Table of Contents

Parallel Execution

The Idea –

The execution of UI test cases takes a huge amount of time to execute. Hence, the time required to certify a build has increased significantly. If you consider the present situation, we have around 1400 test cases across all modules that takes about 30 hours to run!!! We have estimated that we could have around 4000 tests for our product. Still we are adding more features in masterworks. The number of tests is likely to shoot up. Don't be amazed if we take 3 days to certify a build. The whole point of automation is lost in the bargain.

That is why we have to move to a position where we can simultaneously test different areas of the product and reduce this huge lag for QA certification.

The Flow

API

We will have an end point where we will request for an automation run. The end point will delegate the job to the manager.

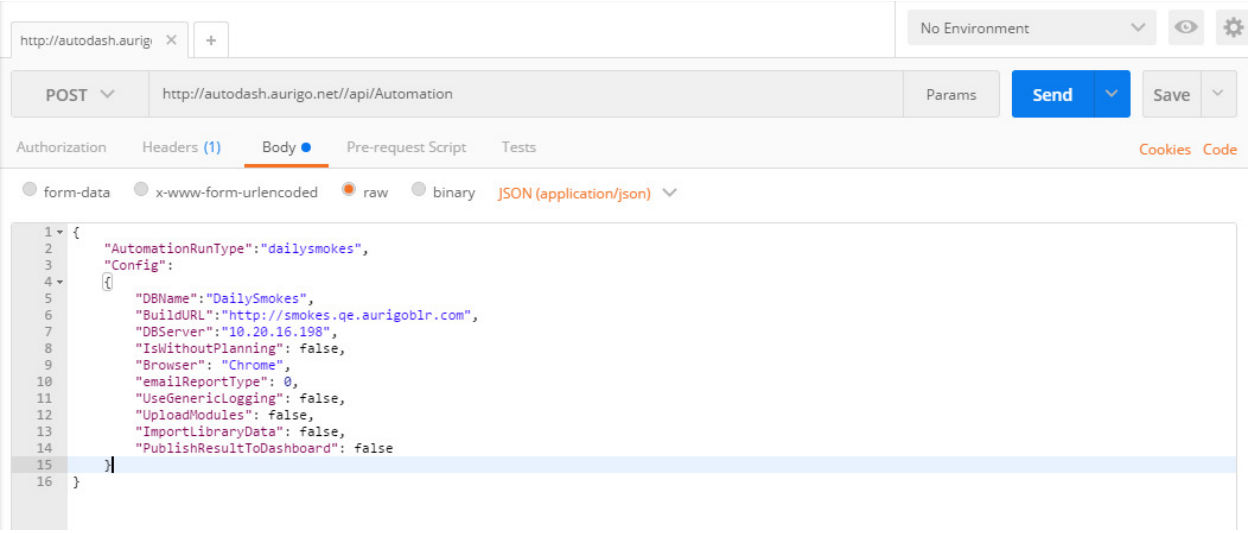
Resource URL is - http://autodash.aurigo.net//api/Automation

Method – POST

Request Body –

```
{
        "AutomationRunType":"dailysmokes",
        "Config":
        {
                "DBName":"DailySmokes",
                "BuildURL":"http://smokes.qe.aurigoblr.com",
                "DBServer":"10.20.16.198",
                 "IsWithoutPlanning": false,
                 "Browser": "Chrome",
                "emailReportType": 0,
                "UseGenericLogging": false,
                 "UploadModules": false,
                 "ImportLibraryData": false,
                 "PublishResultToDashboard": false
        }
}
```

All the properties of the Config(Configurations.cs file in MWProduct) class can be modified using the Config Key of the request body. Just pass the correct value. It should be true or false for Boolean values, integer denoting the enum value for enums and strings in “” for strings. Here is a sample request using Postman –



