**Developer Guide**

**Gauge Automation**

**Index**

[Developing Gauge Automation Application](#_egbmyiu09f9t) **3**

[Overview of Gauge Automation application](#_hks91dgtb9s5) 3

[Skills needed to develop gauge advantage automation](#_nj6jx2n5j305) 3

[System-level configurations](#_fofk9na1l00j) 3

[Set Display scale to 100%](#_li1cgdq0a103) 3

[IE11 Settings for Protected mode](#_o3shsojp3gx6) 4

[Set Registry](#_khjh032wi6ed) 6

[Directory Structure](#_ssag0tni61t2) **7**

[Overview](#_hdsjn96o195c) 8

[Working with Gauge Automation application](#_qt5tj4klej9s) **10**

[Gauge UI](#_x4gqq428su8v) 10

[Change UI server Port](#_2tgbz25dk67f) 10

[Details of User and Database](#_aprjfz5ag6xd) 10

[Test case selection](#_6d0y25fv23v9) 11

[Add New Test Case](#_48nq3cmrhuq2) 13

[Writing specification(spec) file](#_mmyvimerxxc7) 15

## 

## Developing Gauge Automation Application

### Overview of Gauge Automation application

Gauge is a free and open-source framework for writing and running acceptance tests.

The Syntax for Gauge tests are based on Markdown which makes writing and maintaining tests easier. Users can reuse specifications and robust re-factoring to reduce duplication. Less code and readable specifications mean less time spent on maintaining the test suite.

### Skills needed to develop gauge advantage automation

The following are the skills and experience that is required and useful in developing Gauge Automation applications:

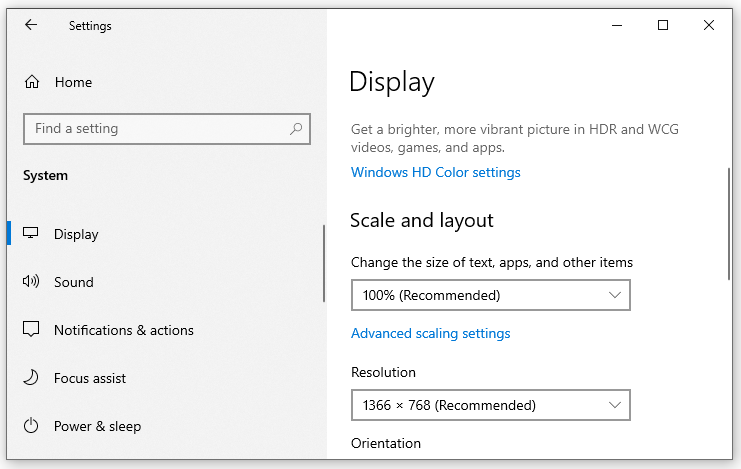
* + Web development skills (HTML, CSS, JavaScript, HTTP, cross-browser issues, etc.)
  + Overall understanding and detailed knowledge of:
    - Gauge
    - Webdriver.io
    - Selenium Standalone
  + NodeJs
    - Handling of the server-side processing of parameter’s JSON, execute the script to trigger Gauge Automation and SSH server bouncing.
  + Oracle - Knowledge about Database connectivity and executing queries.

### System-level configurations

Apply the following mandatory configurations in order to test an application using Gauge Automation in IE11.

#### Set Display scale to 100%

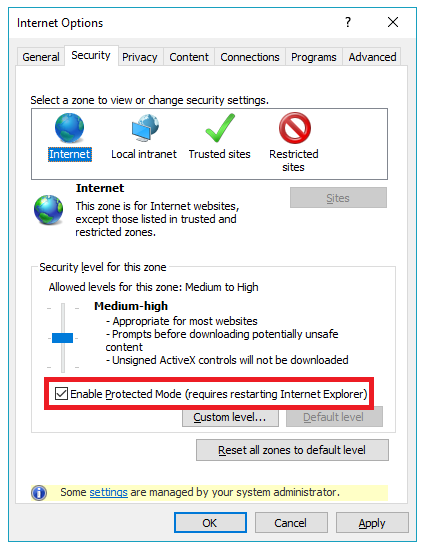
* + Go to Change display settings under the System Settings.
  + Select a 100% scale for “Change the size of text, apps, and other items” under “Scale and layout section”



