

**OpenSSO QA Test Automation**  
Agents module Testing Setup & Development

## Revision History

Date	Version	Author	Comments
09/25/07	1	Rahul Misra	Initial Version
03/10/08	1.1	Indra Thangasamy	Updated document to reflect current framework.
04/18/08	1.2	Nithya Srinivasan	Updated document to reflect current framework.

## Reviewers

Role	Name	Review and Aprrove	Date(mm/dd/yyyy)

## Table of Contents

1.0 Introduction.....	4
2.0 Requirements.....	4
2.1 Product Setup and Configuration .....	4
2.2 Test Data Configuration.....	9
3.0 Agents Tests .....	10
3.1 What Is Being Tested.....	10
3.2 Test Structure and Layout .....	11
3.3 Grouping.....	12
4.0 How To Execute the Agents module Tests.....	13
4.1 Setup the qatest client framework.....	13
4.2 Complete the prerequisite steps as described in section 2.0.....	13
4.3 Create Local Configuration properties.....	14
4.2 Test Execution Details.....	14
5.0 Tests Implementation details.....	15
5.1 Current Tests Classes:.....	15
5.2 Properties Files.....	17
5.3 Common Files.....	17
6.0 Interpreting Tests Results.....	18
7.0 Debugging the agents Automated test failures.....	19
8.0 Adding New Tests.....	20
8.1 Adding New Tests to an existing Class.....	20
8.2 Adding New Tests using a New Java Class.....	21
8.3 General Guidelines to Develop New Tests .....	21
9.0 Summary .....	22

# 1.0 Introduction

The goal of this document is to describe the required details to execute and augment agents module tests in the OpenSSO qatest framework. In summary the following details will be covered in this document.

- The Requirements for this module
- How tests are organized.
- Execution details
- Tests in the framework
- Interpreting the report
- Debugging the test failures
- How to add new tests

## 2.0 Requirements

**PreRequisite :** OpenSSO Test Framework is a must read to understand the framework and set the framework to execute the tests.

This section contains the information about the minimum requirements to run the agents module tests. This is a must read section for any one who wants to run the agents module automated tests.

### 2.1 Product Setup and Configuration

Before executing the tests there are certain prerequisites that has to be satisfied for a successful execution of the agents module tests.

**Policy/J2EE Server Setup:** The agents module tests can be executed against Access Manager 7.1 version (by setting agentsGlobal.executeAgainstOpenSSO=false) or OpenSSO (and its commercial version of Federated Access Manager 8.0)server.

The only difference is that if you are planning to execute agents tests pointing to a Access Manager 7.1 policy server then the resources policies, agent profile and the associated identity subjects has to created prior to the agents module test execution. This can be achieved using a single XML file. A few additional changes needs to be done to the property file to support policy evaluation with all policies present at the same time. This is under progress.

When the OpenSSO is used as FAM server then the policies, agent profile and the associated identity subjects are created automatically by the framework.

**Policy/J2EE Agents Setup:** Currently, Web Server Agents version 2.2/3.0 & Java EE agents version 2.2/3.0 are supported with this automation framework. You can test Sun Java ES Web Server agents, Apache agents and Java EE Application Server using this framework.

Policy Agents installation and configuration is out of scope for this framework, it expects the policy agent is already installed and provisioned with proper resources as stipulated by this framework.

Java EE tests expects that the J2EE Agent is already installed and the agentsample deployed, tested manually. Additionally the following changes are required.

#### **Changes required for running J2EE Agentsample tests:**

- Change the Role mapping in the sun-web.xml, sun-application.xml to ou=role, recreate the agentsample.ear and redeploy it.
- Add the following to the AMAgent.properties file.

*com.sun.identity.agents.config.privileged.attribute.type[0] = Role*

*com.sun.identity.agents.config.privileged.attribute.toLowerCase[Role] = true*

- The default showHttpHeaders.jsp file should be replaced with the one from qatest/www/jsp/showHttpHeader.jsp file.
- Troubleshooting J2EE agent is beyond the scope of this document. Please refer to

the [J2EE-Troubleshooting wiki](#).

- NOTE : Though generally ,referred to as AMAgent.properties, please interpret it as follows depending on the configuration you are testing.
  - AMAgent.properties - For 2.2 agents
  - FAMAgentConfiguration.properties - For 3.0 agent with local Config
  - For 3.0 Agent with Centralized Agent, configuring the agent profile is taken care by qatest. No user action required.

