OpenSSO
Open Access . Open Federation

OpenSSO Test Framework

Automation Test Document

Document Version 0.6 07/21/2008

Revision History

Version	Date (mm/dd/yyyy)	Author	Description
0.1	07/05/2007	Rahul Misra	Created a formal document for the test automation framework
0.1	07/18/2007	Indira Thangasamy	Made some editorial changes
0.2	07/25/2007	Indira Thangasamy	Incorporated comments from the Team, Added the text for running against secured OpenSSO server
0.3	12/05/2007	Rahul Misra	Incorporated latest changes to qatest
0.4	02/07/2008	Rahul Misra	Incorporated latest changes and user comments to qatest
0.5	04/28/08	Rahul Misra	Removed the Sun proprietary tag in footer
0.6	07/21/08	Raul Misra	Updated to incorporated changes in qatest related to server properties and datastore properties inheritance

Reviews And Approvals

Role	Name	Review And Approve	Date (mm/dd/yyyy)
OpenSSO Governess			
Product Marketing			

Table Of Contents

1.0 Introduction

The purpose of this document is to describe the architecture of the OpenSSO quality test automation framework that is bundled in the OpenSSO code base. The scope of this documentation also covers the detailed sequence of steps that are to be performed in order to successfully invoke this automation framework. Additionally this documentation captures the following:

- The need for such a framework
- The test framework
- How tests are organized
- How to execute the testcases
- How to add new testcases
- How to modify existing testcases

2.0 Framework Requirements

The primary requirements are:

- The primary focus of this framework is the ability to perform scenario testing.
- Scenarios represent logical grouping of functionality to test a desired product feature
- Scenarios span across and/or within functional modules such as IdRepo, SSO and Authorization in order to create a realistic test scenario
- Scenarios encompasses product install and configuration. The functional testing tests the product against the libraries bundled with the installed product.
- Extensible and can be customized
- QA Automation framework should be able to execute test cases in the following modes:
    - Execute all test cases

    - Execute a specific module

- All QA related automated scenario testing on OpenSSO project should be able use this framework.

- It is not a requirement to work with the predecessor versions of OpenSSO such as Sun Java System Access Manager or Sun Java System Federation Manager

- This framework should be made available to the OpenSSO community.

- QA test framework should be used as a regression suite that run on nightly builds and will eventually constitute the quality criteria for promoting the OpenSSO builds.

Quality Criteria are the criteria that must be satisfied in order to release any product. Although QA is taking responsibility for writing and owning this document, the criteria defined in this document should be a collaborative effort based on feedback and suggestions from development, support, documentation and QA. This quality criteria must be approved by the OpenSSO governing council. The QA group will test the product to meet quality criteria to achieve product quality and readiness. The Quality Criteria ensures that the released product meets the expected quality to meet the customer deployments.

3.0 Test Environment Requirements

3.1 Test Tools

Following are the list of tools (and their supported versions) currently used by the framework.

This is a living document and this list can change in future, based upon the requirements.

1. Ant (1.6.5)

2. TestNG (5.5)

3. HtmlUnit (1.11)

4. Jetty (6.0)

5. JDK (1.5+)

3.2 OpenSSO Server Requirements

All tests require that opensso.war is deployed in a container and client programs have access to openssoclientsdk.jar.

4.0 System Architecture

The following diagram represents the overall architecture of the QA test automation framework. Automated tests can be executed from the same host where the OpenSSO system is configured or from a remote host.

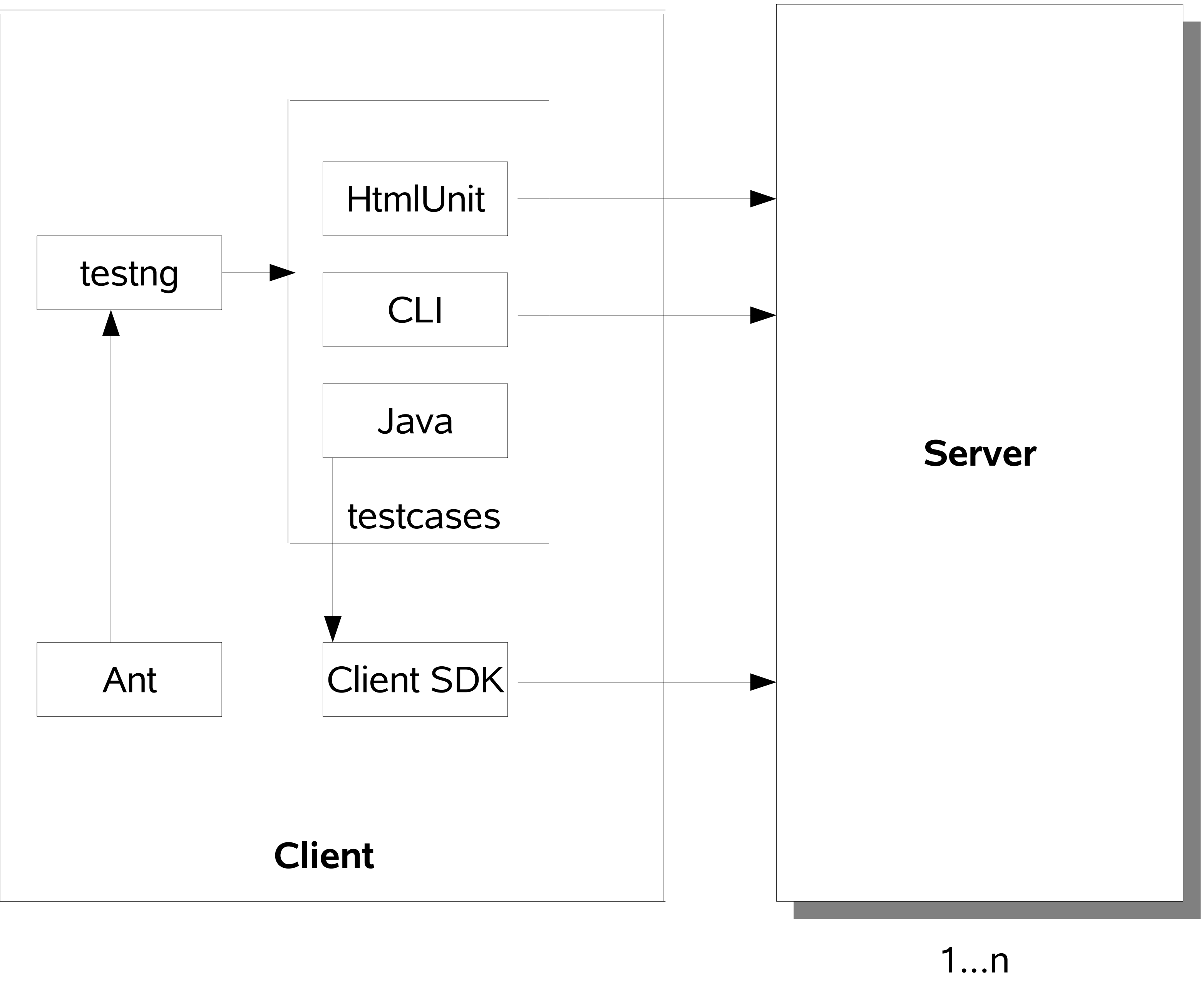
The command line (CLI) interface tests can be executed from a remote host as long as that remote host has access to the bootstrap file of the host where the OpenSSO application is configured.

