Integration Unit Testing on SAP NetWeaver Application Server

Applies To:

This technical article applies to the SAP NetWeaver Application Server (Java), SAP NetWeaver Developer Studio, Unit Testing, Integration Unit Testing, JUnit, and JUnitEE.

Summary

Unit testing is an excellent way to improve software quality and decrease the cost of software defects by finding and addressing defects early in the development cycle. The JUnit open source framework is the standard API/tool for constructing and executing unit tests. The JUnit toolset is an excellent API for testing client-side Java applications but does not provide a built-in way to execute tests that require Application Server resources (such as a JNDI lookup for a database connection). An open source framework called JUnitEE was developed to address this need for “integration” unit testing. The JUnitEE toolset extends and enhances the familiar JUnit capabilities and allows the development of unit tests that are executed on the application server and have full access to server-side resources. This document covers an example of using the JUnitEE toolset to develop and execute integration unit tests on the SAP NetWeaver Application Server.

By: Jason Cline

Company: Meridium, Inc.

Date: 10 January 2006

Table of Contents

Introduction

Unit testing is a procedure used to verify that a particular portion of source code is working properly. The basic theory behind unit testing is that if one can verify the individual functions of a piece of software are working properly then the chance of the software working properly when integrated with other software is increased.

Unit testing has been an informal verification process used by software developers for years. As part of the informal verification process software developers would often construct “throw-away” test applications or test harnesses that would exercise the functionality of the software they constructed. The test harnesses were built by each developer so there was no uniform framework to which the tests adhered so it was difficult to share tests among the development team. These informal unit tests were generally disposed of (or ill maintained) after the first successful test execution so they were not run on a regular basis to ensure later changes to the software did not introduce defects to the existing software.

Unit testing underwent a dramatic evolution in the early-to-mid 90’s when Kent Beck developed a set of standard patterns and tools for building unit tests in SmallTalk. The patterns and tools developed by Kent overcame the deficiencies of informal unit testing approaches and provides the ability to construct re-usable, standard unit tests and run them frequently. These patterns and tools are foundations of Extreme Programming (XP) and Test Driven Development (TDD) however good quality, repeatable unit tests are the cornerstone to any high-quality development process you may be using.

The unit testing patterns and foundations of Kent’s work in SmallTalk have been applied and adapted many times over to a wide array of programming languages. These are generally referred to as the “xUnit” patterns and specific implementations for a given language are for example called “JUnit” for Java, “SUnit” for SmallTalk, and “CUnit” for C. This paper will deal specifically with the JUnit implementation of the unit testing framework pattern, which is provided in the SAP NetWeaver Developer Studio, and some extensions and tools that overcome deficiencies in the base implementation when testing classes that require access to application server resources such as JNDI lookups for configured JDBC datasources.

Unit Testing vs. Integration Unit Testing

The classic approach to unit testing using JUnit assumes isolation of the code being tested. Running tests in isolation insures that any failures are really failures in the code being tested and not failures of other objects or failures to access other resources. For example, let's take a look at the following Java class called StandardCalculator that adds two integers and returns the value:

```
package com.mycompany;
public class StandardCalculator
{
  public StandardCalculator()
```

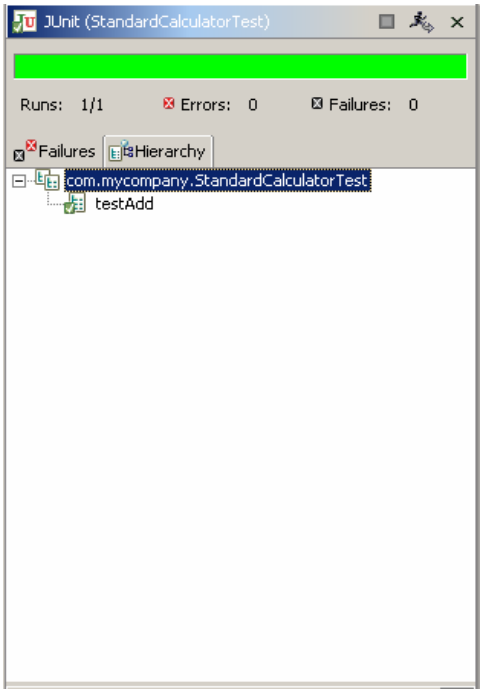
```
    {
    }
    public int Add(int v1, int v2)
    {
          int result = v1 + v2;
          return result;
    }
}
```

As you can see the Add method of the StandardCalculator class is could easily be isolated into a single unit test. Because the function does not utilize any application server resources a JUnit unit test for this class would not require the development of any mock objects. The following code sample illustrates a very simple JUnit unit test for this function:

```
package com.mycompany;
import junit.framework.TestCase;
public class StandardCalculatorTest extends TestCase
{
   public StandardCalculatorTest(String name)
   {
         super(name);
   }
   public void testAdd()
   {
         int value1 = 99;
         int value2 = 1;
         int result = 0;

         StandardCalculator calc = new StandardCalculator();
         result = calc.Add(value1, value2);
         TestCase.assertEquals(result, 100);
   }
}
```

After creating the StandardCalculator class and its associated unit test you can build and run the unit test in the SAP NetWeaver Developer studio to verify it was successful.