### Additional Configurations:

When the policy/j2ee agents are configured one must make sure following configurations are taken care, if not agents tests bound to fail.

- Provision the required resources in the server.
  - For Web Agents the resources should be in the web server's document root directory
  - For J2EE Agents, the resources should be placed under a resources directory under the agentsample application .

For.eg. /agentsample/agentservlets\_war/resources/allow.html

- Edit the following AMAgent.properties and set the corresponding values as given below
- Fetch Profile/Session/Response attributes of Identities as HTTP Header for policy agents.
  - Fetch mode's supported - *HTTP\_HEADER or HTTP\_COOKIE* for Web Agents, & *HTTP\_HEADER or HTTP\_COOKIE or REQUEST\_ATTRIBUTE* for J2EE Agents.
  - Sample values for 2.2 Web Agents

```
com.sun.am.policy.agents.config.profile.attribute.fetch.mode=HTTP_HEADER
```

```
com.sun.am.policy.agents.config.profile.attribute.map=cn | PROFILE-cn,iplanet-am-user-alias-list | PROFILE-alias,nsrole | PROFILE-nsrole
```

```
com.sun.am.policy.agents.config.session.attribute.fetch.mode=HTTP_HEADER
```

```
com.sun.am.policy.agents.config.session.attribute.map=sun.am.UniversalIdentifier | SESSION_UniversalIdentifier,MyProperty | SESSION_MyProperty
```

```
com.sun.am.policy.agents.config.response.attribute.fetch.mode=HTTP_HEADER
```

```
com.sun.am.policy.agents.config.response.attribute.map=statSingle |
```

*RESPONSE\_statSingle,statMultiple | RESPONSE\_statMultiple,cn | RESPONSE\_cn*

- Sample values for 2.2/3.0 J2EE Agents

```
com.sun.identity.agents.config.profile.attribute.fetch.mode = HTTP_HEADER
com.sun.identity.agents.config.profile.attribute.mapping[cn] = HTTP_PROFILE_CN
com.sun.identity.agents.config.profile.attribute.mapping[iplanet-am-user-alias-list]=HTTP_PROFILE_ALIAS
com.sun.identity.agents.config.profile.attribute.mapping[nsrole] = HTTP_PROFILE_NSROLE
com.sun.identity.agents.config.session.attribute.fetch.mode = HTTP_HEADER
com.sun.identity.agents.config.session.attribute.mapping[sun.am.UniversalIdentifier] =
HTTP_SESSION_UNIVERSALIDENTIFIER
com.sun.identity.agents.config.session.attribute.mapping[MyProperty] = HTTP_SESSION_MYPROPERTY
com.sun.identity.agents.config.response.attribute.fetch.mode = HTTP_HEADER
com.sun.identity.agents.config.response.attribute.mapping[statSingle] = HTTP_RESPONSE_STATSINGLE
com.sun.identity.agents.config.response.attribute.mapping[statMultiple] =
HTTP_RESPONSE_STATMULTIPLE
com.sun.identity.agents.config.response.attribute.mapping[cn] = HTTP_RESPONSE_CN
```

- Sample values for 3.0 Web Agents

```
com.sun.identity.agents.config.profile.attribute.fetch.mode = HTTP_HEADER
com.sun.identity.agents.config.profile.attribute.mapping[cn] = PROFILE_CN
com.sun.identity.agents.config.profile.attribute.mapping[iplanet-am-user-alias-list]=PROFILE_ALIAS
com.sun.identity.agents.config.profile.attribute.mapping[nsrole] = PROFILE_NSROLE
com.sun.identity.agents.config.session.attribute.fetch.mode = HTTP_HEADER
com.sun.identity.agents.config.session.attribute.mapping[sun.am.UniversalIdentifier] =
SESSION_UNIVERSALIDENTIFIER
com.sun.identity.agents.config.session.attribute.mapping[MyProperty] = SESSION_MYPROPERTY
com.sun.identity.agents.config.response.attribute.fetch.mode = HTTP_HEADER
com.sun.identity.agents.config.response.attribute.mapping[statSingle] = RESPONSE_STATSINGLE
com.sun.identity.agents.config.response.attribute.mapping[statMultiple] =
RESPONSE_STATMULTIPLE
```

```
com.sun.identity.agents.config.response.attribute.mapping[cn] = RESPONSE_CN
```

