***Live Assist QA Automation Regression Test Infrastructure & Test Drivers***

***QA Mocha Basic Test Infrastructure and App Test Driver (Nodejs/Mocha):***

*Object*: Create a basic QA Mocha test infrastructure with following QA test drivers which can be used for QA Live Assist feature/functional automation tests (mostly on API & GUI) and regression test, also it provides basic coding/functions/drivers support for api+gui performance tests/load tests.

*Overview:*

1. QA Mocha regression test infrastructure for Live Assist which provides basic test framework, test functions (along with QA test drivers) and coding structure/blocks for new QA automation test cases creation, managing the case setup, run, result report, test logging, and basic exception handling, including: nodejs modules for testing, mocha test framework support, global QA test set/case configurations, config for before test run setup, config for after test run setup, logging, test result and assertion, etc.
2. QA App test driver for Live Assist RESTful APIs testing and RPC over HTTP APIs testing, which can be used for testing on: MRCP API, IVR api, web api, VXML APIs, session storage APIs, DEBUG (info) APIs, escalation classifier APIs, etc. It also can simulate commands and requests from others (IVR application or web app) and verify returned results/logs/responses.
3. QA web test driver for Live Assist agent GUI testing, with protractor lib + webdriverJS + selenium, which can be used for testing on: agent GUI HTML implementation and agent GUI AngularJS implementation. It also can simulate commands and requests from Agent and verify the returned results/logs/responses between the Agent Router Interface (part of Live Assist server) and Agent Server/Client (GUI, via browser).
4. QA MySQL DB test driver, which can be used for most Live Assist MySQL DB related testing and customization: application context and parameters configurations, escalation classifier settings, etc.

Web

IVRHTTPHTTP

QA App test driver/simulator



HTTP

Agent GUI (browser)

HTTP

Live Assist Server

Admin GUI



Agent GUI (browser)

Live Assist configuration DB



QA MySQL DB test driver

QA Web test driver/simulator

Figure 1: QA Test Driver (Grey with dot frame – simulated components; Green – QA Test Driver; Blue – Live Assist system components)

*Note: With above QA test driver, tester can easily follow or copy/paste the test driver sample codes and customize it for specific test purpose for many Live Assist feature/functional automations and testing.*

1. The basic Nodejs modules used for QA Live Assist test and QA test drivers with Mocha (Nodejs) test framework:

- Mocha installation:

* + Nodejs: <http://nodejs.org/>
  + Mocha(test framework): <http://visionmedia.github.io/mocha/>

- HTTP server for testing:

* + 'express', http server setup for QA testing/faking: <https://github.com/visionmedia/express>
  + 'ejs' modudle[: https://github.com/visionmedia/ejs](\\\\mt-nasrops01\\qa\\LiveAssist\\QA_Doc\\: https:\\github.com\\visionmedia\\ejs)
  + 'nock', for HTTP server/backend database mocking/stubing, and http test details tracing: [https://github.com/flatiron/nock](https://github.com/flatiron/nock%20)

- HTTP client for testing:

* + 'supertest': REST API test client, <https://github.com/visionmedia/supertest>
  + 'request': Powerful simplified http test client, <https://github.com/mikeal/request>

- Logging module:

* + 'winston logging nodejs module, <https://github.com/flatiron/winston>

- Monitoring/Spying:

* + 'sinon': <http://sinonjs.org/>

- Assertion library:

* + 'chai': <http://chaijs.com/>
  + 'expect': <https://github.com/LearnBoost/expect.js/>
  + 'should' (optional): <https://github.com/visionmedia/should.js/>
  + 'sinon-chai': <http://chaijs.com/plugins/sinon-chai>

- Flow /process control module:

* + 'async': <https://github.com/caolan/async>
  + ‘promise’ (integrated with selenium webdriver, optional): <https://github.com/stackp/promisejs>

- Web test Nodejs module:

* + - Selenium-WebDriver: <http://docs.seleniumhq.org/docs/03_webdriver.jsp>
    - Selenium WebDriverJS user guide: <https://code.google.com/p/selenium/wiki/WebDriverJs>
    - Selenium WebDriver NodeJS doc:

<http://selenium.googlecode.com/git/docs/api/javascript/index.html>

* + - Selenium WebDriver Java API doc: <http://selenium.googlecode.com/git/docs/api/java/index.html>
    - Mocha-PhantomJS test runner(optional): <https://github.com/metaskills/mocha-phantomjs>
    - WebDriverJS bindings for nodejs module (reference, optional): <https://github.com/camme/webdriverjs>
    - AngularJS test Protractor module: <https://github.com/angular/protractor>
* DB test nodejs module
  + Node-MySQL module for DB: <https://github.com/felixge/node-mysql>

2. Basic Test Folder/Directory setup:

/usr/src/test ---- Main test folder(all QA test infrastructure/driver/build)

* /test\_output ---- test logs (diagnostic log from each QA test case)
* /reports ---- test result/report (for Anthill Pro and Rally updates)
* /QA\_TEST ---- Mocha test set
* /LiveAssist ---- Live Assist server build (unzipped)
* run\_test.bat ---- run Mocha regression test and report result to screen
* run\_xunit.bat ---- run Mocha regression test and report to XML file (JUnit format)
* Package.json ---- Test modules dependency setup

3. The basic Mocha test infrastructure and QA created test drivers:

***- App Test driver:***

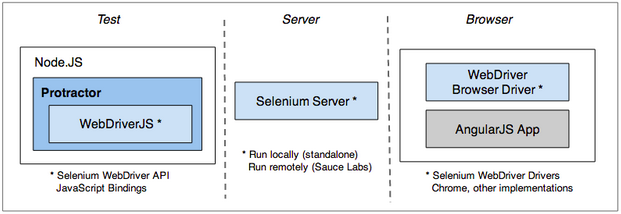
1. Load modules for testing, including chai (assertion lib), async, Winston, request, etc.
2. Setup spy or sandbox (optional, if necessary)
3. Define variables for QA test needs
4. Hijack the process.stdout.write function and stderr.write (optional)
5. QA test server or mocking setup (express or nock, optional if necessary)
6. HTTP request test client setup
7. Before hooks/handler for setup ‘before’ case run
8. After hooks/handler for cleanup ‘after’ case run
9. Test case main body:
   1. Trigger the HTTP request
   2. Collect responses
   3. assertion check
   4. Logging
   5. Exit with case defined message (pass or QA customized error)

For the QA App test driver on Live Assist API (RESTful like api or app APIs) testing with sample functions/codes/usage, please refer at: [\\mt-nasrops01\qa\LiveAssist\Test\_Set\QA\_Sample\_test\_and\_driver\Test\_drivers\_sample\QA\_api\_test\_driver\_sample](file:///\\mt-nasrops01\qa\LiveAssist\Test_Set\QA_Sample_test_and_driver\Test_drivers_sample\QA_api_test_driver_sample)

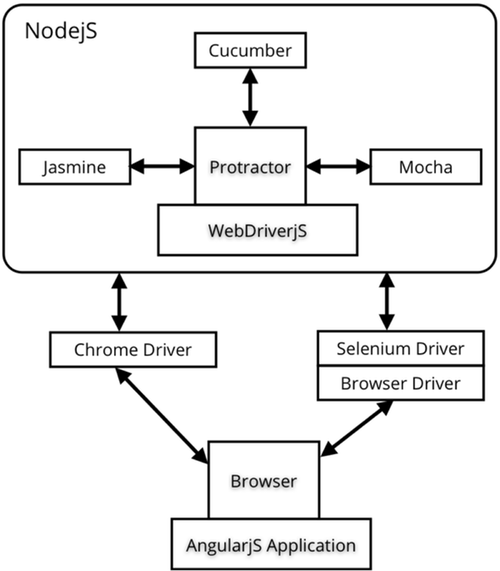
* ***Web Test Drive (with webdriverJS + Selenium + protractor):***

The QA web test driver is based on popular web test used framework/libraries from selenium, webdriverJS and protractor:

* Protractor is an end-to-end test framework for AngularJS applications. Protractor is a Node.js program that supports the Jasmine, Mocha, and Cucumber test frameworks.
* Selenium is a browser automation framework. Selenium includes the Selenium Server, the WebDriver APIs, and the WebDriver browser drivers.
* Protractor works in conjunction with Selenium to provide an automated test infrastructure that can simulate a user’s interaction with an Angular application running in a browser or mobile device.



