SST: JUnit IdentityIQ Helper User Guide

Document Revision History

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

Table of Contents

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.

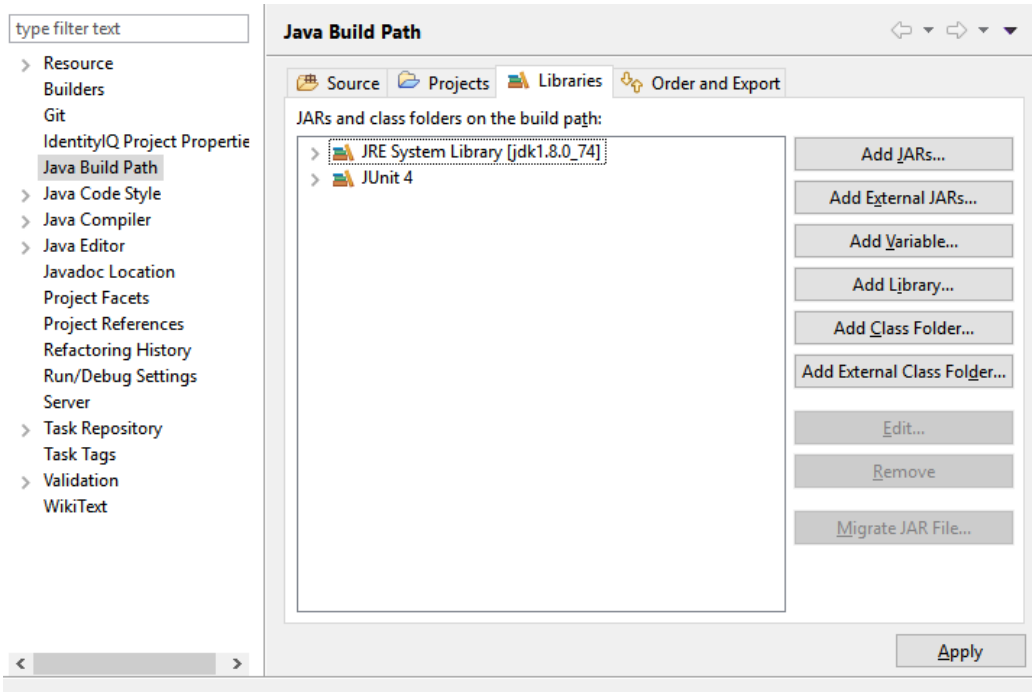
Filename	Description
IdentityIQJUnitHelper.jar	The JUnit helper classes

Eclipse Configuration

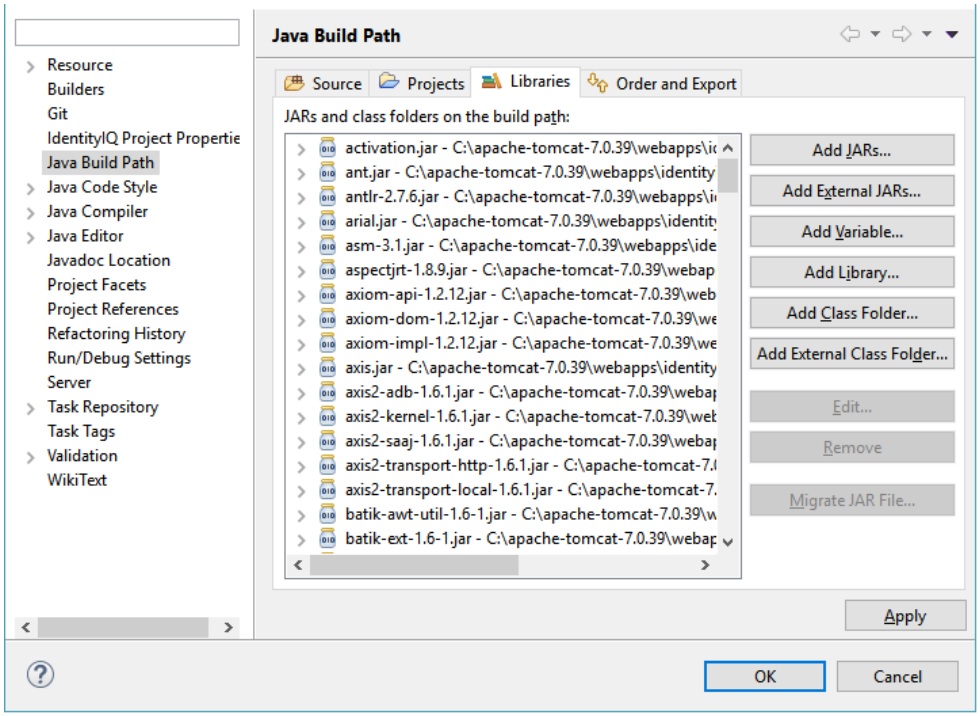
It is assumed that JUnit has already been configured in the Eclipse IDE. JUnit is usually installed as part of the J2EE version of Eclipse and should not require any further configuration.

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.

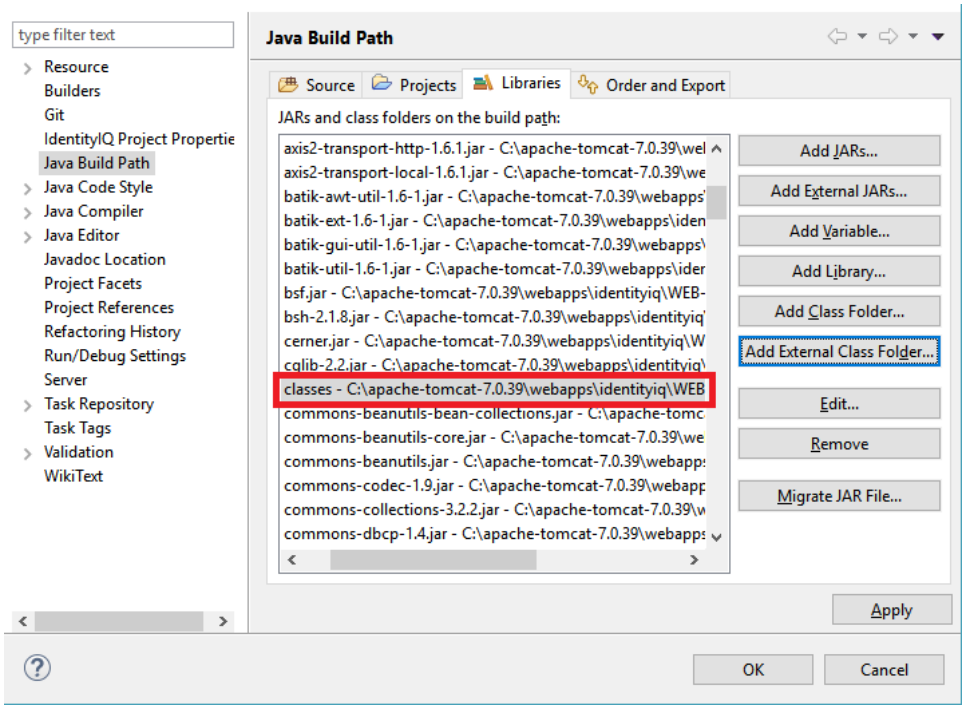
1. Open up Eclipse. The Java Build Path for your project needs to be modified to include the IdentityIQ.jar and support libraries.
2. Right click on your project in the Package Explorer view and select “Java Build Path”:



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.