



SST: JUnit IdentityIQ Helper User Guide

Document Revision History

Revision Date	Written/Edited By	Comments
June 2017	Christian Cairney	Initial release with SSD v4

© Copyright 2017 SailPoint Technologies, Inc., All Rights Reserved.

SailPoint Technologies, Inc. makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. SailPoint Technologies shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Restricted Rights Legend. All rights are reserved. No part of this document may be photocopied, reproduced, or translated to another language without the prior written consent of SailPoint Technologies. The information contained in this document is subject to change without notice.

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 for DOD agencies, and subparagraphs (c) (1) and (c) (2) of the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 for other agencies.

Regulatory/Export Compliance. The export and reexport of this software is controlled for export purposes by the U.S. Government. By accepting this software and/or documentation, licensee agrees to comply with all U.S. and foreign export laws and regulations as they relate to software and related documentation. Licensee will not export or reexport outside the United States software or documentation, whether directly or indirectly, to any Prohibited Party and will not cause, approve or otherwise intentionally facilitate others in so doing. A Prohibited Party includes: a party in a U.S. embargoed country or country the United States has named as a supporter of international terrorism; a party involved in proliferation; a party identified by the U.S. Government as a Denied Party; a party named on the U.S. Government's Entities List; a party prohibited from participation in export or reexport transactions by a U.S. Government General Order; a party listed by the U.S. Government's Office of Foreign Assets Control as ineligible to participate in transactions subject to U.S. jurisdiction; or any party that licensee knows or has reason to know has violated or plans to violate U.S. or foreign export laws or regulations. Licensee shall ensure that each of its software users complies with U.S. and foreign export laws and regulations as they relate to software and related documentation.

Trademark Notices. Copyright © 2017 SailPoint Technologies, Inc. All rights reserved. SailPoint, the SailPoint logo, SailPoint IdentityIQ, and SailPoint Identity Analyzer are trademarks of SailPoint Technologies, Inc. and may not be used without the prior express written permission of SailPoint Technologies, Inc. All other trademarks shown herein are owned by the respective companies or persons indicated.

Table of Contents

Overview.....	4
Installation and Configuration	4
Eclipse Configuration.....	4
SailPointJUnitTestHelper	7
My first JUnit.....	8

Overview

The JUnit IdentityIQ Helper classes allow you to leverage the JUnit framework for unit testing an IdentityIQ implementation in an IDE. The helper classes allow the unit tester to easily access the SailPointContext and some of the functionality of the IdentityIQ Console to reduce the time taken to implement unit tests.

Installation and Configuration

No components are necessary to be deployed to the IdentityIQ instance, the JUnit and helper classes are meant to be deployed to your IDE of choice. This helper library has been tested using Eclipse Neon.

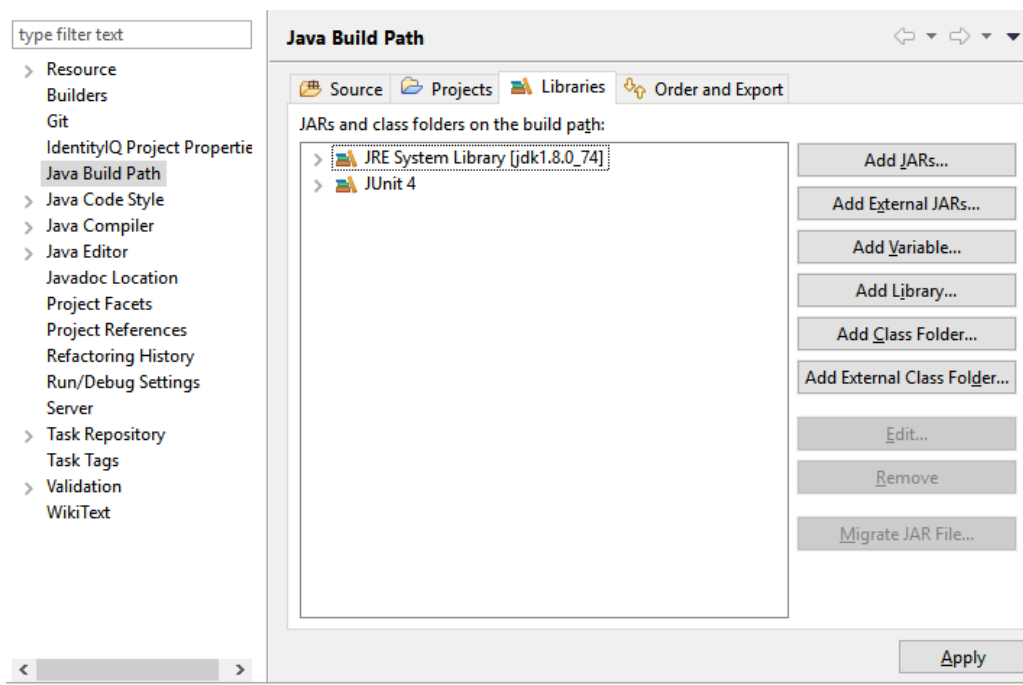
Filename	Description
IdentityIQJUnitHelper.jar	The JUnit helper classes