QA created the web test driver with sample codes/functions based on above framework/libraries to ease QA test on Live Assist web GUI testing including HTML and AngularJS app.



**Fig. The agent GUI protractor + selenium test architecture**

The basic functions/flow:

1. Load web test drivers (selenium webdriver, webdriverjs, phantomjs etc)
2. Create web driver object along with specific browser driver (Firefox, Chrome, IE…)
3. Get/assign the driver with specific web page URL
4. Wait for expected web page element (form, button, radio, … etc element)
5. Find specific HTML element and triggering case customized action (input text, click button, …etc)
6. Check/verify the selected action/element action(optional)
7. Submit the form/page along with customized action
8. Quit created web driver

 For QA web test driver setup:

                - Selenium standalone server:

                                a) For web page testing by selenium, user need to start selenium server first before run any tests (IP: 127.0.0.0, port: 4444):

                                                java -jar c:\LiveAssist\_test\test\QA\_TEST\selenium-server-standalone-2.xx.0.jar

                                b) Copy the selenium JAR to \LiveAssist\_test\test folder

                - WebdriverJs (Selenium javascript bindings for nodejs, optional if protractor installed):

                                Installed by npm globally: "npm install -g webdriverjs"

* Protractor
  + Installed by npm globally: “npm install –g protractor”

                - Chrome driver:

                                a) install chromedriver.exe at: \LiveAssist\_test\test\QA\_TEST

                                b) using with selenium + webdriverJs + chrome driver (note: compatible with phantomjs) by create new batch file 'run:

                                       start "Selenium + ChromeDriver" /min cmd /k java -jar c:\LiveAssist\_test\test\QA\_TEST\selenium-server-standalone-2.xx.0.jar -Dwebdriver.chrome.driver=c:\LiveAssist\_test\test\QA\_TEST\chromedriver.exe

                - Firefox driver:

                                It included in the Selenium by default, so no need special installation or setup

For the web test driver with Selenium on HTML page testing and sample functions/codes/usage, please refer at: [\\mt-nasrops01\qa\LiveAssist\Test\_Set\QA\_Sample\_test\_and\_driver\Test\_drivers\_sample\QA\_gui\_test\_driver\_sample-selenium](file:///\\mt-nasrops01\qa\LiveAssist\Test_Set\QA_Sample_test_and_driver\Test_drivers_sample\QA_gui_test_driver_sample-selenium)

* QA web test driver global configuration

*//Global Live Assist QA Mocha test parameters/variables settings****exports***.**options** = {  
**server** : **"10.3.41.56"**, *//AHP: mt-* **port** : 8083, *//default Live Assist server port* **test\_location** : **"c:\\LiveAssist\_test\\test"**, *//test base folder* **test\_Url** : **"https://10.3.41.56:8446/liveassist/app"**, **selenium\_server** : **"http://localhost:4444/wd/hub"**, *//selenium server URL* **browser\_driver** : **"firefox"**, *//config browser driver to "firefox". NOTE: need to start selenium standalone server + Firefox driver first (or batch file)  
// browser\_driver : "chrome", //config browser driver to "chrome". NOTE: need to start selenium standalone server + Chrome driver first (or batch file)*

**ptor\_timeout\_regression** : 60000, *// protractor driver & script time out* …

* QA web test driver wrapper for firefox/chrome:

*/\*\* QA browser test driver wrapper for Firefox/Chrome \*\*/***function** *ProtractorDriver*(selenium\_server, browser\_driver) {  
 **this**.**seleniumAddr** = selenium\_server;  
 **this**.**driverName** = browser\_driver;  
  
 **this**.**protractor** = *require*(**'protractor'**); *//require protractor library* **this**.**driver** = **null**;  
  
 *//Add for Chrome driver* **this**.**chromeOptions** = **this**.**protractor**.Capabilities.**chrome**();  
 **this**.**chromeOptions**[**'caps\_'**].**chromeOptions** = {  
 **args**: [**'--disable-web-security'**, **'--start-maximized'**]  
 };  
}  
  
*// Add methods  
ProtractorDriver*.**prototype**.create = **function**(){  
 **var** self = **this**;  
  