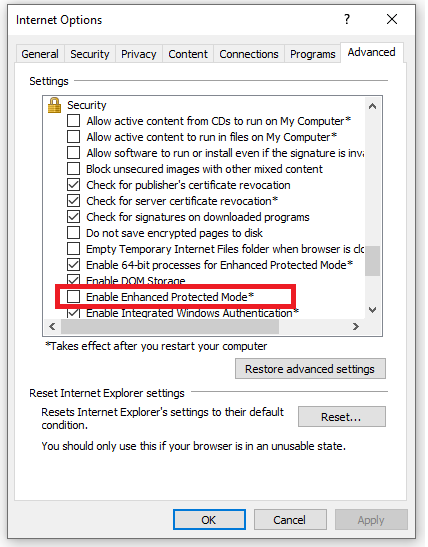
#### 

#### IE11 Settings for Protected mode

* + **Protected mode setting - Step 1**
    - Open “Internet Options” from settings of Internet Explorer.
    - Click on the “Security” tab.
    - Tick on the checkbox "Enable Protected Mode" for all the zones (Internet, Local internet, Trusted sites, Restricted sites).
    - Click on “Apply”.



* + **Protected mode setting - Step 2**
    - Open “Internet Options” from settings of Internet Explorer.
    - Click on the “Advanced” tab
    - Scroll down to the “Security” section and perform these steps:
      * Check “Enable 64-bit processes for Enhanced Protected Mode\*”
      * Check “Enable DOM Storage”
      * Uncheck “Enable Enhanced Protected Mode \*”
    - Click on “Apply”.



#### Set Registry

* + - Open the Registry editor by running "regedit" in the Run program.
    - Navigate to:

For 32-bit Windows installations,

HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\

InternetExplorer\Main\FeatureControl

For 64-bit Windows installations,

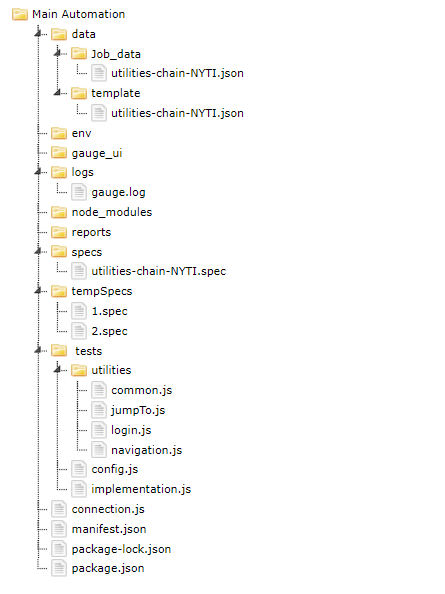
HKEY\_LOCAL\_MACHINE\SOFTWARE\Wow6432Node\

Microsoft\Internet Explorer\Main\FeatureControl

* + - Create key, FEATURE\_BFCACHE, if not already present.
    - Inside this key, create a DWORD value named “iexplore.exe” with a value of “0”. Even if QWORD is suggested for 64-bit machines, create a DWORD.

## Directory Structure

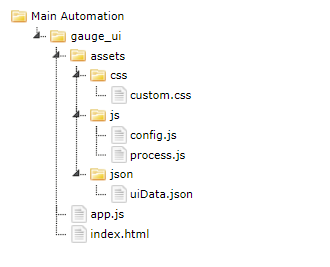
The directory structure of the Gauge automation project is as below:



### 

### Overview

* data
  + The data folder contains subfolder and files as below:
    - **job\_data:** Folder contains job data in json format.
    - **Template:** Folder contains files in the json format of schema and default job data.
      * Schema from the template file is used to manipulate json in the JSON editor plugin.
      * The template file will never modify from UI and act as a read-only file.
* env
  + The env directory contains multiple environment-specific directories. Each directory defines the environment variables set during spec execution for that specific environment.
* gauge\_ui
  + All the files which are used for UI, are stored in this directory.