The data for this framework is read from a set of properties files, these files can be customized as per the requirement of the underlying deployment. Since it is written using the platform neutral tools the framework can be used on any system with supported JDK version and the associated tools as described in section 3.0.

The tests in this framework can be executed on a pre-existing already configured server or a brand new server that has been already deployed but not configured. The configurator code that is in the framework will try to configure if not already configured. If configured then tests will be executed. Refer the control flow depicted in the diagram[--]

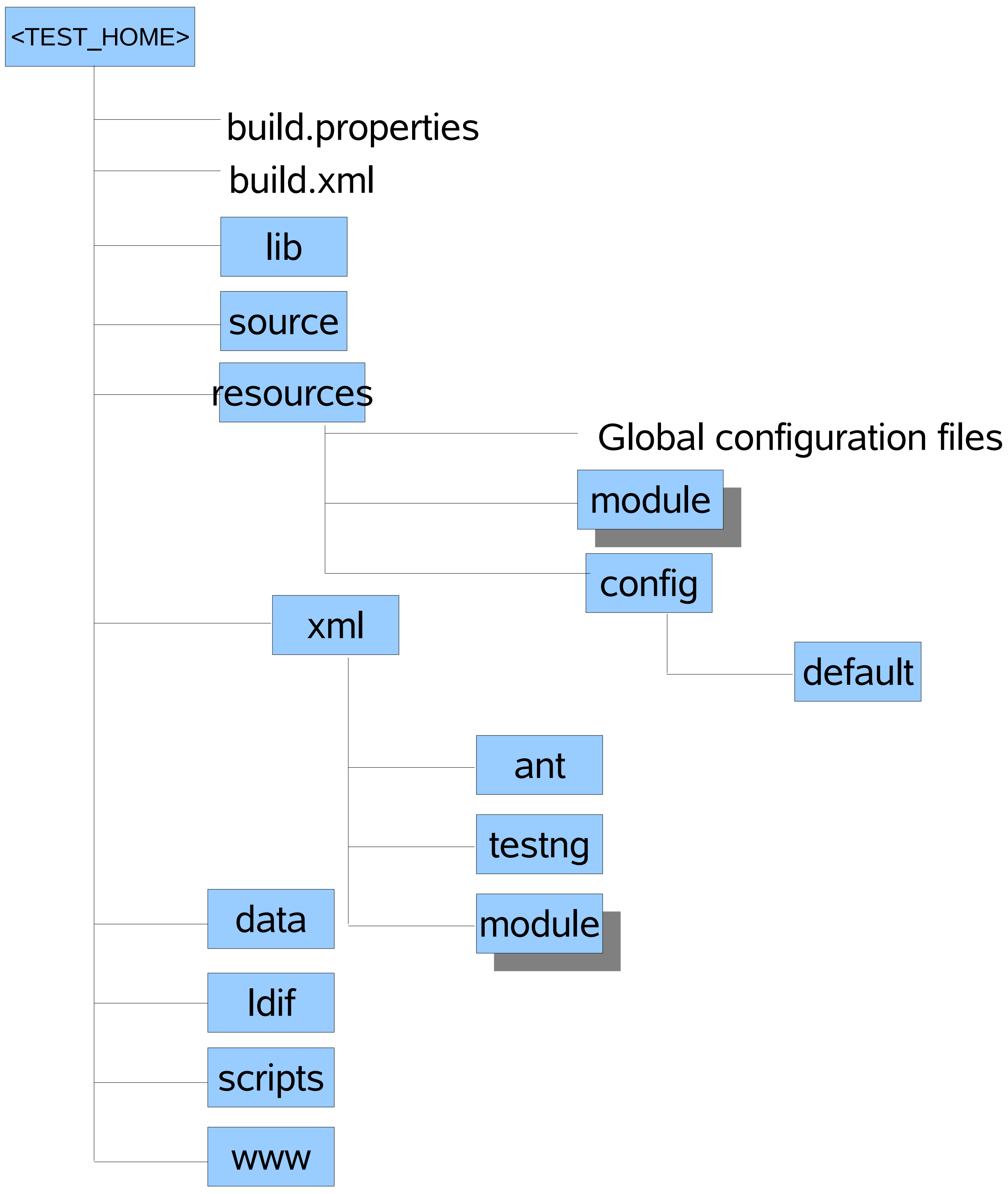
To execute the framework from the remote clients, the framework needs only the openssoclientsdk.jar based on the the type of tests selected to run. No other server side class libraries are required. For the CLI tests it is assumed the *famadm* utility is properly configured in the OpenSSO system.

One of the key requirement of any automation framework is to provide sufficient debug logs about the trace of the whole execution. This framework leverages JDK logging feature to provide granular logging of the execution trace. The consumer of this framework can customize the logging level that is desired to the execution scenario. Once the execution is completed this framework creates a very detailed report in the pre configured report directory. One can also customize them into emailable reports.



5.0 Structure of Test Suites

The test suites in this framework are organized in way that related functional features are grouped together to make it manageable. Most of the tests in this framework represent a use case scenario of the product being tested. Hence potentially a single use case test or scenario could subsume more than one functional test case of the product. Grouping logical scenarios in to test suite make sense to measure the test coverage of the feature. Further it provides a means to execute only selected set of test suite, in case of quick validation of a particular fix in the specific product functional area. Test scenario definition is independent of test execution. This means test scenario defines what needs to be tested and not how its tested. One can have different ways to perform the test such as using Java standalone application, command line tools or Administration console. The test suites are well organized in a directory structure that is mandated by maven guidelines. Here is how the directory structure looks like:



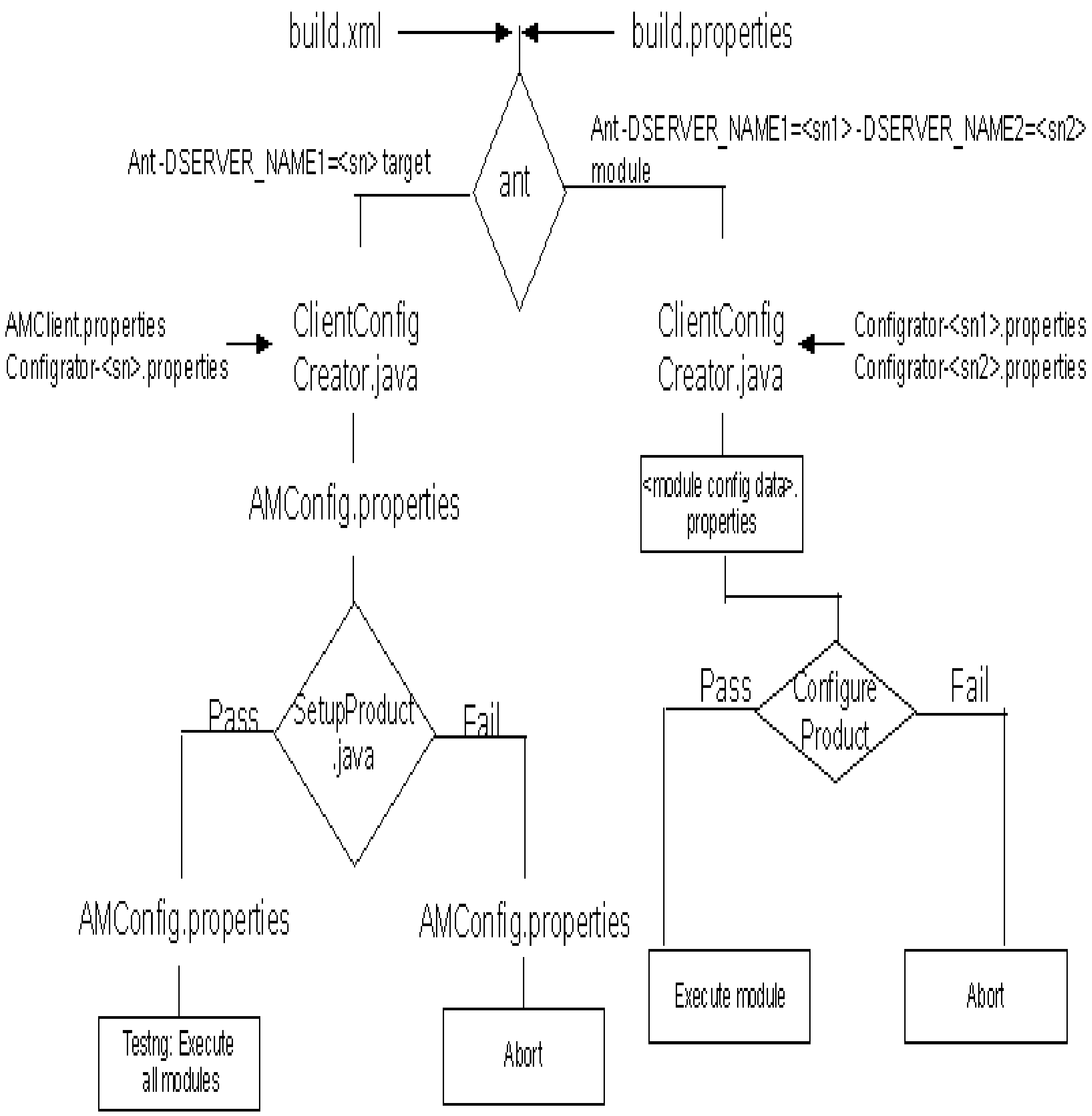
Directory Name	***Description***
QATEST_HOME	This is the location where the qatest framework is checked out from the opensso/openfm workspace
<*QATEST_HOME*>/xml/ant	ANT related configuration files such as opensso-properties.xml
<*QATEST_HOME*>/xml/testng	TestNG harness related files
<*QATEST_HOME*>/xml/<*test module*>	Contains XML files required by the individual test modules
<*QATEST_HOME*>/resources	This location contains all global properties files
<*QATEST_HOME*>/resources/config	Contains configuration files for global configuration like Datastores, authentication, policy etc which need to be configured at root realm
<*QATEST_HOME*>/resources/config/default	Contains default data for configuration files used for global configuration like Datastores, authentication, policy etc which need to be configured at root realm or sub realm
<*QATEST_HOME*>/resources/<*module name*>	This is the location to place all module specific configuration data. These files may have tags which are swapped at run time. These tag swapped files are placed under <*QATEST_HOME*>/<*SERVER_NAME*>/built/classes/<module name>
<*QATEST_HOME*>/<*SERVER_NAME*>/built/classes	All the Java byte code files are placed in this directory. This directory is also the locations where temporary files like password files are created and removed prior to the end of test harness exits. This also contains the dynamically generated server and AMConfig.properties resource bundles. These files have -Generated suffix.
<*QATEST_HOME*>/<*SERVER_NAME*>/built/classes/<module name>	Contains all the properties files used by this module. This includes files which are tag swapped at run time. This is the place to look to find what data was used at run time by this module for execution
<REPORT_DIR>	This directory contains all the generated reports with the following convention: <REPORT_DIR>/<SERVER_NAME>/<EXECUTION_MODE>/<TIME_STAMP>
<REPORT_DIR>/<SERVER_NAME>/<EXECUTION_MODE>/<TIME_STAMP>_<MODULE_NAME>/logs	This file contains all the output from the debug statements in the code base.

Directory Name	***Description***
<REPORT_DIR>/<SERVER_NAME>/<EXECUTION_MODE>/<TIME_STAMP>_<MODULE_NAME>/<EXECUTION_MODE>.output	Output file contains all the output from the ant execution. It captures all the output on the console for all executed test cases. For module specific execution this is of the form <module name>.output and for all mode its all.output
<REPORT_DIR>/<SERVER_NAME>/<EXECUTION_MODE>/<TIME_STAMP>_<MODULE_NAME>/properties	Copies all the resource bundles used in test execution
<REPORT_DIR>/<SERVER_NAME>/<EXECUTION_MODE>/<TIME_STAMP>_<MODULE_NAME>/debug	Contains all the client side debug logs
<REPORT_DIR>/<SERVER_NAME>/<EXECUTION_MODE>/<TIME_STAMP>_<MODULE_NAME>/xml	Copies all the xml files used or dynamically generated during test execution

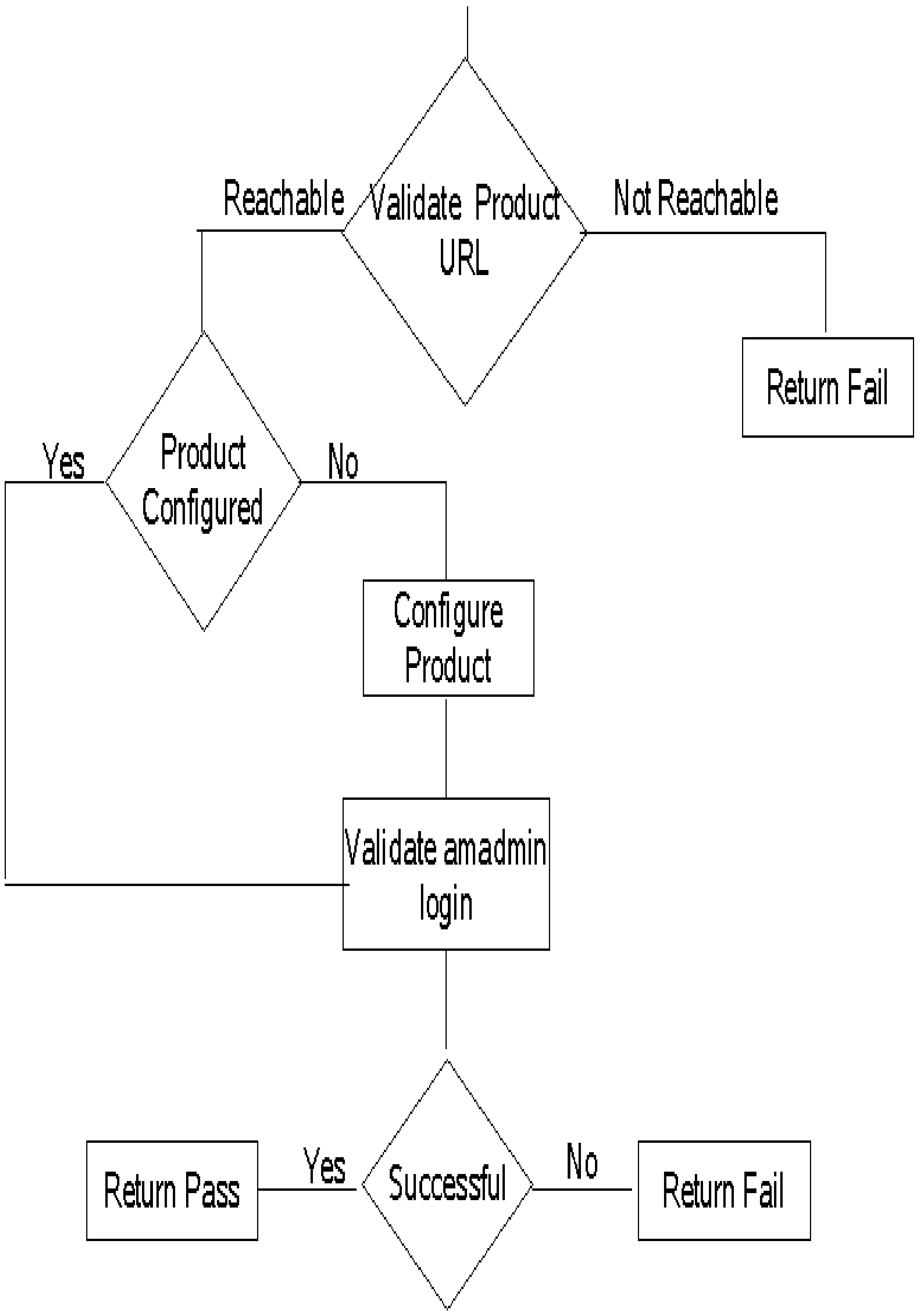
The *EXECUTION_MODE* corresponds to the install mode we are executing the tests on and the type of tests being executed. This will have values like server, serversmoke, clientsdk etc. From a TestNG perspective, these *EXECUTION_MODEs* map to groups. The TestNG xml files under the <QATEST_HOME>/xml/testng have the following naming convention: <EXECUTION_MODE>-<test module name>-testng.xml

