ALLSEEN ALLIANCE

AllJoyn™ Validation Test User Guide

March 21, 2014

Contents

1 Introduction

1.1 Purpose

This document provides a tutorial to understand how to write tests to verify that a device/application functionally behaves according to AllJoyn™ service framework interface specifications. It walks the user through each step to write a validation test using the AllJoyn validation framework and JUnit 3 framework. It also covers executing the AllJoyn Base Services validation test cases using the Maven or ADB tool.

1.2 Scope

This document is intended for software engineers and assumes familiarity with the AllJoyn SDK and Java programming language.

1.3 References

- *Validation Framework API Reference Manual*
- *Validation Test Case Specification Template*

1.4 Acronyms and terms

Term	Definition
AllJoyn service frameworks	A collection of full-feature implementations using the AllJoyn framework that provides specific functionality. These are building blocks that can be combined together to build interoperable devices and applications.
AllJoyn base services	This includes the Config, Control Panel, Notification, and Onboarding interfaces that support the relevant service frameworks.

2 Overview

The AllJoyn framework is open source software that allows for proximity peer to peer over various transports. The AllJoyn service frameworks are a collection of full-feature implementations using the AllJoyn framework that provide specific functionality. These are building blocks that can be combined together to build interoperable devices and applications.

The AllJoyn service framework implementations must conform to the published service framework interface specifications. This document goes over the steps to write tests to verify that an implementation behaves according to AllJoyn service framework interface specifications and the steps involved in executing the tests.

2.1 AllJoyn validation framework

The AllJoyn validation framework provides the set of Java interface definitions and classes required to write a test which can verify an AllJoyn service/interface implementation. It defines the API for plugging in the test cases/suites.

2.1.1 ValidationTestCase/ValidationBaseTestCase classes

- A class containing a test must extend *junit.framework.TestCase* and also implement *org.alljoyn.validation.framework.ValidationTestCase*.
- Optionally, the test case class can extend *org.alljoyn.validation.framework.ValidationBaseTestCase*.

ValidationBaseTestCase provides an implementation of methods specified in *ValidationTestCase*.

2.1.2 ValidationSuite/ValidationTest annotations

- The test case class must be annotated with *org.alljoyn.validation.framework.annotation.ValidationSuite* annotation to indicate that it's a test case class.
- The methods implementing the tests must be annotated with *org.alljoyn.validation.framework.annotation.ValidationTest* annotation.

The annotations provide the mapping from a particular test case class and method to the test suite name and test case name.

2.1.3 ValidationTestContext class

The *org.alljoyn.validation.framework.ValidationTestContext* class facilitates communication between a test case and the validation framework. A test case can retrieve test parameters from the *ValidationTestContext* as well as provide information about the test case to the *ValidationTestContext*.

3 Writing a sample test

This section details the steps for writing a sample test case class and adding the test to a sample test suite.

3.1 Test case class

1. Create a class that extends *org.alljoyn.validation.framework.ValidationBaseTestCase* and annotate the class with *org.alljoyn.validation.framework.annotation.ValidationSuite* annotation.
2. Specify a name for the testGroupId to be used when creating an *org.alljoyn.validation.framework.ValidationTestGroup* object.

   ```
   @ValidationSuite(name = "Sample-v1")
   public class SampleTestSuite extends ValidationBaseTestCase
   ```

   The expected naming convention for testGroupId includes the name of the service framework and its version (for example, Config-v1). This value must uniquely identify the test suite that includes one or more test cases.

   The test cases for a test suite can also be contained in multiple Java classes. In that case, all the classes for the test suite must contain this annotation and the same name.
3. Implement a *setUp()* method in which the parameters passed in to the test are read. Any steps that are common to all tests in the class should also be implemented in this method.

   Since JUnit 3 framework is used, the *setUp()* method is invoked before executing each test method.

   ```
   @Override
   protected void setUp()
   {
      super.setUp();
      testParameterValue =
      getValidationTestContext().getTestParameter(testParameterName);
      ....
      ....
   }
   ```
4. Create a *tearDown()* method to perform cleanup after a test completes execution.

   Since JUnit 3 framework is used, the *tearDown()* method is called after executing each test method.

   ```
   @Override
   protected void tearDown()
   {
      super.tearDown();
      ……
   ```

```
}
```

5. Implement the test method and annotate it with the *org.alljoyn.validation.framework.annotation.ValidationTest* annotation.
6. Specify a test method name to be used later when creating an *org.alljoyn.validation.framework.ValidationTestItem* object.