* + assets :
    - CSS: This folder contains CSS files for the attractive and responsive UI.
    - Js: contains js files used for UI.
      * **process.js:** File contains a list of all the processes to list out on UI.
      * **config.js:** File contains setups for the port, spec files, and json files.
    - json: contains json files used for UI.
      * **uiData.json:** File contains user data and database details filled by the user from UI.
  + App.js
    - app.js file contains all the node js APIs, which is used in gauge automation application.
  + Index.html
    - How the UI looks like is displayed in this file.
* logs
  + Logs of Gauge automation will be kept in the gauge.log file under this directory.
* node\_modules
  + Dependencies related to node.js and Gauge will be stored in this directory.
* reports
  + Under the HTML-report folder, the index.html file will display the report of all test cases, which were executed and this report will display the number of specs and scenarios that are passed or failed.
* specs
  + All the processes’ spec files will be stored in this directory.
* tempSpecs
  + Copy of the spec files that were executed is stored in this directory with name as 1.spec, 2.spec.
* tests
  + The tests folder contains the test code including step implementations.
    - **utilities:** It contains common.js, jumpTo.js, login.js, navigation.js.
    - **config.js:** config.js contains app URL, app username, app password, browser name, etc.
    - **implementation.js:** It contains steps which are implemented in spec files
* connection.js
  + connection.js file is the file that creates a connection between application and database.
* manifest.json
  + It contains information such as the language used and plugins required for the Gauge project.
* package.json
  + This file is used to send the information to npm that allows it to identify the project as well as to handle the project's dependencies.
* package-lock.json
  + This file is automatically generated for any operations where npm modifies either the node\_modules tree or package.json.

## Working with Gauge Automation application

### Gauge UI

* Run the “startServer.bat” file in order to run the node server and start the selenium-standalone server.
* The Node server and selenium standalone server are running on port 8080 and 4444 respectively.
* Files for Gauge UI are kept under the gauge\_ui directory.

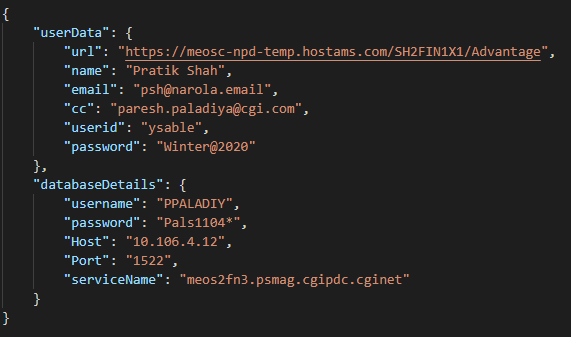
#### Change UI server Port

* Port of node.js server can be changed from **config.js** file under the gauge\_ui directory.
* config.js file contains a json of UI configuration as below:

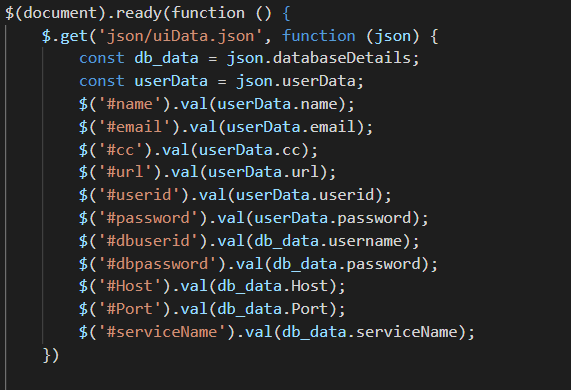


#### Details of User and Database

* User data and database details will be stored in “uiData.json” file in json format as below:

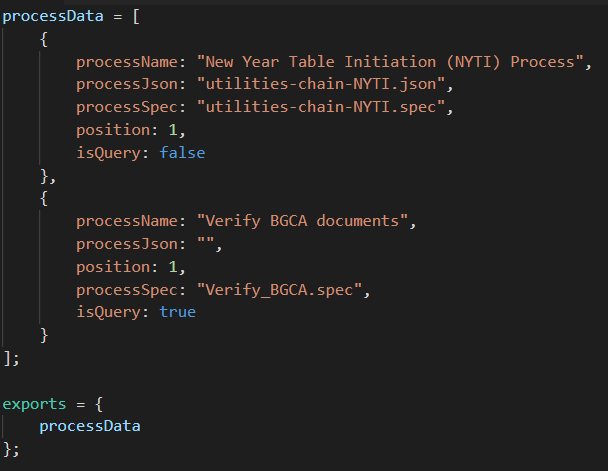


* On UI, details for User and Database are obtained using Ajax call as shown below from the index.html file.