- For 2.2 Agents & 3.0 Agents using local config, ensure that attribute.fetch.mode in the AmAgent.properties file has the same value for property **agentsGlobal.headerFetchMode**
- Other property values that need to be modified.

```
com.sun.am.policy.agents.config.anonymous_user.enable=true
```

```
com.sun.am.policy.agents.config.accessdenied.url=AGENT_SERVER_PROTO://AGENT_SERVER_HOST:AGENT_SERVER_PORT/deny.html
```

```
com.sun.am.policy.agents.config.notenforced_list=AGENT_SERVER_PROTO://AGENT_SERVER_HOST:AGENT_SERVER_PORT/cgi-bin/headers.cgi  
AGENT_SERVER_PROTO://AGENT_SERVER_HOST:AGENT_SERVER_PORT/notenf.html
```

For example

```
com.sun.am.policy.agents.config.notenforced_list=http://myagent.example.com:7080/cgi-bin/headers.cgi  
http://myagent.example.com:7080/notenf.html
```

```
com.sun.am.policy.agents.config.accessdenied.url=http://myagent.example.com:7080/deny.html
```

NOTE: For 3.0 Web Agents, headers.cgi needs to be a protected resource. Please create a Policy allowing access to headers.cgi for Authenticated users.

- [headers.cgi](#) script can be found under the [qatest/scripts](#) directory of the CVS source.
- The agentsample list of entries in the Notenforced list should be done in case of J2EE Agent.
- **Configure the test resources:**
  - In case of Web agents, copy the agents\_resources.jar from [qatest/data/agents](#) directory into your web server's document root directory and do a jar xvf agents\_resources.jar.
  - In case of J2EE agents, create a directory called resources under the application context and unjar into the agentsample's resources directory.



## 2.2 Test Data Configuration

The test data for the agents module obtained from the following five properties files, these property files are located under [qatest/resources/agents](#)

**AgentsTests.properties**: This property file contains the data for the expected results, identities for subjects, IP and DNS conditions for the various resources. If you are using resource files in the web container which does contain different strings to match for, then you need to include those strings in this file in the appropriate place. One thing you may need to change here is the IP address range and DNS name.

1) Change the following properties so that the ip of the client machine is within the range specified here.

1. AgentsTests2.policy0.condition0.att0=StartIp=**192.18.21.85**  
AgentsTests2.policy0.condition0.att1=EndIp=**192.18.118.85**
2. AgentsTests2.policy2.condition0.att0=StartIp=**192.18.18.80**  
AgentsTests2.policy2.condition0.att1=EndIp=**192.18.118.90**
3. AgentsTests2.policy6.condition0.att0=StartIp=**192.18.18.80**  
AgentsTests2.policy6.condition0.att1=EndIp=**192.18.118.90**
4. AgentsTests2.policy8.condition0.att0=StartIp=**192.18.18.80**  
AgentsTests2.policy8.condition0.att1=EndIp=**192.18.118.90**

2) The dns name of the client should match the following properties

1. AgentsTests1.policy0.condition0.att0=DnsName=**\*.red.iplanet.com**
2. AgentsTests2.policy4.condition2.att0=DnsName=**red.iplanet.com**
3. AgentsTests2.policy5.condition2.att0=DnsName=**red.iplanet.com**
4. AgentsTests2.policy8.condition2.att0=DnsName=**red.iplanet.com**

**HeaderAttributeTests.properties**: This file supplies the required data for running the tests that are related to fetching the User profile, session and policy response attributes. You can customize this properties file to test any of the valid attributes from the Policy server. But remember you need to sync up the AMAgent.properties appropriately whenever a change happened in the header attributes in this file.

**GeneralAgentTests.properties**: This property file contains the data for running the general agents tests exercising the policy subjects.

**agentsGlobal.properties:** This property file supply the key information for the agents to communicate with the server, the agent's profile user name and password. This must match what is given in the AMAgent.properties while configuring the agents, otherwise all the tests will fail. This file will also contain the agents resources that needs to protected. **Remember to update this file with the correct URLs for the resources.**

**J2EEAgentTests.properties:** This property file supplies the subject, policy information needed to test the AgentSample execution.

**HotSwapProperties.properties:** This property file supplies the property name,value pair for running the HotSwap Test Cases for 3.0 Agents with Centralized Agent configuration.

## 3.0 Agents Tests

### 3.1 What Is Being Tested

Agents module focus on testing the typical scenarios that can be used by the OpenSSO Policy/J2EE agents. Most of the core tests scenarios are covered including following features

Resources protected by various Policy subjects

- AMIdentity User Subjects,
- AMIdentityRoles
- AMIdentityGroups
- Authenticated Users.
- Realm
- LDAP Groups/Roles[TBD]
- Resources Protected by
- DNS Conditions
- IP Address Conditions
- AuthScheme Conditions
- Validate resource access based on active and inactive policies
- Validate resource access for different rules
- Wild card resources

- Wildcard in the port number
- Wildcard in the protocol
- Wildcard in DNS name
- Valid and invalid resource names
  
- Setting REMOTE\_USER header from session token property
- setting REMOTE\_USER header as anonymous
- Not enforced and access deny URI properties
- Session notification events
  - Session termination
  - Logout
- Case Sensitivity in the resources
- Setting Authenticated Identities' attributes as HTTP header.
- Fetch Session properties
- Fetch profile properties
- Fetch policy response attributes

The following are the specific tests for j2ee agents.

- Declarative & Programmatic protection by testing the agentsample bundled with the Agent.
- Header Attribute TCs with all 3 fetch modes - HTTP\_HEADER, HTTP\_COOKIE, REQUEST\_ATTRIBUTE.
- If agent being tested is of version 3.0 & above, hotswapping the agent properties and rerunning the tests with different properties.

## 3.2 Test Structure and Layout

The agents automation tests are structured in to multiple JAVA classes based on the features that are being tested, the features that are tested are listed in the subsection 3.1.

These tests are well organized into a very structured directory layout as described in the table below.

<code>&lt;QATEST_HOME&gt;/xml/testng</code>	Is a directory that consists all the testng XML files that describe what needs to be executed in the agents test module.
<code>&lt;QATEST_HOME&gt;/resources/agents</code>	Is a directory that contain all the resource properties file that define the testcase data
<code>&lt;QATEST_HOME&gt;/source/common/sun/identity/qatest/agents</code>	Is a directory that contains all sources of Java implementations for agents tests.
<code>&lt;QATEST_HOME&gt;/source/common/sun/identity/qatest/common/AgentsCommon.java</code>	Is a Java source file containing the common function required by agents module.
<code>&lt;QATEST_HOME&gt;/&lt;SERVER_NAME&gt;/built/classes</code>	Is a directory containing all the XML files that are generated at run time while executing the for agents module. The SERVER_NAME could be SERVER_NAME1 or SERVER_NAME2

### 3.3 Grouping

Agents module tests are categorized in to different groups to make it executable under various supported configurations such as

ff_ds	Flatfile for User management & directory Server as the backend repository, Deprecated
ds_ds	Directory Server for User management and the backend repository
ff_ds_sec	Flatfile for User management & directory Server as the backend repository with security enabled. Deprecated

ds_ds_sec	Directory Server for User management and the backend repository with security enabled
-----------	---

These values are supplied in to the framework's EXECUTION\_MODE variable.

## 4.0 How To Execute the Agents module Tests

Once you have checked out the test frame work there only few steps required to execute the agents module tests.

### 4.1 Setup the qatest client framework

Check out the qatest automation framework as described in the [OpenSSO Test Framework](#) document, The key files that are required to be copied to the lib directory of the automation framework include

- The openssoclientsdk.jar matching with server's build/version
- htmlunit jars – Get the latest 1.14 jars
- jetty embedded container jars
- testng 5.7 jars

### 4.2 Complete the prerequisite steps as described in section 2.0

You need to make sure that you have followed and completed steps meticulously. You can perform a manual test before running the framework.

### 4.3 Create Local Configuration properties

Once you check out the framework change directory to opensso/qatest, under this directory create a resources/Configurator-<SERVER\_NAME>.properties(SERVER\_NAME could be SERVER\_NAME1 or SERVER\_NAME2 depending on where your Agents are

configured to) that corresponds to your OpenSSO Policy server deployment. Please refer to OpenSSO QATest Test automation architecture document for details.

Customize the following parameters to suit your deployment

- build.properties file
- QATEST\_HOME
- TEST\_MODULE to agents
- EXECUTION\_MODE to appropriate group name as described in section 3.3
- Set REPORT\_DIR to a desired location

a typical build.properties would look like this

```
<project>
  <property name="QATEST_HOME" value="/export/opensso/qatest"/>
  <property name="EXECUTION_MODE" value="ds_ds"/>
  <property name="TEST_MODULE" value="agents"/>
  <property name="REPORT_DIR" value="/export/qatest-reports"/>
</project>
```

Once you complete these prerequisites steps, you can now kick off the ant task by simply running the following command to execute agents module tests.

```
ant -DSERVER_NAME1=Test_Host_Name module
```

The text in *italics* should be entered as it is

Test\_Host\_Name – Is the hostname of the server where opensso is deployed, You should have a Configuration-hostname.properties in your framework as described in the prerequisite section.