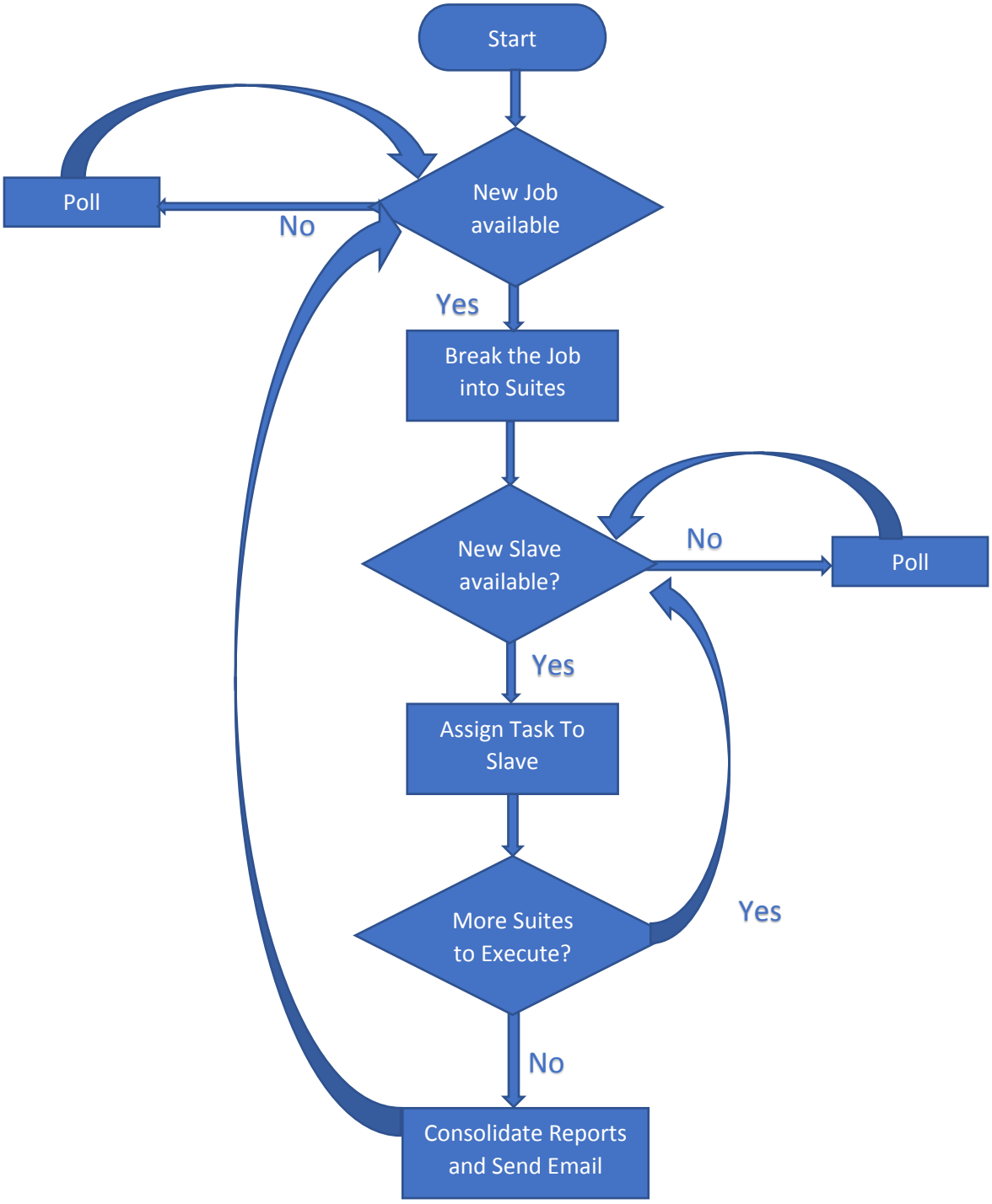
Manager

The manager has the following main tasks –

1. Get the requirements for the run(Configurations).
2. Break the automation task into Suites (same as in ProductAreas.xml file)
3. Pick up a machine which is available and assign it one suite of tests to execute. (this process will be repeated until all the areas are assigned)
4. Consolidate the reports from all the slaves for the Job once all the modules have finished execution.

Following flowchart explains the role of manage in detail.