Eclipse Configuration

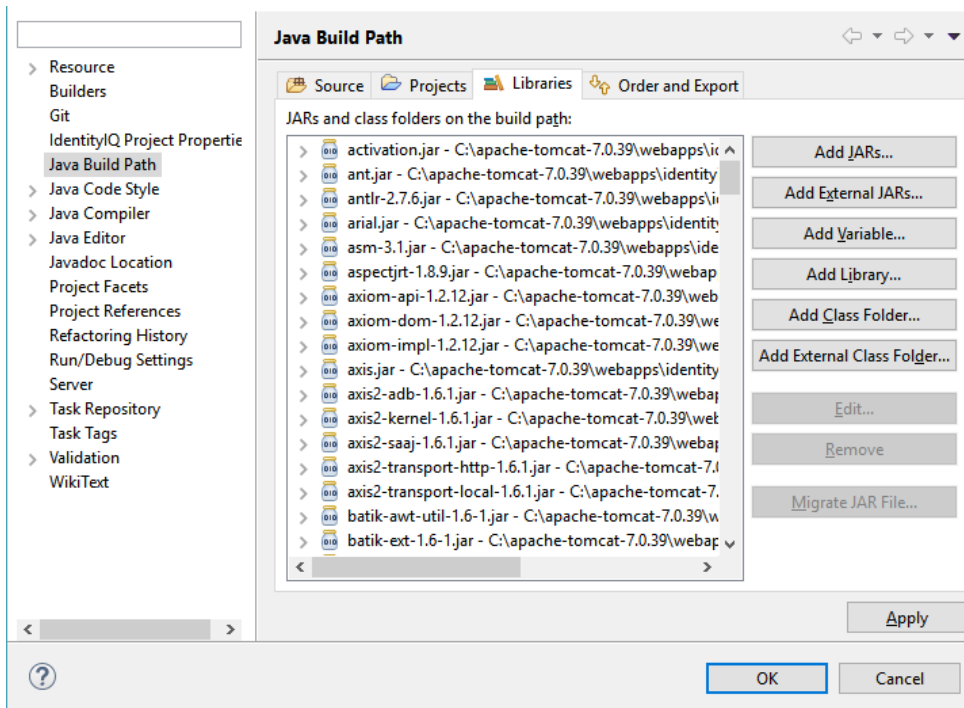
It is assumed that JUnit has already been configured in the Eclipse IDE. JUnit is usually installed as part of the J2EE version of Eclipse and should not require any further configuration.

The Eclipse class path and jar files will need to be configured to enable the helper classes; the following steps allow Eclipse to reference the IdentityIQ jars and database configuration.

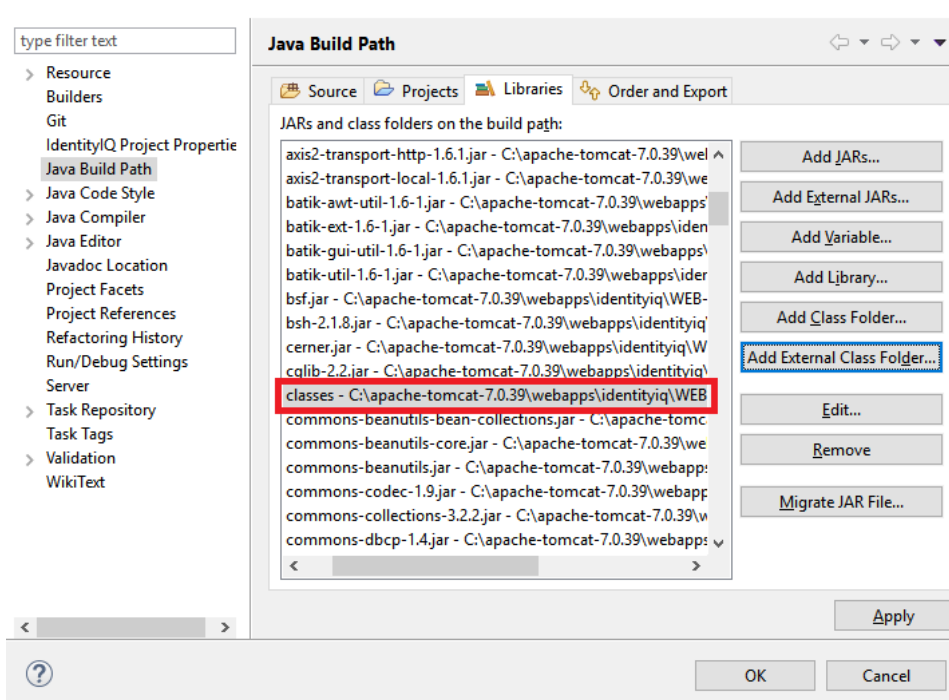
1. Open up Eclipse. The Java Build Path for your project needs to be modified to include the IdentityIQ.jar and support libraries.
2. Right click on your project in the Package Explorer view and select “Java Build Path”:



3. Add the IdentityIQ jar files so that an instance can be instantiated in Eclipse. Click the “Add External JARs...” button and navigate to the instance of IdentityIQ on your local machine. Select the WEB-INF\lib folder and all the JARs in that folder:



4. Next add the Class folder, click “Add External Class Folder...” and navigate to the instance of IdentityIQ on your local machine. Select the WEB-INF\classes folder:



5. Finally, you should reference the IdentityIQJUnitHelper.jar file; you can copy the JAR file to a folder in your project or reference it externally. It can be found in the following location in the SSD file structure:
externaltools/SST_JUnitIdentityIQHelper/jar/IdentityIQJUnitHelper.jar

After these steps have been completed you should be ready to implement the JUnit helper class.

SailPointJUnitTestHelper

The SailPointJUnitTestHelper class allows for an instance of IdentityIQ to be run inside your IDE. The JUnit class used to test functionality must extend the Helper class.

```
public class MyUnitClass extends SailPointJUnitTestHelper
```

The constructor of the JUnit class must be used to authenticate to the instance of IdentityIQ in the IDE, by calling the Super class inherited from extending the SailPointJUnitTestHelper.

```
public class MyUnitClass extends SailPointJUnitTestHelper {  
  
    public MyFirstJUnitTest() throws Exception {  
        super("spadmin", "admin");  
    }  
}
```

The new JUnit class will now have access to the following additional variables:

- `context` - The SailPointContext
- `console` – The console helper class

The SailPointJUnitTestHelper class also exposes methods which can be used in a JUnit class. Javadocs have been generated for these classes and can be found under `doc/JavaDoc/JUnitIdentityIQHelper` in the SSD file structure.

My first JUnit

The following example shows the testing of a simple rule in IdentityIQ from a JUnit test.

When we start to build a JUnit test with this helper, we must first initialize the Helper class with authentication details for your implementation. This is done in the Class's constructor by calling the Super Class's constructor as highlighted below:

```
import java.util.HashMap;
import java.util.Map;
import org.apache.log4j.Logger;
import org.junit.Test;
import sailpoint.object.Rule;
import sailpoint.services.standard.junit.SailPointJUnitTestHelper;
import sailpoint.tools.GeneralException;

public class MyFirstJUnitTest extends SailPointJUnitTestHelper {

    private static Logger log = Logger.getLogger(MyFirstJUnitTest.class);

    public MyFirstJUnitTest() throws Exception {
        super("spadmin", "admin");
    }

}
```

Next, I want to test my rule which is currently imported into IdentityIQ. The rule I am testing is:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Rule PUBLIC "sailpoint.dtd" "sailpoint.dtd">
<Rule language="beanshell" name="Add values rule" >
    <Source><![CDATA[

        return value1 + value2;

    ]]></Source>
</Rule>
```

This rule simply adds two values together. My JUnit test method looks like:

```
@Test
public void addValuesTest() throws GeneralException {

    Rule rule = console.getObjectByName(Rule.class, "Add values rule");
    Map<String, Object> args = new HashMap<String, Object>();

    int value1 = 1;
    int value2 = 2;

    args.put("value1", value1);
    args.put("value2", value2);
```



```
int count = (int) console.runRule(rule, args);  
log.debug("Add " + value1 + " + " + value2 + " == " + count + ".");  
  
assert(count == (value1 + value2));  
  
}
```

Note the `@Test` annotation used by JUnit. The method fetches the “Add values rule” and runs the rule using the JUnit Helper’s `console.runRule` method. The returning object is cast back to an integer and the `assert` statement ensures the result is `value1 + value2`.

If the `assert` method fails at this point, then this method will fail on the unit test.