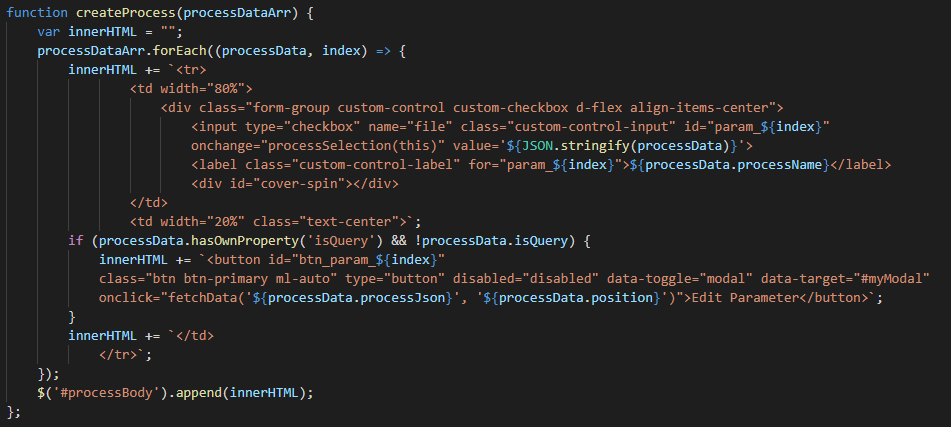


#### Test case selection

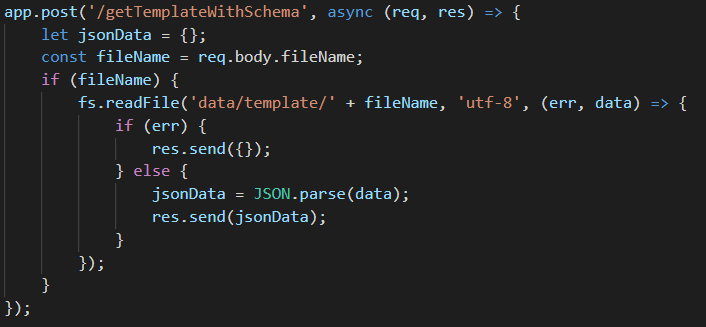
* All the test cases are stored in “process.js” file in an array of JSONs format as below:



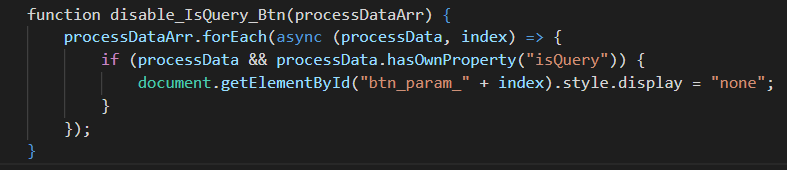
* A list of test case selection with a checkbox has been created on UI using the following function after fetching all the test cases from process.js.



* When a user edits the parameters of a specific process and saves that changes, It will fetch only data from the json file of the template folder and update data in a json file in its given position. if data is not given for that particular position, then it stores a null value.



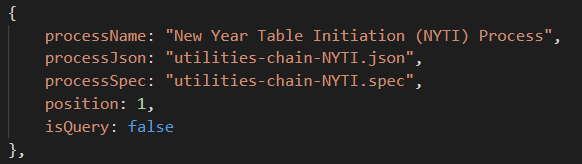
* In some processes, the “Edit Parameter” button will not be displayed because those processes don’t need to fetch job data, for that use disable\_IsQuery\_Btn function.



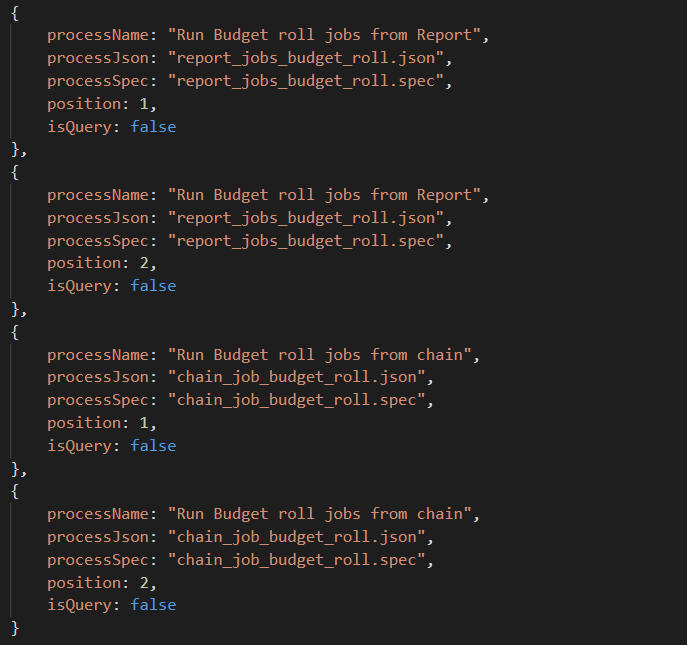
* When submitting selected test cases, It fetches all test cases spec files and json data.
* Json data will be stored in its given position. If need to edit the parameters of the test case, then it stores the updated data else it will store the default data.
* Thereafter, copies of the Spec files of selected test cases are created and stored in the “tempSpec” folder. Now, execute the gauge run command to run the test cases.
* When the “gauge run” command is run, first run login spec as a default login is set. Now, run the other spec files that are fetched from the “tempSpec” folder.