6.0 Control Flow of the Framework Execution

As mentioned in the earlier section this framework can be launched against an already configured OpenSSO system or a system that has only the web archive of OpenSSO deployed on to a supported container. The following illustrations represent the control flow of the framework when invoked against the OpenSSO system.


build.xml
build.properties
ant
Ant -DSERVER_NAME1=<sn> target
Ant -DSERVER_NAME1=<sn1> -DSERVER_NAME2=<sn2> module
AMClient.properties
Configrator-<sn>.properties
ClientConfig Creator.java
ClientConfig Creator.java
Configrator-<sn1>.properties
Configrator-<sn2>.properties
AMConfig.properties
<module config data>. properties
Pass
SetupProduct .java
Fail
Pass
Configure Product
Fail
AMConfig.properties
AMConfig.properties
Execute module
Abort
Testng: Execute all modules
Abort

SetupProduct.java



6.1 How To Execute the Framework

You can execute the whole or part of the test suites by following the procedure described in this section. The framework has multiple options to make it more customizable and granular.

6.1.1 Checkout the Framework

Obtain the test framework by performing a CVS checkout from the OpenSSO repository, you can find the detailed procedure on how to checkout the OpenSSO source from the repository. Once you checkout the source the QA tests will be available under the directory ***<checkout dir>***/opensso/qatest.

6.1.2 Install Java SDK 1.5

The Framework acts as a client to the OpenSSO system.

6.1.3 Copy Client SDK libraries from the OpenSSO Server

At this step copy the openssoclientsdk.jar from the product build directory to <QATEST_HOME>/lib directory depending up on what kind of tests that you are planning to run.

6.1.4 Copy other libraries

qatest needs following additional jars to be copied under the <QATEST_HOME>/lib directory:

- jsse.jar (Get it from $JAVA_HOME/jre/lib)
- javaee.jar
- ldapjdk.jar
- saaj-api-jar
- saaj-impl.jar
- webservices-rt.jar
- testng-5.1-jdk15.jar (Get it from http://testng.org)
- HtmlIUnit jars – Download HtmlUnit from http://htmlunit.sourceforge.net/ and place all the jars under <QATEST_HOME>/lib/htmlunit
- xacml jars – Place all jars under <QATEST_HOME>/lib/xacml
  - jaxb-impl.jar
  - jaxb-libs.jar
- Jetty jars – Download jetty from http://www.mortbay.org/ and place the following jars under <QATEST_HOME>/lib/jetty

- ant-1.6.5.jar
- jetty-6.1.6rc0.jar
- jetty-util-6.1.6rc0.jar
- jsp-2.1.jar
- jsp-api-2.1.jar
- servlet-api-2.5-6.1.6rc0.jar
- tools.jar – Copy this from <JAVA_HOME>/lib. The jdk should be 1.5 or higher.

6.1.5 Configure the build.properties

Edit the build.properties located in the <QATEST_HOME> Directory. The table of configuration parameters that are required to be updated are as follows

Name of the Variable	***Description***
QATEST_HOME	This is the directory that contains the testcode checked out from the OpenSSO code base
EXECUTION_MODE	Denotes the kind of tests that we want to execute (ff, ds, ldapv3 etc) the list can go on; these correspond to TestNG groups and define logical grouping of test cases for different execution modes.
TEST_MODULE	To test an individual module, this variable has to be set to the specific module name.
REPORT_DIR	To write the TestNG logs and reports

6.1.6 Supported execution modes

1. ff_ds : Flatfile for User management & directory server as the backend repository

2. ds_ds : Directory server for user management and the backend repository

3. ff_ds_sec : Flatfile for user management & directory server as the backend repository with keystore configured

4. ds_ds_sec : Directory Server for user management and the backend repository with keystore configured

5. ff : Falt file as user repsoitory is no longer a supported configuration and will soon be depricated from qatest. Current only ds_ds and ds_ds_sec are supported.

6.1.7 Client side AMConfig Properties

- Multi server tests do not make use of any client side AMConfig.properties file
- Single server tests do need an AMConfig.properties file on the client side.
  - This file is DYNAMICALLY generated by qatest and end user DOES NOT need to provide one.
  - This file is generated using properties defined in resources/config/defualt/ ConfiguratorCommon.properties and Configuration-<server>.properties file.
  - ConfiguratorCommon.properties file lists minimum list of properties required on the client side and are considered to be invariant across all executions. It is not mandatory to make any changes to this file unless required.
  - Custom properties (server specific) are picked from Configuration-<server>.properties file. Its mandatory to specify and change all the values in this file according to your server configuration.

6.1.8 Server Configuration Properties

Server configuration properties are distributed in two files. The files are as follows:

- ConfiguratorCommon.properties – This file resides under <QATEST_HOME>/resources/ config/default and contains all the properties which don't need to changed very often and can be same across different servers.
- Configuration-<server name>.properties – This file resides under <QATEST_HOME>/resources and contains properties which are unique for each server. *All properties defined in this file are mandatory.*
- The name of server configuration file cannot contains "." . This file is used as a resource bundle and java cannot accept more than a single "." in resource bundle names. Hence one should not name the file with server FQDN.
- Configuration-<server name>-Generated.properties – This file is used at run time to

configure the server. This can be found under <QATEST_HOME>/<server name>/built/classes. This file is generated by merging the two files mentioned above, the ConfiguratorCommon.properties file and the Configuration-<server name>.properties. The logic is if a property exists in the Configuration-<server name>.properties, it takes precedence over the one mentioned in the ConfiguratorCommon.properties file. If not, one from ConfiguratorCommon.properties is taken. This helps user to configure the system by specifying a very small number of properties and at the same time keep the flexibility to uniquely specify any number of properties.

Properties defined in Configuration-<server>.properties

Configuration Property Name	***Description***
cookiedomain	The domain name to which the SSO Token is set, typically the DNS name of the servers. If the machine name is abc.xxx.yyy.com then .yyy.com is the root domain and value should be set to this.
com.iplanet.am.naming.url	Naming Service URL of the OpenSSO server, though the server may not yet be setup but this value is decomposed to obtain the server deploy URI,port and hostname.
config_dir	The configuration directory for server. This is used only at the time of configuring the war.
amadmin_password	Password for the user defined in amadmin_user
com.iplanet.am.service.password	The password to set for UrlAccessAgent user. This should be different from the amadmin password. This property is utilized both at the time of configuring the deployed war and while running the qatest framework, this password is being used to obtain the app ssotoken. This overrides the AMConfig.properties generated by the clientsdksamples module. Counter part to this property is com.sun.identity.agents.app.username
Following five properties list the details of the directory server, where services or configuration (SM) data will be stored. These are used, both for embedded and remote directory. For embedded, only directory_port and config_root_suffix are used. For remote directory all the properties are used. These are used only at the time of configuring the war.	
directory_server	Directory Server FQDN
directory_port	Directory Server port
config_root_suffix	Configuration root suffix
ds_dirmgrpasswd	The password for the variable ds_dirmgrdn

Configuration Property Name	*Description*
load_ums	A boolean value used to instruct the configurator whether to load the user management schema in to the directory (right now only applicable to Sun Java DS)

Properties defined in ConfiguratorCommon.properties file

Configuration Property Name	*Description*
notification_sleep	The number of milliseconds client calls should wait for notification from server to dirty the client side cache. The minimum should be set to at least 5000 (5 seconds).
am.encryption.pwd	The encryption key to use for server configuration and by IDM test to fill in the value of am.encryption.pwd in AMClient.properties.
log_level	Loglevel for the TestNG framework. The supported log levels are : SERVER, CONFIG, FINE and FINEST. For maximum debug logs, one should set this to FINEST.
com.iplanet.services.debug.level	Logging level for the client sdk
com.iplanet.am.cookie.name	Cookie name for the client sdk
amadmin_user	This is the top level administrative user name for the OpenSSO system
com.sun.identity.agents.app.username	app user name, this user is used to obtain the app ssotoken required by the client sdk to talk to the server. Any agent entity name is valid for this property
ds_dirmgrdn	This is the privileged user DN to connect to the LDAP server. This is for the directory server, where services data will be stored. This is used, both for embedded and remote directory. This is used only at the time of configuring the war.
realm	This is the default realm name used for the TestNG execution, individual module could override this value in the framework.
com.sun.identity.client.notification.url	The fqdn of client host on/from where client sdk calls are being made. '<protocol>://<fqdn>:<notification port>/<notification uri>'. This value auto generated by the qatest

Configuration Property Name	***Description***
umdatastore	User Management Datastore. This can take only two values: embedded or dirServer. If embedded it uses the default datastore configured when embedded configuration is selected. If dirServer is used, qatest behaviour is defined by values set in resources/config/default/UMGlobalConfig.properties file.This value is used each time qatest is executed.
datastore	Service/Config Management Datastore. This can take only two values: embedded or dirServer. If set to embedded, embedded OpenDS is used as the datastore. If set to dirServer, remote directory is used as the datastore and qatest behaviour is defined by values set in resources/Configurator-<server name>.properties file. This is used only at the time of configuring the war.
execution_realm	Execution realm against which the tests should be run (default is set to "/") this could be any valid realm. Currently only federation tests are supported with any realm.
subrealm_recursive_delete	This flag indicates if the execution realm tree should be deleted recursively or not. If execution_realm is set to /sub1/sub2/sub3 and this flag is set to true then /sub1 realm will be deleted recursively. But if this flag is set to false then only /sub3 realm will be deleted
com.iplanet.am.session.client.polling.enable	If polling is enabled for client sdk or not
com.iplanet.am.session.client.polling.period	Polling interval for session
com.sun.identity.agents.polling.interval	Agents polling interval in sec
com.sun.identity.idm.cache.enabled	Enable client side idm caching or not.
com.sun.identity.policy.client.cacheMode	Cache mode for policy on client sdk
com.sun.identity.agents.logging.level	Whether to enable logging for agents or not
com.sun.identity.agents.server.log.file.name	File name used to log agents log
com.iplanet.am.sdk.package	SDK package
com.iplanet.am.serverMode	Server mode to false indicates it is client side application.
com.iplanet.services.debug.directory	Debug directory on the server for debug logs
com.iplanet.am.defaultOrg	Name of default root organization/realm

Configuration Property Name	*Description*
com.sun.identity.monitoring	Enable or disable system monitoring
dist_auth_enabled	Will distributed authentication be used in the installation under test. Value should be "true" if distributed authentication is used and "false" if not.
dist_auth_notification_service	The notification URL of the distributed authentication deployment. This value should have the following form: '<da-protocol>://<da-fqdn>:<da-port>/<da-deploy-uri>/notificationservice This value is only used if dist_auth_enabled is set to "true".
Properties utilized for coupled configurations like saml, idff, multi protocol etc. By default the values for these properties are set using the values listed above but they can be changed to whatever the end user desires. User defined values will override the default values. These properties do not become part of client side AMConfig.properties file. This is becuase multiserver tests do not use client sdk to talk to the server. Currently that communication happens through famadm.jsp.	
metaalias	Meta alias to use when configurating the metadata. By default it is set to FQDN of the host name
entity_name	The name of entity to be created. By default it is set to FQDN of the host name
cot	Name of circle of trust. By default it is set as spcot for SP and idpcot for IDP
certalias	Certificate alias used for XML signing and encryption.
com.iplanet.am.server.protocol	Protocol for SP or IDP. By default the protcocl is picked from the namingservice attribute.
com.iplanet.am.server.host	Host name for SP or IDP. By default the protcocl is picked from the namingservice attribute.
com.iplanet.am.server.port	Port for SP or IDP. By default the protcocl is picked from the namingservice attribute.
com.iplanet.am.services.deploymentDescriptor	Deployment URI for SP or IDP. By default the protcocl is picked from the namingservice attribute.
internal.webapp.uri	It is used for deploying web application which contains html files. These url's are used by different modules. This is a web app internal to qatest and is deployed on embedded jetty server at run time. This prevents dependency on any external url's.

Configuration Property Name	*Description*
multiprotocol_enabled	If this flag is set to true, multiprotocol test are enabled. These tests require four setups which one instance serving as IDP and the rest three serving as SP's talking to IDP using different protocols.
idff_sp	SP talking through IDFF protocol (multiprotocol scenario only)
wsfed_sp	SP talking through WS-FED protocol (multiprotocol scenario only)
Properties (following six)utilized by tests which use keystore configuration (WSS). These properties become part of AMConfig.properties and used by client sdk. These are NOT MANDATORY.	
com.sun.identity.saml.xmlsig.keystore	Location of the keystore file.
com.sun.identity.saml.xmlsig.keypass	Location of keypass file.
com.sun.identity.saml.xmlsig.storepass	Location of storepass file.
com.sun.identity.saml.xmlsig.certalias	Certificate alias used for XML signing and encryption.
com.sun.identity.liberty.ws.wsc.certalias	Default certificate alias for issuing web service security token for this web service client
com.sun.identity.liberty.ws.soap.certalias	
com.sun.identity.idm.cache.enabled	Enable client side idm caching or not. This should be set to false for web services security tests.
com.sun.identity.saml.checkcert	Flag for checking the Certificate which is embedded in the KeyInfo against the certificates in the keystore (specified by the "com.sun.identity.saml.xmlsig.keystore" property). Possible values for the key are: on|off. If the flag is "on", the certification must be presented in the keystore for XML signature validation. If the flag is "off", skip the presence checking.
com.sun.identity.saml.xmlsig.xmlSigAlgorithm	XML signature algorithm. Used for SAML XML Signature generation and verification. When not specified, or value is empty, default value will be used.
com.sun.identity.saml2.crl.check	SAML2 XML Signing Certificate Validation
com.sun.identity.saml2.crl.check.ca	SAML2 XML Signing Certificate Validation
productSetupResult	Flag used internally by qatest. Not to be changed.
com.iplanet.security.SecureRandomFactoryImpl	Factory class name for SecureRandomFactory
com.iplanet.security.encryptor	The encrypting class implementation

