OpenSSO QA Test Automation

CLI Automated Testing Setup and Development

Prepared by Charles Wesley

1. Introduction

This document describes the following items:

- Requirements for executing the CLI testing module.
- Organization of tests in the CLI module.
- Execution of the CLI automated tests in the QATest framework.
- Reading the results report and debugging test failures.
- Debugging CLI automated test failures.
- Creating new CLI automated tests in the framework.

2. Requirements for Executing the CLI Module

Here are requirements for executing the CLI test module. These tests must be executed on the same machine where Federated Access Manager is deployed.

- The ant utility must be installed on the local machine.
- Deploy the fam.war or fam-samples.war must be deployed to a supported web container.
- Setup the CLI utilities.
    1. Create a directory under which the CLI utilities should reside. We will refer to this directory as “TOOLS_DIR”.
    2. Go to the directory created in step #1.
    3. Download or copy the famAdminTools.zip file to the directory created in step #1 on the local machine on which the CLI automated tests will be executed.
    4. Unzip famAdminTools.zip in TOOLS_DIR.

- Edit the build.properties file.
    1. Go to the OpenSSO workspace directory.
    2. Go to the “opensso/qatest” directory. We will refer to this directory as “TEST_HOME”.
    3. Edit the build.properties file.
        - Set the value of the “EXECUTION_MODE” property to one of the following values depending on the configuration store of the deployment under test:
          “ff_ds” - if using flat files as a user management store and directory server for service management.
          “ff_ds_sec” - if using flat files as a user management store and directory server for service management and security is enabled.
          “ds_ds” - if using directory server for both user and service management.
          “ds_ds_sec” - if using directory server for both user and service management and security is enabled.
        - Set the value of the “TEST_MODULE” property to “cli”.
        - Set the value of the “REPORT_DIR” property to the directory in which the test results should be stored.
- Edit the cliTest.properties file.
    1. Go to the OpenSSO workspace directory.

2. Go to the “<TEST_HOME>/resources/cli” directory.
3. Eidt the cliTest.properties file.
    - Set the value of the “cli-path” property to the directory in which famAdminTools.zip was unzipped (i.e. <TOOLS_DIR>).
    - If necessary, set the value of the “command-timeout” property to the number of maximum number of seconds the framework should wait for any CLI command to exit. The default value of 60 seconds should be adequate for systems with at least 1024 MB of RAM.
    - If a locale other than “en_US” is desired, set the value of the “locale” property to the locale value that the CLI should use.

- If using flat files as a configuration repository, edit the following properties in the CreateIdentityTest.properties file.
    1. createidentity-test6-message-to-find – Change the value “/fam80/openfm” to “<CONFIG_DIR>/<DEPLOY_URI>”.
    2. createidentity-test12-message-to-find -- Change the value “/fam80/openfm” to “<CONFIG_DIR>/<DEPLOY_URI>”.
    3. createidentity-test18-message-to-find -- Change the value “/fam80/openfm” to “<CONFIG_DIR>/<DEPLOY_URI>”.
- Create a Configurator-<hostname>.properties in the <TEST_HOME>/resources directory. Please refer to the QATest automation framework document for details on this process.

3. Organization of Tests

The existing CLI tests involve verification of the famadm CLI. There is one test class per famadm sub-command. Currently there are tests for the following sub-commands:

- create-auth-instance
- create-realm
- delete-auth-instance
- delete-realm
- delete-realm-attribute
- list-realms
- create-identity
- delete-identities
- get-auth-instance
- get-identity
- list-auth-instances
- list-identities
- add-member
- remove-member
- show-members
- show-memberships
- show-identity-operations
- show-identity-types
- add-realm-attributes
- delete-realm-attribute

- get-realm
- set-realm-attributes
- update-auth-instance

CLI framework directory structure:

1. <TEST_HOME>/xml/testng – This directory contains the TestNG XML files which determine the tests that will be executed and their order.
    2. ff_ds-cli-testng.xml – This TestNG XML file which determines the tests which will be executed for the "ff_ds" group or execution mode.
    - ff_ds_sec-cli-testng.xml – This TestNG XML file which determines the tests which will be executed for the "ff_ds_sec" group or execution mode.
    - ds_ds-cli-testng.xml -- This TestNG XML file which determines the tests which will be executed for the "ds_ds" group or execution mode.
    - ds_ds_sec-cli-testng.xml – This TestNG XML file which determines the tests which will be exeucted for the "ds_ds_sec" group or execution mode.
3. <TEST_HOME>/resources/cli
    - AddMemberTest.properties – The properties file for the AddMemberTest class.
    - AddRealmAttributesTest.properties – The properties file for the AddRealmAttributesTest class.
    - CreateAuthInstanceTest.properties – The properties file for the CreateAuthInstanceTest class.
    - CreateIdentityTest.properties – The properties file for the CreateIdentityTest class.
    - CreateRealmTest.properties – The properties file for the CreateRealmTest class.
    - DeleteAuthInstancesTest.properties – The properties file for the DeleteAuthInstancesTest class.
    - DeleteIdentitiesTest.properties – The properties file for the DeleteIdentitiesTest class.
    - DeleteRealmAttributeTest.properties – The properties file for the DeleteRealmAttributeTest class.
    - DeleteRealmTest.properties – The properties file for the DeleteRealmTest class.
    - GenericCLITest_ShowIdentityOperations.properties – The properties file for the show-identity-operations sub-command.
    - GenericCLITest_ShowIdentityTypes.properties – The properties file for the show-identity-types sub-command.
    - GetAuthInstanceTest.properties – The properties file for the GetAuthInstanceTest class.
    - GetIdentityTest.properties – The properties file for the GetIdentityTest class.
    - GetRealmTest.properties – The properties file for the GetRealmTest class.
    - ListAuthInstancesTest.properties – The properties file for the ListAuthInstancesTest class.
    - ListIdentitiesTest.properties – The properties file for the ListIdentitiesTest class.
    - ListRealmsTest.properties – The properties file for the ListRealmsTest class.
    - ShowMembersTest.properties – The properties file for the ShowMembersTest class.
    - ShowMembershipsTest.properties – The properties file for the ShowMembershipsTest class.
    - UpdateAuthInstanceTest.properties – The properties file for the UpdateAuthInstanceTest class. This file contains properties to perform a module based authentication for LDAP, Membership, NT, AD, and JDBC authentication

modules.

4. <TEST_HOME>/source/com/sun/identity/common/cli
    - FederationManagerCLI.java – A class which contains methods necessary to execute commands using the famadm CLI.
    - CLIUtility.java – A generic class which contains methods to perform common tasks for CLI's such as setting arguments, clearing arguments.
    - CLICommand.java – A class which pertains to the command being executed. This class contains methods to perform tasks such as execute a command, get the command output, get the command error.
    - StreamRedirector.java – A class which redirects the output or error of a CLICommand to a StringBuffer.
5. <TEST_HOME>/source/com/sun/identity/cli – A directory which contains the test classes for the CLI.
    - AddMemberTest.java
    - AddRealmAttributesTest.java
    - CLIConstants.java – An interface containing the famadm sub-commands strings.
    - ConfigureCLI.java – A class which verifies that FAM has been configured and configures the FAM administration tools (i.e. famadm).
    - CreateAuthInstanceTest.java
    - CreateIdentityTest.java
    - CreateRealmTest.java
    - DeleteAuthInstancesTest.java
    - DeleteIdentitiesTest.java
    - DeleteRealmAttributeTest.java
    - DeleteRealmTest.java
    - FederationManagerCLIConstants.java – An interface containing the arguments/options of the famadm CLI.
    - GenericCLITest.java
    - GetAuthInstanceTest.java
    - GetIdentityTest.java
    - GetRealmTest.java
    - GlobalConstants.java – An interface containing the global options/arguments to the CLI's (e.g. “help”, “verbose”).
    - ListAuthInstancesTest.java
    - ListRealmsTest.java
    - RemoveMemberTest.java
    - SetRealmAttributesTest.java
    - ShowMembersTest.java
    - ShowMembershipsTest.java
    - UpdateAuthInstanceTest.java

CLI test organization:

CLI tests are organized into four main groups.

1. ff_ds: Embedded store as the user management repository on the local machine and directory server for service management.
2. ff_ds_sec: Embedded store as the user management repository on the local machine and directory server for service management with security enabled.

3. ds_ds: Sun Directory Server as the user management and service management repositories.
4. ds_ds_sec: Sun Directory Server as the user management and service management repositories with security enabled.

4. Executing CLI automated tests

This section details the steps to execute the tests in the CLI test module.