#### Add New Test Case

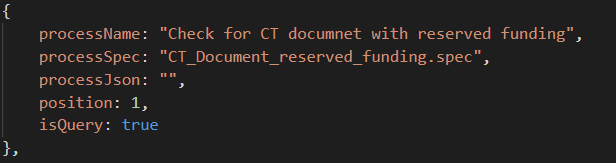
* To add a new test case, add details of the test case in the process.js file.
* process.js file contains an Array of jsons containing detail process as below:



* **JSON key specification:**
  + **processName:** Name of the process
  + **processJson:** Name of json file to fetch job-specific template data or store job data.
  + **processSpec:** Name of the spec file associated with the process.
  + **position :** 
    - Indicates the position in which the test case will be executed.
    - The position will be set for groups of the same test cases. So, the position will be set to “1” for the first test case of each group.
  + **isQuery:**
    - It is used for database query if the test case contains only database query then isQuery value is set as “true”; otherwise “false”.



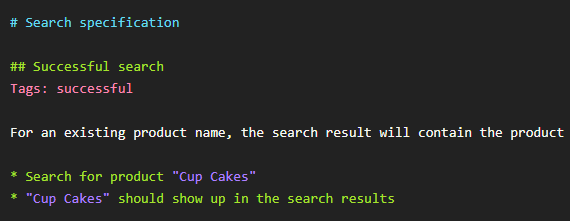
* Some test cases perform database query only. For such a test case, json file is not required to fetch / store job data. To add this kind of test case, add details in the following format.



* In addition to normal test case JSON, add JSON key “isQuery” and set the value to “true”.

### Writing specification(spec) file

* A specification is a business test case that describes a particular feature of the application that needs testing.
* Gauge specifications support a .spec or .md file format and these specifications are written in a syntax similar to Markdown.
* When a Gauge project is created and initialized, a specs directory is automatically created at <project\_root> with a sample file, example.spec.



* A specification consists of different sections, some of which are mandatory and some are optional. The components of a specification are listed as follows.
  + **Specification heading**
    - A specification must begin with a “specification heading”. A specification must contain only one specification heading.
    - In the given example, “# Search specification” is a specification heading.
  + **Scenario**
    - Each scenario represents a single workflow in a particular specification. A specification must contain at least one scenario.
    - A scenario starts after a scenario heading or a scenario name.
    - In given example “##Successful search” is the scenario heading.
    - A scenario contains one or more steps in it.
  + **Step**
    - Steps are the executable components of a specification that are written by using the Markdown unordered list syntax.
    - In the given example, “Search for product ‘Cup Cakes’” is a step.
  + **Parameters**
    - Steps are defined to take values as parameters so that they can be reused with different parameter values.
    - In the given example, “Cup Cakes” is a parameter value.
  + **Tags**
    - Tags are used to associate labels with specifications or scenarios. Tags help in searching or filtering specs or scenarios.
    - Tags are written as comma-separated values in the specification with a prefix “Tags:” as shown in the above screen.
    - Both scenarios and specifications can be separately tagged. However, only one set of tags can be added to a single specification or scenario. It is an optional component.
  + **Comments**
    - Any sentence in plain text which does not follow any syntax is seen as a comment in a spec. Comments help enhance the readability of a spec without being executed. It is an optional component.
    - In the given example, “For an existing product name, the search result will contain the product” is a comment.
* **Step Implementation**
  + Steps have a language-specific implementation that gets executed on the spec execution as depicted below:

