***QA Automation Test Design & Implementations***

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

Goal: Build a basic QA Mocha test infrastructure and App test driver for Live Assist RESTful APIs testing and RPC over HTTP APIs testing, such as: VXML APIs (App APIs)/Live Assist Command APS, session storage APIs, time info APIs, DEBUG APIs, escalation classifier APIs, etc.

*Phase 1 (QA Simulator/Test Driver):*

QA App Test Driver/SimulatorLive Assist ServerQA Agent Test Driver/SimulatorHTTPHTTPIVRWebHTTPHTTPAgent Client ServerAgent ClientHTTP

Agent Client

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

HTTP commands simulators & test drivers:

1. App Simulator & Test Drivers: Simulate commands and requests from IVR application (later WEB in PSI4) and verify returned results/logs/responses between the component ‘Live assist commands’ and ‘Live Assist Server’;
2. Agent Simulator & Test Drivers: Simulate commands and requests from Agent (may include ‘call center server’ + ‘call center agent client’) and verify the returned results/logs/responses between the Agent Router Interface (part of Live Assist server) and Agent Server/Client.

1. The basic modules for the Mocha Live Assist test infrastructure setup:

- 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](:%20https:/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:

* + '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:

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

- Web Test Driver:

* + - 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>

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 test driver:

***- 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)

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

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:

                                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-2.35.0.jar

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

                - WebdriverJs (Selenium javascript bindings for nodejs):

                                a) instally by npm globally: "npm install -g webdriverjs"

                - PhantomJs:

                                a) Install by npm globally: "npm install -g phantomjs"

                                b) path specific setup: After NPM install, add gobal path for phantomjs.ext such as: "C:\Documents and Settings\root\Application Data\npm\node\_modules\phantomjs\lib\phantom"

                - chrome dirver:

                                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-2.35.0.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

* ***QA web with AngularJS test driver (Protractor + Selenium)***

***Test setup/config in test property file example:***

***server : "10.3.41.56", //AHP: mt-ray-vm03***

***port : 8080, //default Live Assist server port***

***test\_location : "c:\\LiveAssist\_test\\test", //test base folder***

***test\_Url : "https://10.3.41.56:8443/liveassist/app", //for Live Assist AngularJS GUI selenium\_server : "http://localhost:4444/wd/hub", //selenium server URL***

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

***ptor\_timeout\_load : 45000, // protractor driver & script time out***

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

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

***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***

***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)***

***The detailed sample test driver and codes please check 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)***

*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 3: 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)***