## 4.2 Test Execution Details

- Framework picks execution of Java classes in random order. Plus all the testcases inside the Java class are also executed in random order unless the order is forced by dependsOn annotation.
- Each Java class contains multiple tests marked with @Test annotation. Methods annotated by @BeforeClass, @BeforeTest are setup methods. They prepare the system for the tests to execute. The examples are adding the users, creating

policies etc. Methods annotated by @AfterClass, @AfterTest are cleanup methods. They bring system back to original state. The examples are delete the users created in setup method, delete the policies etc. These methods will be run even if the tests fail.

- Framework uses famadm.jsp to create policies and user identities on the server.
- ConfigureUnconfigure.java creates the agent profile before any Agent tests are executed and deletes the profile after all tests are done. However this is configurable from the properties files.

## 5.0 Tests Implementation details

This section talks about the implementation details of the agents module tests, if you are planning to add new tests to the agents module then you must read this section.

### 5.1 Current Tests Classes:

Current Tests in the qatest Framework for agents are described in the following table:

Test class	Properties file used	Description
AgentsTests.java	AgentsTests.properties	Policy evaluation tests for different resources, subjects and conditions
ProfileAttributeTests.java	HeaderAttributeTests.properties	Test related to profile attributes.
SessionAttributeTests.java	HeaderAttributeTests.properties	Test related to session attributes.
ResponseAttributeTests.java	HeaderAttributeTests.properties	Test related to response attributes.

GeneralAgentTests.java	GeneralAgentTests.properties	<p>Tests related to following:</p> <ul style="list-style-type: none"> <li>● Remote user from session token</li> <li>● Remote user anonymous</li> <li>● Not enforced by including the cgi script</li> <li>● Session notification events <ul style="list-style-type: none"> <li>○ Session termination</li> <li>○ Session logout</li> </ul> </li> <li>● Access denied url</li> <li>● Case ignore for resource</li> </ul>
J2EEAgentTests.java	J2EEAgentTests.properties	Tests the agentsample application bundled with the J2EE Agents.
J2EEAgentHotSwapTests.java	HotSwapProperties.properties , HeaderAttributeTests.properties	J2EE Agents specific tests using Centralized Agent Config. Has handles to the actual TC's located in other java classes
WebAgentHotSwapTests.java	HotSwapProperties.properties , HeaderAttributeTests.properties	Web Agents specific tests using Centralized Agent Config. Has handles to the actual TC's located in other java classes



AgentTests2_2.java	HeaderAttributeTests.properties	2.2 Agents specific tests. Has handles to the actual TC's located in other java classes
AgentTests3_0Local.java	HeaderAttributeTests.properties	Tests specific to 3.0 Agents using local configuration. Has handles to the actual TC's located in other java classes

## 5.2 Properties Files

Each test class may have a corresponding properties file. These files mainly contain the configuration and execution details used for testing. These details can be changed if needed.

## 5.3 Common Files

There are multiple common files which are used by agents tests.

- AgentsCommon.java  
This class contains the agents related common methods.
- IDMCommon.java  
This class contains methods related to identity management.
- SMSCommon.java  
This class contains methods related to service management.
- PolicyCommon.java  
This class contains methods related to policy management.

## 6.0 Interpreting Tests Results

- After execution of agents test module, the results will be located under `<REPORT_DIR>/<HOSTNAME_1>/<EXECUTION_MODE>/<EXECUTION_DATE_TIME>`.
- Open index.html residing in this directory from the browser. It will display the overall result of the test execution with the following details:
  - a link named “<EXECUTION\_MODE>-agents” which points to the detailed test report
  - the number of tests which passed
  - the number of tests which failed
  - the number of test skipped
  - a link to the TestNG XML file used in this test run
- To learn more about the specific tests click on the link “<EXECUTION\_MODE>-agents”. In the left frame of the resulting page, the individual results of all the tests which were executed. Passing tests will have a background color of green. Failed tests will have a background color of red.
- To find out more information on the results a particular test click on the “Results” link for that test. This will provide you more information about the test such as when the test was executed, the duration of the test in seconds, the test method being executed, and any exception that was thrown during execution of the test.
- To view all the log messages which were displayed for a particular test go to the file `<REPORT_DIR>/<HOSTNAME>/<EXECUTION_MODE>/<EXECUTION_DATE_TIME>/logs`. In this file, search for the name of the test of interest. Below the name the log records produced during the three phases of this test's execution, setup, verification, and cleanup, can be viewed.