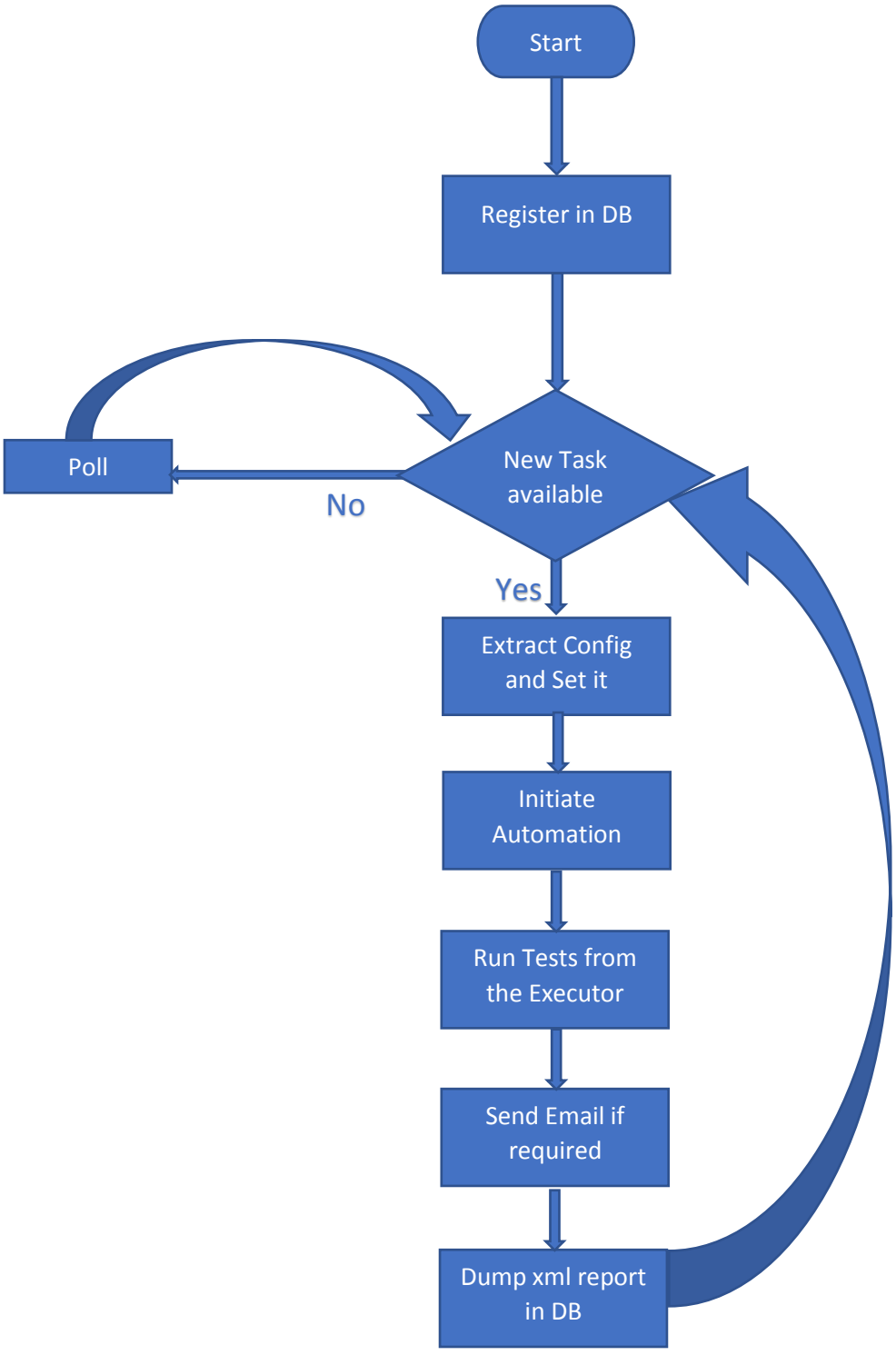
Flow Chart For Manager –



AutomationManager class represents the manager(in AutomationStart assembly.)

Slave

There could be multiple slaves each with a specific Id. Whenever you start a slave, it will register itself with the database and put an entry in the Slaves table. Each slave will keep polling the database and whenever it has a task assigned, it will execute the suite of test cases according the configurations specified by the manager. After the suite is completed, the reports in xml form are dumped in the database. The following flowchart explains the slave flow.



AutomationSlave.cs file in AutomationStart has the code for the slave machines. Class name is AutomationSlave.

Changes done in the framework –

Moving Configuration out of GlobalVariables to a separate class

This is the most important and the biggest change done in Optimus. Since all the configurations of the automation were stored here in static fields, it was very difficult to use the existing structure to run automation on different builds at a single go. Perhaps the only choice was to stop the application, modify the xml files that had configurations and execute again or modify all the fields at runtime.

More elegant approach was to make all the configurations instance variables and have one object of the config for each specific run. So all the configurations are now moved to CurrentConfig.cs file. It is a singleton class to ensure that we have only one copy of the configurations per automation run. This config is initialized by the slave component which are sent to them by the manager. This enabled us to communicate to our framework seamlessly at the run time.