   The expected naming convention for test method name is the name of the service framework, its version, and a test case ID (for example, Config-v1-01). This must uniquely identify the test.

```
@ValidationTest(name = "Sample-v1-01")
public void testSample_v1_01_Sample()
{
      …....
}
```

   - If some information about the test execution needs to be captured, use *org.alljoyn.validation.framework.ValidationTestContext* to add a note in the test.

     The ValidationTestContext interface extends *org.alljoyn.validation.framework.TestCaseNoteListener* interface which defines the *addNote()* method.

```
getValidationTestContext().addNote("Information to be captured");
```

   - If the test requires some user interaction, use the *org.alljoyn.validation.framework.ValidationTestContext* object to wait for user input.

     The ValidationTestContext interface extends *org.alljoyn.validation.framework.UserInputHandler* interface which defines the *waitForUserInput()* method. The *org.alljoyn.validation.framework.UserInputDetails* class is used to interact with the user and the *org.alljoyn.validation.framework.UserResponse* object will contain results of the user operation.

```
@ValidationTest(name = "Sample-v1-01")
public void testSample_v1_01_Sample()
{
   …
   UserResponse userResponse =
      getUserResponse(getValidationTestContext());
   …
}

private UserResponse getUserResponse(UserInputHandler userInputHandler)
{
   String[] messageArray = { "Test Msg 1", "Test Msg 2", "Test Msg 3" };
   UserInputDetails userInputDetails = new UserInputDetails("Select the
      message(s) received", messageArray);

   return userInputHandler.waitForUserInput(userInputDetails);
}
```

3.2 Test suite class

1. Create a class that implements *org.alljoyn.validation.framework.ValidationTestSuite*.

   This class serves the following purposes:

   - It determines if the test cases present in a ValidationSuite should be run against a device based on the received About announcements.
   - It determines the set of test cases to be executed. One test suite class must be created for each ValidationSuite.

     ```
     public class SampleTestSuiteManager implements ValidationTestSuite
     ```

2. Implement the *getApplicableTests()* method. This method has to iterate over the About announcements from the device and return applicable test groups.

   ```
   @Override
   public List<ValidationTestGroup>
      getApplicableTests(AllJoynAnnouncedDevice allJoynAnnouncedDevice)
   {
      for (AboutAnnouncement aboutAnnouncement :
         allJoynAnnouncedDevice.getAnnouncements())
      {
         if (addTestGroupForApplication(aboutAnnouncement))
         {
            testGroups.add(createTestGroup(aboutAnnouncement));
         }
      }

      return testGroups;
   }
   ```

   - The addTestGroupForApplication() method should have the logic to determine if the test suite is applicable based on the About announcement details.
   - The createTestGroup() method should have the logic to create an org.alljoyn.validation.framework.ValidationTestGroup object.

     ```
     private ValidationTestGroup createTestGroup(AboutAnnouncement
     aboutAnnouncement)
     {
           Class<? extends ValidationTestCase> validationTestCaseClass =
                 SampleTestSuite.class;
           ValidationSuite validationSuite =
                 validationTestCaseClass.getAnnotation(ValidationSuite.class
                 );
           String testGroupId = validationSuite.name();
           ValidationTestGroup testGroup = new
                 ValidationTestGroup(testGroupId, aboutAnnouncement);
           addTestItemsToGroupFromAnnotations(testGroup,
                 validationTestCaseClass);

           return testGroup;
     }
     ```

NOTE

The addTestItemsToGroupFromAnnotations() method should take care of creating *org.alljoyn.validation.framework.ValidationTestItem* objects and adding them to *ValidationTestGroup* object.

```
private void addTestItemsToGroupFromAnnotations(ValidationTestGroup
      testGroup, Class<? extends ValidationTestCase>
      validationTestCaseClass)
{
      Method[] methods = validationTestCaseClass.getMethods();
      List<Method> testMethods = new ArrayList<Method>();

      for (Method method : methods)
      {
            ValidationTest validationTest =
                  method.getAnnotation(ValidationTest.class);
            Disabled disabled = method.getAnnotation(Disabled.class);

            if ((validationTest != null) && (disabled == null))
            {
                  testMethods.add(method);
            }
      }

      Collections.sort(testMethods, sampleTestComparator);

      for (Method testMethod : testMethods)
      {
            ValidationTestItem testItem = new
                  ValidationTestItem(validationTest.name(),
                  validationTestCaseClass.getName(),
                  testMethod.getName(), validationTest.timeout());
            testGroup.addTestItem(testItem);
      }
}
```

The *Collections.sort(testMethods, sampleTestComparator)* sequences the tests as per the values specified in the order field of *ValidationTest* annotation.

4 Useful techniques

This section provides helpful techniques for writing test cases.

4.1 Handling asynchronous events

The test case executes synchronously in a single thread. Sometimes a test case must wait for other events to occur before test execution can proceed. The following design provides a technique for waiting for something to happen, and retrieving details on what happened.

4.1.1 Create a handler class that will handle the event

The handler class will maintain an internal optionally bounded blocking queue that holds the events. The handler class must expose a public method that is invoked by the event publisher. When an event is received from the publisher, the event is added to the queue.

The handler class must also expose another method for event receivers to get the event. Whenever this method is invoked, the event is polled from the queue and returned to the receiver. The same instance of handler class is used by both the event publisher and event subscriber.

```
public class EventHandler
{
    private LinkedBlockingDeque<Object> eventQueue = new
        LinkedBlockingDeque<Object>();

    public void handleEvent(Object object)
    {
        eventQueue.add(object);
    }

    public Object waitForNextEvent(long timeout, TimeUnit unit) throws
        InterruptedException
    {
        return eventQueue.poll(timeout, unit);
    }
}
```

The *waitForNextEvent()* method will time out if no event is received within the timeout duration specified in the call. The reason for waiting only a specific length of time is that it allows the test case to fail sooner if the expected event doesn't occur in the expected time (versus waiting for the test case to time out).

If the expectation is that a particular event will happen within a particular time frame, the test case should be coded to wait only that long before having an assertion fail.

4.2 Determine if an AllJoyn service/interface is implemented

Create a utility method that will accept an AboutAnnouncement method and the AllJoyn feature interface name.

```
public boolean supportsInterface(AboutAnnouncement aboutAnnouncement,
    String expectedInterfaceName)
{
   boolean interfaceSupported = false;
   for (BusObjectDescription busObjectDescription :
      aboutAnnouncement.getObjectDescriptions())
   {
       for (String interfaceName : busObjectDescription.interfaces)
       {
          if (interfaceName.equals(expectedInterfaceName))
          {
             interfaceSupported = true;
             break;
          }
       }
   }

   return interfaceSupported;
}
```

5 Running AllJoyn™ Base Services validation tests

This section covers running the AllJoyn Base Services validation test cases.

5.1 Prerequisites

Table 1 lists the minimum software version requirements to execute the AllJoyn validation tests.

Table 1. Software requirements

Software	Minimum version requirement
Java SE Development Kit (JDK)	1.6
Android SDK	API/Platform 16
Apache Maven	3.1.1
Git (required to download code from the validation repository)	N/A

5.2 Download the AllJoyn validation test code

Use git to clone the validation repository to a local directory (referred to as $VAL_DIR).

```
$ git clone {remote path to the validation repository} $VAL_DIR
```

5.3 Install the AllJoyn library dependencies

1. Install the required Alljoyn jars and shared objects into your local Maven repository.

   A Maven project, *validation-dependencies*, has been provided to ease installing these dependencies. See the README.txt file in the *$VAL_DIR/java/components/validation-dependencies/libs* directory for a list of the required dependencies.

2. Copy the necessary dependencies into the lib directory. (They will be installed using the commands in Build the dependencies.)

5.4 Build the dependencies

Execute the following commands to build the necessary dependencies to run the tests.

```
$ cd $VAL_DIR/java/components/validation-base/HEAD
$ mvn clean install

$ cd $VAL_DIR/java/components/validation-dependencies/HEAD
$ mvn clean install
```

```
$ cd $VAL_DIR/java/components/validation-framework/HEAD
$ mvn clean install

$ cd $VAL_DIR/java/components/validation-tests/HEAD
$ mvn clean install
```

5.5 Test parameters

Table 2 lists the parameters for running the test cases. The appId and deviceId parameters are always required. The test cases to run can be specified by giving a list of the classes and optionally a list of the test case names.

These test parameters provide the layout you define to execute the test cases using Maven or ADB, detailed in Execute the validation tests.

