above tasks and methods in it as below:
   1. Browsers list is made with all the capabilities set and this can be launched accordingly with user input, ex: if input is **Local\_Chrome**, then chrome is launched as below:



* 1. If Browserstack must launch, need to have **Browserstacklocal.exe (\drivers\BrowserStackLocal.exe)** file and using the command prompt, file will be launched with user credentials with the command:

BrowserStackLocal.exe --key "+ User AUTOMATE\_KEY +" --local-identifier Test123.

* 1. Once the server is up, we can connect to the browsers and devices depending on the user input. Capabilities will be set and using **RemoteWebDriver(**[**URL**](eclipse-javadoc:%E2%98%82=CyberClone/C:%5C/Users%5C/s.abhishek%5C/.m2%5C/repository%5C/org%5C/seleniumhq%5C/selenium%5C/selenium-remote-driver%5C/3.14.0%5C/selenium-remote-driver-3.14.0.jar%3Corg.openqa.selenium.remote(RemoteWebDriver.class%E2%98%83RemoteWebDriver~RemoteWebDriver~Ljava.net.URL;~Lorg.openqa.selenium.Capabilities;%E2%98%82java.net.URL) **remoteAddress,** [**Capabilities**](eclipse-javadoc:%E2%98%82=CyberClone/C:%5C/Users%5C/s.abhishek%5C/.m2%5C/repository%5C/org%5C/seleniumhq%5C/selenium%5C/selenium-remote-driver%5C/3.14.0%5C/selenium-remote-driver-3.14.0.jar%3Corg.openqa.selenium.remote(RemoteWebDriver.class%E2%98%83RemoteWebDriver~RemoteWebDriver~Ljava.net.URL;~Lorg.openqa.selenium.Capabilities;%E2%98%82org.openqa.selenium.Capabilities) **capabilities)** launch the browser using **browserstack** method in **GetDriver.java** class.

**URL** used in RemoteWebDriver:

https://" + USERNAME + ":" + AUTOMATE\_KEY + "@hub-cloud.browserstack.com/wd/hub

Username & Automate Ker refers to user input of Browserstack credentials.

**Capabilities** will be used as below:



All the above capabilities for each browser can be found on the Browserstack website and need to set up them here in our application. **User should have the account with Browserstack in order to execute the scripts.**

1. For Real devices & emulator refer toRealDeviceDriver.java (\src\main\java\utilities\ RealDeviceDriver.java).

Capabilities are provided same as the other browsers, but then here the device, platform & app names are included.

1. After the completion of the execution, **HTML\_Reports.java (\src\test\java\com\library\HTML\_Reports.java),** class is used to generate the reports for each task. It contains each method for each task. If any regression suite is added to existing, then for HTML report, we can reuse the existing task report method by making a clone of it and customizing it as needed.

**GUI Application Build:**

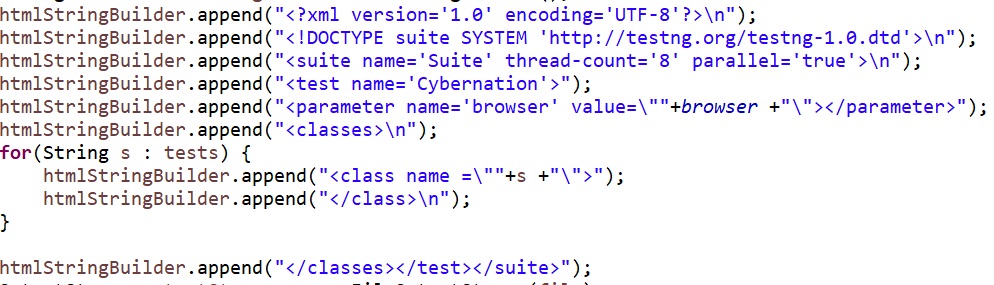
All the above class files are embedded & shown on the GUI & its configuration follows:

1. GUI application developed using Jswing builder library. Jswing window builder plugin needs to be install into eclipse from Market place (In eclipse Toolbar>Help>Eclipse Marketplace).
2. For every component on the application has its own configuration, for example:



In above, check box, Title, Radio buttons and its text are configured individually with required paddings.