# 7.0 Debugging the agents Automated test failures

- Execute the tests by setting the log level to FINEST in the global configuration properties file.
- Ensure that the agent and the server are time-syncd.
- If you are running the tests from a host that does not have its domain name set most likely the DNS domain conditions tests are not going to produce correct results. You need to make sure the IP(both valid and Invalid) numbers range has to be set appropriately based on your client's IP. This is one of the key for evaluating the policies with the IP/DNS based conditions.
- The tests are very sensitive to the string matched in the output pages, so make sure you use the proper agents resources with correct text in it.
- The CGI program/showHttpHeaders.jsp should not be modified in any manner because the header attribute tests are relying on the way this script render the pages, any change will cause string mismatch
- Make sure your role memberships are proper, if you have a filter role with filter objectclass=\* then, the users created by this test will also inherit this role but the validation code does not know about this role hence would fail
- These tests mostly depend on the notification and agent caching behavior, if your tests results are random, try increasing the notification time in the global Configuration-hostname.properties
- Best way to debug is to open the logs file from  
<REPORT\_DIR>/<HOSTNAME>/<EXECUTION\_MODE>/<EXECUTION\_DATE\_TIME>/logs, This file will have the matching string found/not found messages
- To learn more about the specific tests click on the link "<EXECUTION\_MODE>-agents". In the left frame of the resulting page, the individual results of all the tests which were executed. Click on the link for the failed tests to see the exception reported.
- If DNS conditions tests are failing then make sure your client's host is configured properly for naming services to resolve to FQHN of the host

- Also look at the debug logs on server and agent to get further details
- In case of J2EE Agents test failures, ensure that the tests pass manually. If yes, the amWebPolicy log file is a good place to start.

## 8.0 Adding New Tests

This section describes how to add new testcases to the existing or new java class. Agents testcases are divided into different features. New feature related tests should be grouped together and added in the new java class. Testcases related to already automated features should be added to existing classes.

### 8.1 Adding New Tests to an existing Class

- Before adding new testcase in the existing class, make sure you understand all the existing tests in that class.
- Logically new test case should belong to that class.
- If the class has dependencies between the tests then add a testcase in such a way that it doesn't break the existing dependencies. Plus make to add the new dependency for that test too.
- Add new properties in the properties file if needed.

### 8.2 Adding New Tests using a New Java Class

- Create a new class with appropriate class name. The properties files should have similar name. Please follow naming conventions described in the OpenSSO QATest document.
- The class should follow setup, test(s) and cleanup procedures. Cleanup should make sure it restores the server to the default state.
- Appropriate groups should be assigned to these newly added testcases.
- Update all the agents related testng xml files.

- Run agents module with all the tests and make sure all the tests including newly added tests are passing.

### **8.3 General Guidelines to Develop New Tests**

- To add the new test scenario, based on the feature it belongs to, add it to appropriate test class. Ideally new scenario shouldn't have any dependency on existing tests.
- In testNG, tests from each class are run in random order. Thus its better not to have dependency on other test. In case it is really needed, to control the order you can use dependsOnMethod annotation.
- Test related setup & cleanup should be done in already existing methods. After cleanup, the system should be brought back to the original state.
- Tests shouldn't use any command line utility.
- For all operations, one should rely on using API calls through client SDK, as far as possible.
- Write any reusable Agent functionality as a common method in AgentsCommon class such that other test also can use it.

## 8.0 Summary

This is test checklist to get you started. Lets take this simple quiz.

- Is qatest the latest cvs version ?
- Is the directory server for um ready?
- Have you deployed the server war?
- Is agent installed and configured with the qatest specific property changes (if your tests require manual configuring)?
- Are the resources in place in the agent container?
- For J2EE Agent tests, is agentsample deployed correctly?
- Are the property files modified?
  - AgentsGlobal.properties
  - AgentsTests.properties
  - configGlobalData.properties
  - Configurator-<Server-name>.properties
  - build.properties
- Are the jar files in place and are they the correct version mandated by the qatest framework?
- Is the openssoclientsdk.jar, same version as of the FAM server?
- Sure, there are no conflicting policies/filtered roles in your setup?
- Is your agent and server time syncd?

If your answer was a YES to all the above questions, then you are all set to start testing your agents using qatest.

Happy Testing !!!!