Configuration Property Name	*Description*
com.iplanet.security.SSLSocketFactoryImpl	Default value is `com.iplanet.services.ldap.JSSSocketFactory`. Specifies the factory class name for `LDAPSocketFactory`. Available classes are: `com.iplanet.services.ldap.JSSSocketFactory` which uses JSS, and `netscape.ldap.factory.JSSESocketFactory` which uses pure Java
com.sun.identity.policy.client.resourceComparators	ResourceComparators to be used for different service names
com.sun.identity.liberty.ws.security.TokenProviderImpl	Specifies implementation for security token provider
com.sun.identity.saml.xmlsig.c14nMethod	
com.sun.identity.saml.xmlsig.transformAlg	
com.sun.identity.saml2.xmlenc.EncryptionProvider	
com.sun.identity.saml2.xmlsig.SignatureProvider	
com.sun.identity.liberty.ws.soap.staleTimeLimit	If the message timestamp is before current timestamp by this amount (millisec), it is considered a stale message
com.sun.identity.liberty.ws.soap.messageIDCacheCleanupInterval	All the messageID of a valid message will be stored in a cache with the it is received to avoid duplicate messages. If the current time minus the received time is greater than the above staleTimeLimit, it should be removed from the cache. The is property specify the interval(millisec) that a cleanup thread should check the cache and remove those messageID.
com.sun.identity.liberty.ws.soap.supportedActors	Supported SOAP actors. Each actor must be seperated by '|'
com.sun.identity.liberty.ws.jaxb.namespacePrefixMappingList	Namespace prefix mapping used when marshalling a JAXB content tree to a DOM tree. The syntax is* <prefix>=<namespace>|<prefix>=<namespace>|..........

6.1.9 User config datastore configuration inheritance model

OpenSSO supports multiple user datastores in realm and sub realm. The number of properties which can be specified to create and configure a datastore are large and can be painful on part of end user to specify and remember all of them. Hence qatest supports an inheritance model to reduce the data entry by end user. The model works as follows:

- All the possible properties with default values are defined in a global master file. This file resides under <QATEST_HOME>/resources/config/default and its name is UMGlobalDatastoreConfig.properties. This file lists all the properties for each supported datastore type. More properties can be added and existing ones can be modified. The thing to remember is that this is the master file and everything mentioned here is applicable for all datastore created using qatest, whether its at realm or sub realm level.

- A file with the same name resides under <QATEST_HOME>/resources/config and can also reside under any module properties directory which want to configure its own datastore (for eg <QATEST_HOME>/resources/config/idm). The file residing under <QATEST_HOME>/resources/config is mandatory and is used to configure datastore at the root realm level. It specifies minimal data required from end user to configure datastore under root realm.

- At run time, property values are merged between the master file (<QATEST_HOME>/resources/config/default/ UMGlobalDatastoreConfig.properties) and file under <QATEST_HOME>/resources/config/UMGlobalDatastoreConfig.properties and between the master file (<QATEST_HOME>/resources/config/default/ UMGlobalDatastoreConfig.properties) and file under (if any) module <QATEST_HOME>/resources/<module>/UMGlobalDatastoreConfig.properties. Properties mentioned in the <QATEST_HOME>/resources/config/UMGlobalDatastoreConfig.properties and <QATEST_HOME>/resources/<module>/UMGlobalDatastoreConfig.properties take precedence over the values mentioned in the master file. Any value not mentioned in the realm level or module level file is taken as is from the master file.

- The above merge leads to generation of a UMGlobalDatastoreConfig-Generated.properties file, which is used to run time to configure the datastore. This is placed under <QATEST_HOME>/<server name>/built/classes for realm level and <QATEST_HOME>/<server name>/built/classes/<module> for module level datastore configuration.

- qatest also allows to control the datastore creation and deletion behavior. These attributes can be set in <QATEST_HOME>/resources/config/default/UMGlobalConfig.properties. This includes:

    - To delete existing datastores or not. If set to true, all existing datastores, prior to executing qatest will be deleted.

    - To delete qatest created datastores or not. If set to true all datastores created by qatest will be deleted at the end of executions

- qatest does not restore any deleted datastores
- Currently this feature is only supported for root realm

6.1.10 Determine Single or Multi Server tests

All the testcases in this framework can be broadly classified in to two categories,

- Tests that are executed in a single server environment
- Tests that inherently require more than one server instance
- As the framework evolves there would be more tests added in to each of the category.
- Currently available modules
  - Sinlge Server
    - agents
    - authentication
    - cli
    - clientsamples
    - delegation
    - idm
    - notification
    - policy
    - session
    - xacml
  - Multi server
    - idff (sec and non sec mode; can be execuetd for realm and sunrealm)
    - multiprotocol (sec mode only)
    - samlv2 (sec and non sec mode)
    - samlv2idpproxy (sec and non sec mode)
    - wsfed (sec mode only)
    - wss (sec mode only)

The test framework is flexible enough to enable the end user to select specific module or all of the modules together to execute against the given server(s). Arbitration of this can be achieved by using the configuration file and the JVM parameter for the ANT tool.

6.1.11 Invoking Single Server Tests

To execute the testcases in single server environment, customize the build.properties as discussed in the earlier sections of this document. This executes all the single server test cases for the execution mode set in the properties file. Single Server tests take their execution configuration data from a single file under the resources directory and the file name is of the format Configurator-<***server name***>.properties. Tests cane be invoked from ANT tool with proper Java VM parameters.

Typically some thing like this

```
ant -DSERVER_NAME1=<server name>  <all|module>
Where
        module – Denotes the specific module that needs to be executed
        All – Execute all the testcases
```

6.1.12 Invoking Multi Server tests

The multi server tests comprises of test cases which require multiple server installations for the successful test execution and validation. It includes modules like samlv2, idff, id-wsf etc. Like the single server case the configuration data is read from the properties files that reside under the <***QATEST_HOME***>/resources directory, these parameters are global hence common across all the test modules.

The format of the properties file name is Configurator-<***server name1***>.properties and Configurator-<***server name2***>.properties. These properties file contain the basic installation and configuration parameters such as the encryption key,admin user name and password etc. For more details refer the table below.

Multiserver tests can be invoked through the ANT tool by using the following command syntax

```
ant    -DSERVER_NAME1=<opensso-server  name-1>  -DSERVER_NAME2=<opensso-server
```

```
name-2> module
Where
        module - Denotes the specific module that needs to be executed
```

6.1.13 How to name the Global configuration files

Global configuration files has to be named in such a way that the ANT targets match with them. For example if you invoke the ANT

```
ant -DSERVER_NAME1=test all
```

then the configuration file must be named as Configurator-test.properties. If this naming convention is not followed then the framework will immediately exit with a message saying Configurator-<name>.properties not found.