 *//check which browser driver protractor need to use* **if** (self.**driverName** == **"firefox"**) {  
 self.**driver** = **new** self.**protractor**.Builder(). *//define driver instance for Selenium server used/wrapped with protractor* usingServer(self.**seleniumAddr**).  
 withCapabilities(  
 self.**protractor**.Capabilities.firefox() *//Firefox test* ).**build**();  
 } **else if** (self.**driverName** == **"chrome"**) {  
 self.**driver** = **new** self.**protractor**.Builder(). *//define driver instance for Selenium server used/wrapped with protractor* usingServer(self.**seleniumAddr**).  
 withCapabilities(  
 self.**chromeOptions** *//Chrome test* ).**build**();  
 } **else** {  
 self.**driver** = **null**;  
 }  
  
 *//return self.protractor.wrapDriver(self.driver); //old* **return** [self.**protractor**.wrapDriver(self.**driver**), self.**protractor**];  
};  
  
*//exports function for protractor driver creation****exports***.createPtorDriver = **function**(selenium\_server, browser\_driver){  
 **return new** *ProtractorDriver*(selenium\_server, browser\_driver).create();  
};

* Using the QA test global config/wrapper/driver in web GUI test cases:

**var** TestConfig = *require*(**'../Config/TestConfig.js'**);  
**var** TEST\_HOST = TestConfig.**options**.**server**; *//Ex: '10.3.41.59' change it if test server changed***var** TEST\_PORT = TestConfig.**options**.**port**; *//ex: '8080', change it if test server changed***var** test\_location = TestConfig.**options**.**test\_location**; *//test base folder***var** selenium\_server = TestConfig.**options**.**selenium\_server**;  
**var** test\_url= TestConfig.**options**.**test\_Url**; *//agent GUI URL*

…

**var** protractor = **null**;  
**var** ptor = **null**;

…

**var** ptorDriver = *require*(**'../Config/BrowserTestDriver.js'**);  
**var** ret = ptorDriver.createPtorDriver(selenium\_server,TestConfig.**options**.**browser\_driver**);  
ptor = ret[0]; protractor = ret[1];

…

For the web test driver with Protractor library on AngularJS testing and sample functions/codes/usage please refer at: [\\mt-nasrops01\qa\LiveAssist\Test\_Set\QA\_Sample\_test\_and\_driver\Test\_drivers\_sample\QA\_gui\_test\_driver\_sample-protractor](file:///\\mt-nasrops01\qa\LiveAssist\Test_Set\QA_Sample_test_and_driver\Test_drivers_sample\QA_gui_test_driver_sample-protractor)

* ***QA MySQL DB test driver***

*The QA DB test driver coded in Nodejs with ‘mysql’ nodejs module, which supports and ease tester to automation tests with: new DB connection, disconnect, basic supported DB transactions test include: query, update, insert and delete, error handling & rollback, and also multiple DB transactions support.*

1. ***The test driver setup/config in test property file example:***

*LA\_mysql\_host : "10.3.41.56", //in most cases, the mysql host is the same as Live Assist server, if not, need to configure different mysql db host here*

*LA\_mysql\_port : 3306,*

*LA\_mysql\_user : "liveassist",*

*LA\_mysql\_password : "liveassist",*

*LA\_mysql\_database : "liveassist",*

*Using the DB test driver from the QA Mocha test case:*

*//for mysql test driver*

*var mysql = require('mysql');*

*var mysql\_connection = mysql.createConnection({*

*host: TEST\_HOST,*

*port: TestConfig.options.LA\_mysql\_port,*

*user: TestConfig.options.LA\_mysql\_user,*

*password: TestConfig.options.LA\_mysql\_password,*

*database: TestConfig.options.LA\_mysql\_database*

*});*

*mysql\_connection.connect(); //connect MySQL test driver*

*Before the case end, the mysql driver need to be quit/destroy like this:*

*end\_mysqldriver: function(callback){*

*//for MySQL test driver*

*mysql\_connection.destroy();*

*callback(null, 0);*

*},*

1. ***The test driver used in the test case example & usage:***

*//MySQL driver select check test table2 after multiple DB transactions*

*function(callback){*

*mysql\_query = 'SELECT \* FROM ' + mysql\_table2;*

*var query = mysql\_connection.query(mysql\_query, function(err, rows){*

*if(err){*

*logger.info('\nMySQL test driver error: ' + err.stack);*

*callback(err, 7);*

*}*

*mysql\_ret = 'MySQL test driver SELECT query ok: ' + query.sql + '\nReturn: \n' + JSON.stringify(rows) + '\n';*

*logger.info(mysql\_ret);*

*var result=[0, mysql\_ret];*

*callback(null, result);*

*});*

*},*

*…*

*//MySQL driver single update and commit transaction*

*function (callback) {*

*mysql\_connection.beginTransaction(function(err) {*

*if (err) {*

*logger.info('MySQL test driver error: ' + err.stack);*

*callback(err, 5);*

*}*

*mysql\_query = 'UPDATE ' + mysql\_table2 + ' SET escalationClassifierId=\'DynamicEscalationClassifier\' WHERE applicationConfigurationId=4 AND name=\'mainmenu\'';*

*query\_ret = mysql\_connection.query(mysql\_query, function (err, rows) {*

*if (err) {*

*logger.info('MySQL test driver error: ' + err.stack);*

*callback(err, 5);*

*}*

*//commit*

*mysql\_connection.commit(function (err) {*

*if (err) {*

*mysql\_connection.rollback(function () {*

*logger.info('MySQL test driver error: ' + err.stack);*

*callback(err, 5);*

*});*

*}*

*mysql\_ret = 'MySQL test driver UPDATE query ok: ' + query\_ret.sql + '\n Changed: ' + rows.changedRows + ' rows' + '\n';*

*logger.info(mysql\_ret);*

*var result = [0, mysql\_ret];*

*callback(null, result);*

*})*

*});*

*});*

*},*

*...*

*//MySQL driver multiple update and commit DB transaction*

*function (callback) {*

*//Assign input table for updates*

*var test\_item\_table = test\_item1;*

*//execute update DB table transaction*

*mysql\_connection.beginTransaction(function (err) {*

*if (err) {*

*logger.info('MySQL test driver error: ' + err.stack);*

*callback(err, 4);*

*}*

*var cnt = 0;*

*async.whilst(*

*function () {*

*return cnt < test\_item\_table.length;*

*},*

*function (cb) {*

*mysql\_connection.beginTransaction(function (err) {*

*if (err) {*

*logger.info('MySQL test driver error: ' + err.stack);*

*cb(err);*

*}*

*mysql\_query = 'UPDATE ' + mysql\_table1 + ' SET value=? WHERE applicationContextId=? AND name=?';*

*logger.info('\nFor debug: ' + mysql\_query + '; ' + test\_item\_table[cnt].value + '; ' + test\_item\_table[cnt].applicationContextId + '; ' + test\_item\_table[cnt].name + '\n');*

*var query\_ret = mysql\_connection.query(mysql\_query, [test\_item\_table[cnt].value, test\_item\_table[cnt].applicationContextId, test\_item\_table[cnt].name], function (err, rows) {*

*if (err) {*

*logger.info('MySQL test driver error: ' + err.stack);*

*cb(err);*

*}*

*mysql\_ret = 'MySQL test driver UPDATE query ok: ' + query\_ret.sql + '\n Changed: ' + rows.changedRows + ' rows' + '\n';*

*logger.info(mysql\_ret);*

*//*

*++cnt;*

*setTimeout(cb, 200);*

*});*

*});*

*},*

*function (err) {*

*if (err) { //*

*logger.info('MySQL test driver UPDATE error: ' + err.stack);*

*callback(err, 4);*

*} else {*

*//DB transaction commit*

*mysql\_connection.commit(function (err) {*

*if (err) {*

*mysql\_connection.rollback(function () {*

*logger.info('MySQL test driver error: ' + err.stack);*

*cb(err);*

*});*

*}*

*mysql\_ret = 'MySQL test driver UPDATE commit ok. \n';*

*logger.info(mysql\_ret);*

*//*

*var result = [0, mysql\_ret];*

*callback(null, result);*

*});*

*}*

*}*

*);*

*});*

*},*

*…*

1. *Then user can easily follow or copy the test driver sample codes and customize it for specific test purpose with DB. The following function code and samples provided to ease the automation test:*

* *Check the DB connection ok with basic query*
* *DB basic query support*
* *DB ’UPDATE’ single or multiple transactions*
* *DB ‘INSERT’ single or multiple transactions*
* *DB ‘DELETE’ single or multiple transactions*
* *Error handling and rollback support*
* *Assertion check support*
* *Logging support*
* *Integrated with QA Mocha regression test framework*

*Specially, for the above update, insert and delete multiple transaction, the test driver sample code using Nodejs Async while loop (shorter codes and reliable)*

For the DB QA test driver (MySQL) with sample functions/codes/usage, please refer at:  [\\mt-nasrops01\qa\LiveAssist\Test\_Set\QA\_Sample\_test\_and\_driver\Test\_drivers\_sample\QA\_DB\_test\_driver\_sample](file:///\\mt-nasrops01\qa\LiveAssist\Test_Set\QA_Sample_test_and_driver\Test_drivers_sample\QA_DB_test_driver_sample)

***QA-user SIP Call Test drive(Perl, QA-user/NVP3.1; Deprecated from PSI4, replaced by sip server/service)***

*Phase 2 (expanding to end-to-end acceptance Test):*

QA App Test Driver/SimulatorLive Assist ServerQA Agent Test Driver/SimulatorHTTPHTTPNVP+IVR appWebHTTPHTTPAgent Client ServerHTTPSIP call/QA-User Test DriverSIPHTTPHTTPUser select/QA Web Test DriverHTTPUser select/QA Agent Web Test Driver

Agent Client



                      Fig 2: QA Test Driver (Grey with dot frame – LA commands/requests simulated component; Green – QA Test Driver; Blue – Test target component)

                                Test end to end as acceptance test (Manual or semi-auto/automated)

***Using QA-user and Web Test Driver for Live Assist end to end test simulation & Automation(Perl, NodeJS/Mocha)***

***Test framework/libraries updates:***

The QA test infrastructure & QA test drivers based on many nodejs framework/libraries/modules, such as: nodejs, Mocha test framework, protractor/webdriverjs, selenium server, firefox driver, MySQL module, assertion modules, etc, therefore, it required QA to regularly update them especially for web test driver which need to be keep updates with new Firefox versions.

Here are some major steps example:

* For Nodejs:
  + Download latest version from: <http://nodejs.org/download/>
  + For windows test machines, node-v<new\_version>-x86.msi recommended, and for Linux, the binaries installer with .tar.gz extension recommended.
  + Install the new nodejs version on test machines with default settings following instructions from the package. The npm package also got updated which will be used for other modules update/install after.
* For Firefox driver:
  + Firefox browser:
    - Updated via ‘Help’ -> ‘About Firefox’
    - Restart Firefox after updates
  + Notes:
    - Once the Firefox version upgraded, the related selenium server + protractor + firefox driver which used for the QA GUI test automation also need to be updated accordingly.
    - The GUI test framework/library (selenium/protractor/firefox driver) usually take a few days/weeks behind Firefox version upgrade from Mozilla, so we should schedule this updates after and always check the selenium & protractor website to confirm the current new version support for the new firefox version testing.
* For selenium server:
  + Download latest selenium server from: <http://docs.seleniumhq.org/download/>
  + Moved the standalone selenium server to test machine under: \LiveAssist\_test\test\QA\_TEST
  + Update the batch file (start\_selenium.bat) to start the new selenium server using the latest version and comment old version, for example:

REM start "Selenium server" /min cmd /k java -jar c:\LiveAssist\_test\test\QA\_TEST\selenium-server-standalone-2.42.2.jar

REM using latest new version:

start "Selenium server" /min cmd /k java -jar c:\LiveAssist\_test\test\QA\_TEST\selenium-server-standalone-2.43.1.jar

* For Protractor library/framework:
  + Check if protractor has new release: <https://github.com/angular/protractor/releases>
  + Check local install protractor version:
    - Protractor –version
  + Install the new updates:
    - Npm update –g protractor
* For all other nodejs modules updates via NPM:
  + Local modules: Npm update
  + Global modules: npm update –g
  + Notes:
    - Live Assist QA automation test need to use both above commands to update local & global modules.
    - The modules may be updated including: Mocha test framework, request, async, mysql, assertions (expect/should/chai), reporter, logging modules, etc
    - Check updated modules by:
      * Local modules: Npm list
      * Global modules: Npm list -g

Note: For Anthill pro QA regression test env on protractor library & mocha test framework updates, except above steps, user need to log in as ‘Administrator’, then run npm update command with specific package name like this:

npm update –g protractor

npm update –g mocha