- If only some of the CLI tests should be executed, edit the relevant TestNG XML file (xml/testng/ff_ds-cli-testng.xml, xml/testng/ff_ds_sec-cli-testng.xml, xml/testng/ds_ds-cli-testng.xml, and xml/testng/ds_ds_sec-cli-testng.xml) to include only the desired tests. The relevant TestNG XML file is one corresponding to the EXECUTION_MODE to be used. **Be sure to include the class "com.sun.identity.qatest.cli.ConfigureCLI" in the "<class>" tags in the XML for the first test case. See excerpt below ..**

```
<suite name="ff_ds-cli" verbose="1">
  <test name="CLI_create-realm01">
    <parameter name="testName" value="createrealm-test1"/>
    <groups>
      <run>
        <include name="ff_ds"/>
      </run>
    </groups>
    <classes>
      <class name="com.sun.identity.qatest.cli.ConfigureCLI"/>
      <class name="com.sun.identity.qatest.cli.CreateRealmTest"/>
    </classes>
  </test>
```

- Go to the <TEST_HOME> directory.
- Run the following command to execute the CLI module:
  ant -DSERVER_NAME1=<host> module

5. Interpreting the CLI Automated Testing Results

After execution of the CLI test module, the results will be located in the directory <REPORT_DIR>/<HOSTNAME>/<EXECUTION_MODE>/<EXECUTION_DATE_TIME>.

If possible, go to this directory location in a browser. This will display an HTML page of the overall results of the test execution with the following information:

- a link named "<EXECUTION_MODE>-cli" 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>-cli". 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 backgroup 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 see the values that were used in that particular test, click either the “Show outputs” link or the “Show all outputs” link. This will display the following information:

- the name of the test executed
- a brief description of the test's purpose
- a flag indicating whether the password file option to the CLI was used
- a flag indicating whether the debug option to the CLI was used
- a flag indicating whether the verbose option to the CLI was used
- a flag indicating whether long options to the CLI were used
- the message(s) which was (were) expected to be found in the CLI's output (tests with a zero exit status) or error (tests with a non-zero exit status)
- the exepected exit status of the CLI command
- other input values used in the specific command being tested

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.

6. Debugging CLI automated test failures

This section will present some common causes for failures in the CLI automated tests.

- One or more messages was not found in the command's output or error. Review the file logs which contains all the log file messages. Search for the name of the test which failed. Examine the output listed in the “Command output” or “Command error” fields for the command in question. Compare this output to the value of “message-to-find” and see if there are any inconsistencies or differences. Consider that there could be an error in the value of “message-to-find” in the properties file.
- The command has a different exit status than the expected exit status. Review the file logs which contains all the log file messages. Search for the name of the test which failed. Compare the value of “expected-exit-code” with the actual exit code displayed in “Command exit status”.
- Another CLI command used to verify the creation or deletion of an item failed to find the expected output. Verify that the command used for verification did not return a failed (non-zero) exit status. Examine the output for the verification command and see if there are any inconsistencies or errors.

7. Creating New CLI Automated Tests in the Framework

The execution of test cases associated with the existing CLI test classes (CreateRealmTest.java, DeleteRealmTest.java, ListRealmsTest.java, and CreateIdentityTest.java) is determined by the respective properties file (CreateRealmTest.properties, DeleteRealmTest.properties, ListRealmsTest.properties, and CreateIdentityTest.properties) and the relevant TestNG XML files for the CLI module. See section 3 of this document for more details on the location and purpose of these files.

Adding a new test case for one of these existing test classes consists of two main tasks:

1. Edit the associated properties file and create a new block of properties that will used by the new test case. See example below for creating a new test "createrealm-test17".

   - Copy a block of properties in the file (e.g. copy the properties for createrealm-test16).

     createrealm-test16-description=Create a sub-realm with a mixed case (capital and lower case letters) name.
     createrealm-test16-create-setup-realms=
     createrealm-test16-use-password-file=false
     createrealm-test16-use-verbose-option=false
     createrealm-test16-use-debug-option=false
     createrealm-test16-use-long-options=true
     createrealm-test16-message-to-find=Realm is created.
     createrealm-test16-create-realm=/MiXeDCaSeReAlM
     createrealm-test16-expected-exit-code=0

     createrealm-test16-description=Create a sub-realm with a mixed case (capital and lower case letters) name.
     createrealm-test16-create-setup-realms=
     createrealm-test16-use-password-file=false
     createrealm-test16-use-verbose-option=false
     createrealm-test16-use-debug-option=false
     createrealm-test16-use-long-options=true
     createrealm-test16-message-to-find=Realm is created.
     createrealm-test16-create-realm=/MiXeDCaSeReAlM
     createrealm-test16-expected-exit-code=0

   - Update the test case name for the new test. In this example, update "createrealm-test16" to "createrealm-test17".