Each and every module in this framework has a set of properties file associated with it. These files contain he configuration data required to execute these modules. This might have to be changed before particular module can be executed successfully. For more details on what needs to be changed, please read the configuration readme that comes with those specific modules.

Once all the configuration data has been defined, you can execute the framework. To see a list of available targets, please execute the following command from the <*QATEST_HOME*> directory.

```
ant usage
```

6.1.14 Location of various logs

The compiled classes are placed under

- <*QATEST_HOME*>/<server name>/built for single sever execution
- <*QATEST_HOME*>/<server name1>_<server name2>/built for multi sever execution

The reports are placed under:

- <*REPORT_DIR*>/<server name> for single sever execution
- <*REPORT_DIR*>/<server name1>_<server name2>/built for multi sever execution

- All the debugging output goes in the logs file under the *REPORT_DIR*
- All ***ant*** execution output goes in the <execution mode>.output or <module name>.output file under the *REPORT_DIR*
- All the TestNG generated reports reside under the *REPORT_DIR*
- All the client side debug output related to OpenSSO goes in the debug directory under *REPORT_DIR*
- A file called ***test_env.txt*** is also created. This lists all the environment variables and parameters used for execution. This file is also available under REPORT_DIR

6.1.15 Parallel Remote Execution of the Framework

The framework is built in such a way the tests can be executed from a driver machine which will hold all the test framework configuration data , debug output logs and the test reports. This means the framework does not need to exist on the localhost where the Open SSO is running.

This framework can be invoked against multiple OpenSSO servers, all tests can be executed in this manner. For example one can simultaneously execute policy module on multiple J2EE containers that are configured with OpenSSO system. This provide a means to quickly validate the OpenSSO system on multiple containers.

To execute this framework simultaneously on multiple servers this is what needs to be done multiple servers at the same time. To do so:

1. Create a configuration file for each execution under the <*QATEST_HOME*>/resources directory
2. Begin the execution using ***ant*** for each server one after the another
3. Ensure there is a decent level of time delay between executions
4. This is valid for both single and multiple server tests
5. Single and multiple server tests can be mix and matched

7.0 Guidelines for Adding new test scenarios

One of the primary requirement for any test harness is the extensibility, that is one should be able to augment the existing framework by adding more and more test suites. At the same time these additions should not make the whole automation framework unmanageable. To achieve this certain guidelines has to be followed whenever new modifications are made to the framework.

OpenSSO QA team proposes the following guidelines for any one who is interested in contributing to this framework.

1. Test scenarios addition will be based on functional and systems level execution/testing requirement. Test scenarios can be executed using multiple tools (TestNG, cli/clu (if supported in the end product), httpunit or any other open source tool)

2. As far as TestNG is concerned, it is methods driven i.e. It collects a set of classes; looks for methods with @Test annotation in them; include and exclude these methods according to the defined filter; and execute them in parallel. It is a good practice to create dependency runnable methods so that we can exploit the power of TestNG. We can use the dependsOnMethods parameter to control the order execution if necessary

3. Add JavaDoc to the test cases so that test cases are maintainable.

4. All test suites should pass unless there is a known issue. In that case, those classes or methods can be excluded during test execution.

5. All files under following directories need to have following CDDL license displayed at the start:

- <*QATEST_HOME*>/source
- <*QATEST_HOME*>/resource
- <*QATEST_HOME*>/xml

you can find the latest copy of the License in the checked out workspace under opensso/legal directory:

```
<!--
The contents of this file are subject to the terms of the Common Development and Distribution License (the License). You may not use this file except in compliance with the License. You can obtain a copy of the License at https://OpenSSO.dev.java.net/public/CDDLv1.0.html or OpenSSO/legal/CDDLv1.0.txt See the License for the specific language governing permission and limitations under the License. When distributing Covered Code, include this CDDL Header Notice in each file and include the License file at OpenSSO/legal/CDDLv1.0.txt. If applicable, add the following below the CDDL Header, with the fields enclosed by brackets [] replaced by your own identifying information: "Portions Copyrighted [year] [name of copyright owner]" $Id Copyright 2007 Sun Microsystems Inc. All Rights Reserved
-->
```

8.0 Adding new test cases and test suites

Adding a new test suite is a easy process, following section describe the procedure on how to add a new test suite in to the test automation framework.

- Each test module resides in one package. For example all scenario tests related to policy are placed under policy package (<QATEST_HOME>/source/com/sun/identity/qatest/policy)
- Each scenario is self contained. Its has no dependency upon any other test scenarios. There is a setup and cleanup before and after actual test. After the tests execution from each module server's original state must be restored.
- Test cases execution data is defined in the properties files. These files are read at runtime by the executing programs.

8.1 Passing data to the framework

Currently framework is using two ways to pass the data from the properties files to executing programs:

8.1.1 Factory Model

Factory model is the concept promoted by the TestNG framework itself and is suitable for test implementations where each test cases has no dependence upon the other test cases, across the modules. ***This is not a supported model for creating tests in qatest and should not be used.***

8.1.2 TestNG xml file

Data passing through this model relies on test execution in the TestNG xml file, this is suitable for test implementations where test cases need to be executed in strict sequential manner.

8.2 Adding a test case to an existing module

It is quite possible in the development of a product to add new tests to an existing test suite this could be a regression test case or a testcase to test a new product enhancement fix. Here is how you can add a new test cases to the existing tests within the framework

1. Go to <QATEST_HOME>/source/com/sun/identity/qatest/<module name>
2. To add a new test case to an existing file, identify the code logic which needs to be implemented and the data required for execution
3. Open the file and add a new method. If the new tests require any new properties, they need to be defined in the properties file associated with this test module

8.2.1 Test the new changes Incrementally

It is a recommended practice to add and test the changes incrementally rather than dumping the whole changes in one single update. Having this practice makes it easy to debug as well as reviewing the code changes. To test the new changes:

1. Go to <QATEST_HOME> and change the module name in build.properties to the <module name>
2. Do *ant -DSERVER_NAME1=<server name> module* for single server or *ant -DSERVER_NAME1=<server name> -DSERVER_NAME2=<server name> module* for multi server.

   This will compile and execute only the module related tests. If there is a compilation error, system will abort at this point. If there are no compilation errors, module will be executed. Any run time error will require a look at either the configuration or the code logic. One can look at logs or output file for debugging info for any failures

8.2.2 Adding a new source file.

- Identify the code logic which needs to be implemented and the data required for execution
- Go to <QATEST_HOME>/source/com/sun/identity/<module>. Open a new file and code the test logic.
- Go to <QATEST_HOME>/resources/<module> and add a properties file <Class file name>.properties and add the properties required for execution

- Go to <QATEST_HOME>/xml/testng.
- If the execution is going to be governed through TestNG xml file, open the <execution mode>-<module name>-testng.xml file and add the newly added class to, either a new target or an existing target, depending upon the requirement.
- If the execution is going to utilize the factory model, ensure to implement a factory class for the implemented class. open the <execution mode>-<module name>-testng.xml file and add the new factory class.

To test the new changes follow the section 8.2.1

8.3 Add a new test module

This section addresses the procedure required to add a new test module in to this framework.