The Suite Structure

Somkes will now be considered a suite. So basically smokes is a suite of test cases and that will be defined in the product areas xml file. For executing smokes, we just have to make use of command line arguments and pass its respective Id.

```
<ProductAreas>
  <Suite id="dailysmokes" Type="Smokes" executeVal ="Yes">DailySmokes.xml</Suite>
  <Suite id="coremodsmokes" Type="Smokes" executeVal ="Yes">CoreModSmokes.xml</Suite>
  <Suite id="apidailysmokes" Type="Smokes" executeVal ="Yes">APIDailySmokes.xml</Suite>
  <Suite id="withoutplanningsmokes" Type="Smokes" executeVal ="Yes">WithoutPlanningSmokes.xml</Suite>
```

The type of the smokes suites will be “Smokes” and all these suites will be ignored during the full pass runs. The test settings file for the smokes are now moved to the TestSettings folder and the executor is also modified to account for this change in the execution path.

Changes for running smokes and full pass

The new App.config is now redesigned.

```
<appSettings>
  <add key="CurrentTestTeam" value="Product"/>
  <add key="RunAs" value="Manager"/>

  <add key="Path" value="\AutomationStart" />

  <add key="CreateScratchBuild" value="false" />
  <add key="IsWithOutPlanning" value="false" />

  <add key="Browser" value="Chrome" />
  <add key="EmailReport" value="Consolidated" />
  <add key="ModuleUploadThroughAPI" value="false" />
  <add key="UseGenericLogging" value="No" />
  <add key="ExecuteJMeterScriptsOnly" value="false" />

  <add key="ProductAreaPriorities" value="0,1,2" />
  <add key="TestCasePriorities" value="0,1,2" />

  <add key="LogPath" value="Logs/AutomationLogs.xml" />
</appSettings>
```

We don't have keys for all the different types of smokes run. Now every run will be interpreted as a collection of suites. You can choose a specific suite by passing the command line arguments or let the full pass run ignoring the smokes suites in the product areas config.

We have the option to specify whether the build is without-planning. CreateScratchBuild key will be used to create a scratch build and all the following automation runs will be done on the new build if that option is selected.

RunAs attribute will specify what type the current instance will run as i.e. Slave or Manager or the old usual single instance.

Current test team specifies what is the team running automation. It is used for marking ownership for smokes runs.

Entity Framework

We are using EF Database first approach as ORM for talking to the database.
AutomationSlaveDBContext is the context used for mapping.

Changing the reference of GlobalVariables in all the files in platform and test bed

We have changed all the occurrences of GlobalVariables class that were used to map to the configurations that were static members of the class. All of them are now made to use the configuration properties provided by the Config class.

TextLogger for Executor

We have implemented the text logger in Automation Utilities for logging information from the AutomationStart Assembly that is not related to the test cases.

All the logs will be done in Logs folder directly in the bin.

All the Loggers including extent reports are moved to MWPlatform\Loggers Folder.

Q & A's

Dependency

We have a product where there is a huge dependency across the modules. For instance, if you want to run contract tests, then you will have to create a project to be able to do that. How do we set up pre-requisites for running a specific module?

Solution –

We will be using the Dependency features in Optimus that will allow us to run only those tests that will be required to set up the project for testing a module. For instance we will run only create project test case of the planning module to test Bid Management.

We can also opt to set up the project via API calls but we don't have API's for all the forms in masterworks. But, this would be the best approach to do this job.

Reporting

How do we merge the reports of all the modules into one?

Solution

We have to establish communication between the slave machines and the manager. As soon as the slave finishes a task, it sends the report back to the manager. We will be using database for the purpose of communication. The slave will finish the task and will save the report in the database (in xml format). The manager will pick up all the reports for a specific automation run and merge all of them (in xml format).

How do we make a readable report from the xml logs?

We will be fetching all the test nodes from the xml file and for each node we will create a corresponding test using extent reports and log all the information for that test in the extent report.

Where is the database hosted?

The database is hosted on the aurvpc-buildsrv machine. The database name is AutomationDB.

How do we figure out the time required for each module to run?

The slave machine will update the corresponding entry in the AutomationTasks table specifying the time required for that Suite to complete.

How to find the time required for entire fullpass?

The manager will update the entry in the ManagerJobs table with the time required for the job till completion.