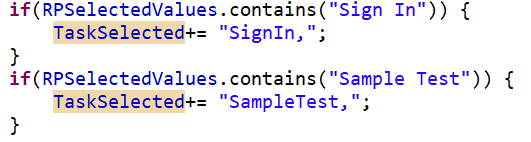
1. This configuration can be edited in 2 modes i.e., Source & Design. In source, code is editable & in design view the components will be displayed for editing.
2. In design view, user can drag & drop the required components on to the configuration pane & then we can write the action listeners for each component.
3. All the GUI application related code is written in WATT.java class file which is placed under **(\src\test\java\global\_suite\ITAC.java).**
4. Action listeners for all the fields on the application are placed in above class.
5. CheckCombobox1.java is used to create combo box with given elements.
6. Browsers list are maintained in a list which is sent to CheckCombobox1.java method to form a combobox dropdown.
7. Results2.java file is used for the results pop-up after completion of execution which contains the HTML path in it.
8. AddionalDetails.java is used for th additional details pane in which reference pointers are metioned.
9. Inputs are stored into a config file for each action performed on the application.
10. Once the execution button is clicked, all these inputs will be written into a config file (Main project folder-->**testConfig.properties**) & is placed in project level folder.
11. XmlGenerate.java **(\src\test\java\global\_suite\** **XmlGenerate.java)** class file contains the code for reading of the config file.
12. Using the switch case, the selected tasks on the application will be added to a list which is used to generate the dynamic XML by adding each task as a class using TriggerTestNGxml method where in all the content is prefilled and need to just add the class names into it as below:



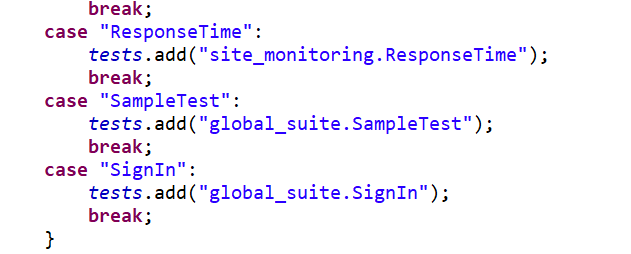
1. XML file (Main project folder-->**testngRunner.xml**) is executed using testing runner method.

Each tasks of the application have each class file which will be triggered independently.

1. In regression suite we can add more into the list just by adding the name into regression\_packList array list in WATT.class and add it to the **TaskSelected** string in action listeners of start execution button:



1. Regression suite which are added needs to place in GUI & add the action listeners for each class, which in turn writes in config file. Also, in XML generate class, newly added regression class needs to be updated as follow:



**Utility class files:**

1. **CreatingCSV.java (\src\main\java\utilities\CreatingCSV.java)** is used to convert the multiple excel sheets to CSV files, as the excel sheet may get corrupted during the execution. User input excel sheets will be converted to CSV files and then these are used in execution.
2. **CSV\_Report.java (\src\test\java\com\library\CSV\_Report.java)** is used to write the results to CSV. Usually HTML reports can give this data but can’t be copied quickly for any further works so CSV files.
3. **AdditionalDetails.java(\src\test\java\global\_suite\AdditionalDetails.java)** is used for the additional panel section in application, all the details of this pane are configured here.
4. **Logger.java (\src\test\java\com\library\Logger.java)** is used for the write the logs.
5. Additional Comments:
   1. Make sure all the excel files are saved and close before you download & upload the template into application
   2. Make sure all the inputs are provided before execution starts.
   3. User can’t manipulate anything on application during the execution.
   4. Do not rename the file names in the folder and don’t change the path of any file.
   5. Do not delete pre-existing files in any of the folders.