1. Go to <QATEST_HOME>/source/com/sun/identity/qatest and create a new directory. The name of the directory will be the name of you module. The package name for this new module should be the same as module name

2. Create a module testng.xml file (for TestNG) and place that under <QATEST_HOME>/xml/testng. The file name format is as follows: <execution mode>-<module name>-testng.xml. (e.g.:- server-idm-testng.xml, serverauthentication-testng.xml). This file contains TestNG execution semantics and module specific parameters.

3. Add any required resource bundles under <QATEST_HOME>/resources/<module name>.

4. Add any required xml files under <QATEST_HOME>/xml/<module name>

5. Follow the instructions under section 8.2 to add a new test to the new module

9.0 Executing the framework with a Secured Naming Service

1. Make sure the container is configured with proper certificates, manully access the container to make sure it is up
2. Extract the CA certificate of the web container in to /tmp/cacert.txt
3. Go to the client machine from where you are planning to kick off your QA tests, find the

JAVA_HOME that is being used for qatest.

4. *cd* to *$JAVA_HOME/jre/lib/security* and then run the following command

```
keytool -keystore cacerts  -keyalg RSA -import -trustcacerts -alias "myCA" -storepass changeit -file
~/cacert.txt
```

10.0 qatest Documents

All documents related to qatest reside under <QATEST_HOME>/www/public/docs/pdf. This includes overall qatest document and documents for each specific module.

- Overall document, OpenSSOTestFrameWork.pdf, describes the overall architecture of qates and how to setup qatest
- All other documents are module specific documents which describe how to configure, execute, trouble shoot and create or modify tests in that module.

APPENDIX A

A1 Sample *build.properties*

For readability removed the CDDL license header

```
#

# Property definitions.

#

# QATEST_HOME    : Home directory for qatest

# EXECUTION_MODE : This refers to what your user management and service
#                 management datstore are. The format is UM_SM. It can take
#                 one of the four values: ff_ds, ff_ds_sec, ds_ds and
#                 ds_ds_sec. _sec is used for cases where security is enabled.
#                 This includes both transport and message level security.
#                 qatest does not do any security configuration except for
#                 enabling security flags in the product.
# TEST_MODULE    : The module to test when executing tests for a single module
# REPORT_DIR     : Directory where all the reports and debug files will be
#                 created
-->
<project>
<property name="QATEST_HOME" value="/qatest/opensso/qatest"/>
    <property name="EXECUTION_MODE" value="ds_ds"/>
    <property name="TEST_MODULE" value="clientsamples"/>
    <property name="REPORT_DIR" value="/tmp"/>
</project>
```

APPENDIX B

B1 Sample Configurator-buffy-Generated.properties

```
com.sun.identity.saml.xmlsig.xmlSigAlgorithm=http://www.w3.org/2000/09/xmldsig#rsa-sha1
internal.webapp.uri=/internalwebapp
com.iplanet.am.sdk.package=com.iplanet.am.sdk.remote
cookiedomain=.iplanet.com
com.sun.identity.agents.logging.level=ALLOW
com.sun.identity.liberty.ws.wsc.certalias=test
com.iplanet.security.encryptor=com.iplanet.services.util.JCEEncryption

com.sun.identity.saml2.crl.check=false

com.sun.identity.agents.polling.interval=3

metaalias=@COPY_FROM_CONFIG@

com.sun.identity.saml.checkcert=off

com.iplanet.am.defaultOrg=@COPY_FROM_CONFIG@

com.sun.identity.saml.xmlsig.keypass=

com.iplanet.services.debug.directory=@BASE_DIR@/debug

com.sun.identity.saml2.xmlenc.EncryptionProvider=com.sun.identity.saml2.xmlenc.FMEncProvider

com.sun.identity.liberty.ws.soap.supportedActors=http://schemas.xmlsoap.org/soap/actor/next

com.sun.identity.idm.cache.enabled=false

com.sun.identity.liberty.ws.jaxb.namespacePrefixMappingList=S=http://schemas.xmlsoap.org/soap/
envelope/|sb=urn:liberty:sb:2003-08|pp=urn:liberty:id-sis-pp:2003-08|
ispp=http://www.sun.com/identity/liberty/pp|is=urn:liberty:is:2003-08

com.sun.identity.saml.xmlsig.certalias=test

idff_sp=golden16

dist_auth_notification_service=

certalias=test

com.iplanet.am.session.client.polling.period=180

com.sun.identity.liberty.ws.soap.staleTimeLimit=300000

multiprotocol_enabled=true

umdatastore=dirServer

com.iplanet.am.naming.url=http://orchid16.red.iplanet.com:8080/openfmemb/namingservice

productSetupResult=pass

com.iplanet.security.SecureRandomFactoryImpl=com.iplanet.am.util.SecureRandomFactoryImpl

realm=/

com.iplanet.services.debug.level=message

com.sun.identity.liberty.ws.security.TokenProviderImpl=com.sun.identity.liberty.ws.security.AMSecu
rityTokenProvider

com.sun.identity.saml.xmlsig.storepass=

com.sun.identity.agents.server.log.file.name=amRemotePolicyLog

log_level=FINEST

com.sun.identity.client.notification.url=@COPY_FROM_CONFIG@

am.encryption.pwd=FederatedAccessManagerEncryptionKeyfederatedaccessmanager

com.iplanet.am.cookie.name=iPlanetDirectoryPro

ds_dirmgrpasswd=secret12

com.sun.identity.policy.client.resourceComparators=serviceType=iPlanetAMWebAgentService|
class=com.sun.identity.policy.plugins.URLResourceName|wildcard=*|delimiter=/|caseSensitive=true
config_dir=/openfmemb
com.sun.identity.liberty.ws.soap.certalias=test
directory_port=50389
com.iplanet.am.server.port=@COPY_FROM_CONFIG@
dist_auth_enabled=false
com.sun.identity.saml.xmlsig.c14nMethod=http://www.w3.org/2001/10/xml-exc-c14n#
datastore=embedded
com.iplanet.security.SSLSocketFactoryImpl=netscape.ldap.factory.JSSESocketFactory
cot=@COPY_FROM_CONFIG@
com.sun.identity.saml.xmlsig.transformAlg=http://www.w3.org/2001/10/xml-exc-c14n#
com.sun.identity.saml.xmlsig.keystore=
load_ums=yes
subrealm_recursive_delete=false
execution_realm=/
com.iplanet.am.service.password=changeit
notification_sleep=5000
com.iplanet.am.session.client.polling.enable=true
com.iplanet.am.serverMode=false
ds_dirmgrdn=cn=Directory Manager
directory_server=orchid16.red.iplanet.com
com.sun.identity.monitoring=off
wsfed_sp=blackgate
amadmin_username=amadmin
com.sun.identity.agents.app.username=UrlAccessAgent
entity_name=@COPY_FROM_CONFIG@
com.iplanet.am.server.host=@COPY_FROM_CONFIG@
amadmin_password=secret12
com.sun.identity.saml2.xmlsig.SignatureProvider=com.sun.identity.saml2.xmlsig.FMSigProvider
com.sun.identity.liberty.ws.soap.messageIDCacheCleanupInterval=60000
com.iplanet.am.server.protocol=@COPY_FROM_CONFIG@
com.sun.identity.policy.client.cacheMode=subtree
com.sun.identity.saml2.crl.check.ca=false
config_root_suffix=dc=red,dc=iplanet,dc=com
com.iplanet.am.services.deploymentDescriptor=@COPY_FROM_CONFIG@
```