Table 2. Test parameters

Parameter	Description
appId	Globally unique identifier for the application given by its About interface's AppId metadata field.
deviceId	Device identifier set by platform-specific means found in the About interface's DeviceId metadata field.
testSuiteList	Comma-separated list of the test suites to run (for example, org.alljoyn.validation.testing.suites.about.AboutTestSuite).
testCaseName	Comma-separated list of the test cases to run (for examlpe, About-v1-01, About-v1-02, About-v1-03).
enableInteractive	Determines if user interaction is required during test case execuction. Default value is "false". This parameter must be set to "true" for NotificationProducerTestSuite and NotificationConsumerTestSuite test cases.
userInputTimeoutValueInMS	Time in milliseconds to wait for a test to complete before considering it failed. This will override the default timeout value.
org.alljoyn.Onboarding.SoftApSsid	SSID for the device's Soft AP (required for OnboardingTestSuite test cases).
org.alljoyn.Onboarding.SoftApSecurity	Security type for the device's Soft AP (required for OnboardingTestSuite test cases). Must be WPA, WPA2, or NONE.
org.alljoyn.Onboarding.SoftApPassphrase	Passphrase for the device's Soft AP (if the Soft AP has security).
org.alljoyn.Onboarding.PersonalApSsid	SSID for the personal AP (required for OnboardingTestSuite test cases)
org.alljoyn.Onboarding.PersonalApSecurity	Security type for the personal AP (required for OnboardingTestSuite test cases). Must be WPA or WP2.

Parameter	Description
org.alljoyn.Onboarding.PersonalApPassphrase	Passphrase for the personal AP (required for OnboardingTestSuite test cases).

5.6 Execute the validation tests

The validation tests cases can be run as Android JUnit tests on an Android device. The test cases have been verified to run on a Nexus 7 tablet and may run on other AllJoyn devices running on Android platform 16 or higher.

The Device Under Test (DUT) must be on the same network as the test device. The validation-it module enables the test cases to be run against a particular AllJoyn application. The parameters for the test cases are provided as instrumentation arguments to the test apk.

To execute a particular set of tests, verify the test apk has been copied to the test device.

```
$ cd $VAL_DIR/java/components/validation-tests/HEAD/validation-tests-it
$ mvn android:deploy
```

5.6.1 Using Maven

1. Create a Maven profile in the pom.xml file with the appropriate instrumentation arguments.
2. Use the following command (referencing the created profile) from the validation-tests-it directory. The following command refers to the instrument-test profile.

   ```
   $ mvn android:instrument -Pinstrument-test
   ```

   The output of this command should be the results of running the specified set of tests as Android JUnit tests.

Below is an example Maven profile that would run the About-v1-01 and Config-v1-01 test cases.

```
<profile>
       <id>example-test</id>
       <build>
           <plugins>
               <plugin>
                <groupId>com.jayway.maven.plugins.android.generation2</groupId>
                <artifactId>android-maven-plugin</artifactId>
                <configuration>
                  <test>
                       <skip>false</skip>
                       <instrumentationArgs>
```

```
                        <instrumentationArg>appId 5b7a70f0-fd5b-49be-8e0a-
                        9640f49cd597</instrumentationArg>

                        <instrumentationArg>deviceId 5b7a70f0-fd5b-49be-8e0a-
                        9640f49cd597</instrumentationArg>

                        <instrumentationArg>testSuiteList
                        org.alljoyn.validation.testing.suites.about.AboutTestSui
                        te,org.alljoyn.validation.testing.suites.config.ConfigTe
                        stSuite</instrumentationArg>

                        <instrumentationArg>testCaseName About-v1-01,Config-v1-
                        01</instrumentationArg>

                    </instrumentationArgs>

                  </test>

                 </configuration>

                </plugin>

               </plugins>

        </build>

</profile>
```

5.6.2 Using ADB

Use the “instrument” command of Activity Manager (am) tool which is part of Android Debug Bridge (adb).

- The package name and test runner class must be provided along with the command. The package name is *org.alljoyn.validation.validation_tests.validation_tests_it*.
- The test runner class is *org.alljoyn.validation.testing.instrument.ValidationInstrumentationTestRunner*.

Provide the test parameters along with the command.

Below is an example command that would run the About-v1-01 and Config-v1-01 test cases.

```
$ adb shell am instrument -w -e appId 5b7a70f0-fd5b-49be-8e0a-9640f49cd597 -e
deviceId 5b7a70f0-fd5b-49be-8e0a-9640f49cd597 -e testSuiteList
org.alljoyn.validation.testing.suites.about.AboutTestSuite,org.alljoyn.validati
on.testing.suites.config.ConfigTestSuite -e testCaseName About-v1-01,Config-v1-
01
org.alljoyn.validation.validation_tests.validation_tests_it/org.alljoyn.validat
ion.testing.instrument.ValidationInstrumentationTestRunner
```