     createrealm-test16-description=Create a sub-realm with a mixed case (capital and lower case letters) name.
     createrealm-test16-create-setup-realms=
     createrealm-test16-use-password-file=false
     createrealm-test16-use-verbose-option=false
     createrealm-test16-use-debug-option=false
     createrealm-test16-use-long-options=true
     createrealm-test16-message-to-find=Realm is created.
     createrealm-test16-create-realm=/MiXeDCaSeReAlM
     createrealm-test16-expected-exit-code=0

     createrealm-test**17**-description=Create a sub-realm with a mixed case (capital and

lower case letters) name.
createrealm-test**17**-create-setup-realms=
createrealm-test**17**-use-password-file=false
createrealm-test**17**-use-verbose-option=false
createrealm-test**17**-use-debug-option=false
createrealm-test**17**-use-long-options=true
createrealm-test**17**-message-to-find=Realm is created.
createrealm-test**17**-create-realm=/MiXeDCaSeReAlM
createrealm-test**17**-expected-exit-code=0

- Update the values of the properties for the new test "createrealm-test17" according to the comments in the beginning of the properties file.

```
createrealm-test17-description=A new test case.
createrealm-test17-create-setup-realms=
createrealm-test17-use-password-file=false
createrealm-test17-use-verbose-option=false
createrealm-test17-use-debug-option=false
createrealm-test17-use-long-options=true
createrealm-test17-message-to-find=Realm is created.
createrealm-test17-create-realm=/comma,realm
createrealm-test17-expected-exit-code=0
```

2. Add an XML block in the TestNG XML files (ds-local-cli-testng.xml, ff-local-cli-testng.xml, and ldapv3-local-testng.xml). Add the block only in those files related to the execution mode in which this test should be required (e.g. if a test is related to use with a directory server, add the block to the ds-local-cli-testng.xml and ldapv3-local-testng.xml files).

    - Copy an XML block from one "<test>" to the following "</test>". See an copied block in italics.

```
<test name="CreateRealm16">
   <parameter name="testName" value="createrealm-test16"/>
   <groups>
      <run>
         <include name="ff-local"/>
      </run>
   </groups>
   <classes>
      <class name="com.sun.identity.qatest.cli.CreateRealmTest"/>
   </classes>
</test>
<test name="CreateRealm16">
   <parameter name="testName" value="createrealm-test16"/>
   <groups>
      <run>
         <include name="ff-local"/>
      </run>
   </groups>
```

```
    <classes>
      <class name="com.sun.identity.qatest.cli.CreateRealmTest"/>
    </classes>
</test>
```

- Update the values for the test name values shown in bold.

```
<test name="CreateRealm16">
    <parameter name="testName" value="createrealm-test16"/>
    <groups>
      <run>
        <include name="ff-local"/>
      </run>
    </groups>
    <classes>
      <class name="com.sun.identity.qatest.cli.CreateRealmTest"/>
    </classes>
</test>
<test name="CreateRealm17">
    <parameter name="testName" value="createrealm-test17"/>
    <groups>
      <run>
        <include name="ff-local"/>
      </run>
    </groups>
    <classes>
      <class name="com.sun.identity.qatest.cli.CreateRealmTest"/>
    </classes>
</test>
```

Adding test cases for a new class:

Here are the steps for creating test cases with a new test class.

1. If all the arguments to the famadm CLI will be provided in the properties file (excluding arguments like "--adminid", "--password-file"), strings in the output or error should be matched, and the exit code is verified, then the GenericCLITest class may be used for these test cases.
2. If the GenericCLITest class can not be used, determine if the new class will need methods to be added to the com.sun.identity.qatest.common.cli.FederationManagerCLI class. It may be necessary to create methods in the FederationManagerCLI to execute the famadm command which would be verified in the test class.
3. Create a new test class with an appropriate name under the "<TEST_HOME>/source/com/sun/identity/qatest/cli" directory. This class would typically extend the class com.sun.identity.qatest.common.TestCommon. This class would also typically import the following TestNG classes: org.testng.annotations.AfterClass, org.testng.annotations.BeforeClass,

org.testng.annotations.Parameters, org.testng.annotations.Test, and org.testng.Reporter. The new test class should implement at least three methods: setup(), an appropriately named test method, and cleanup(). The setup method should accept the testName parameter as an input parameter. This setup method should create any items (e.g. realms, identities) that will be needed by the test method. The test class should use the assert statement in the test method to construct a list of one or more conditions which decide wheter a particular test case has passed or failed. The cleanup method should remove all the items that were created during the setup method.

4. A properties file with a name corresponding to the test class should be created under the <TEST_HOME>/resources/cli directory.
5. The CLI related TestNG XML files under <TEST_HOME>/xml/testng directory should be updated to include the XML needed for the new tests to be executed.