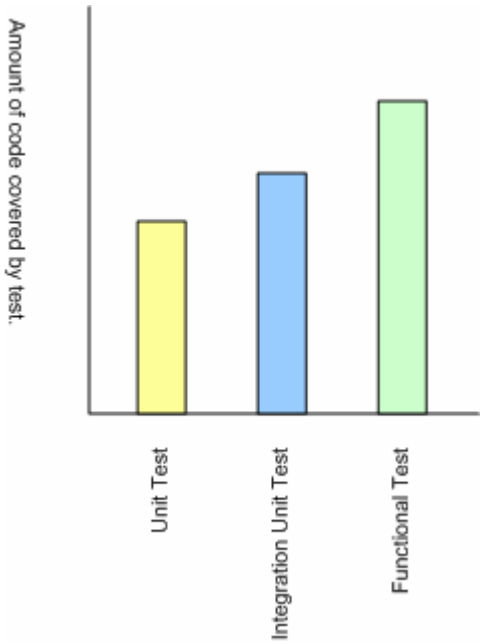


The classic approach to unit testing works great for standalone Java applications and classes which can easily be executed in isolation and do not require access to application server resources. Most applications developed for the SAP Web Application Server are much more complex than this simple example and will require access to other objects and resources provided by the application server.

From the classic unit testing perspective, these more complex pieces of code which access external resources provided by an application server should use “mock” objects to simulate the resource. Developing mock objects to simulate the resources provided by the SAP Web Application Server would be extremely difficult, time consuming, and ultimately add little value to the task at hand.

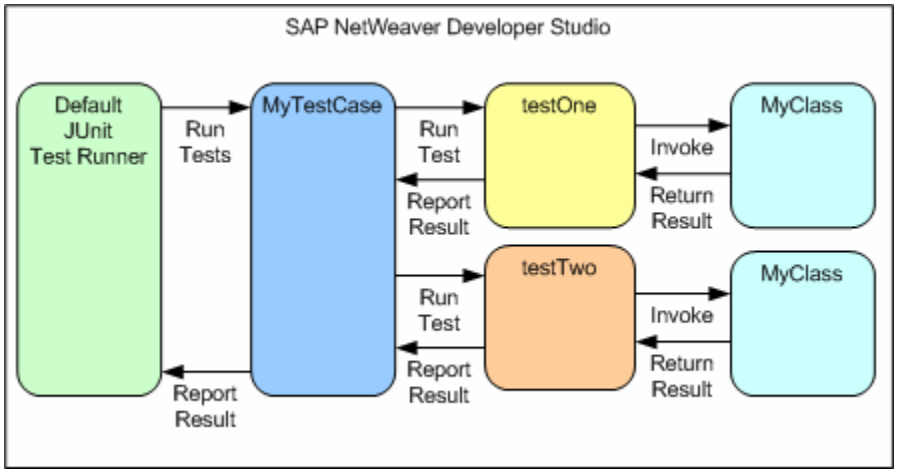
To address the issue of business objects and EJBs which need access to the application server context during a unit test there is an alterative to the classic unit testing approach called integration unit testing. Integration unit testing involves running unit tests in an environment where they have full access to the “real” objects and resources involved in the unit.

Integration unit tests involve the execution of the unit and anything the unit is dependent upon. A typical integration unit test will fall somewhere between a classic unit test and a functional unit test on the testing continuum with these tests having greater coverage than a classic unit test but less than the typical functional test. The following chart illustrates the relative differences in the amount of code executed for typical categories of testing:

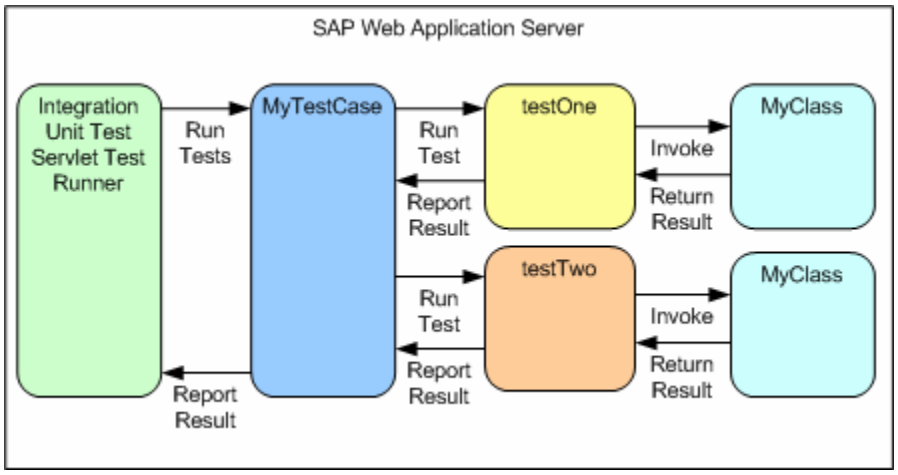


Integration Unit Testing Extensions

Integration unit tests are really nothing more than “standard” JUnit unit test that are executed in an environment where the test and the objects being tests have full access to the application server resources. To illustrate this concept let’s first look at the flow of execution when running a standard JUnit test in SAP NetWeaver Developer Studio as depicted in the following illustration:



Integration unit testing extensions work in a very similar way to the classic JUnit test execution with the exception that the execution context is moved from the Developer Studio environment to the SAP NetWeaver Application Server environment. The following illustration shows the execution flow of an integration unit test:



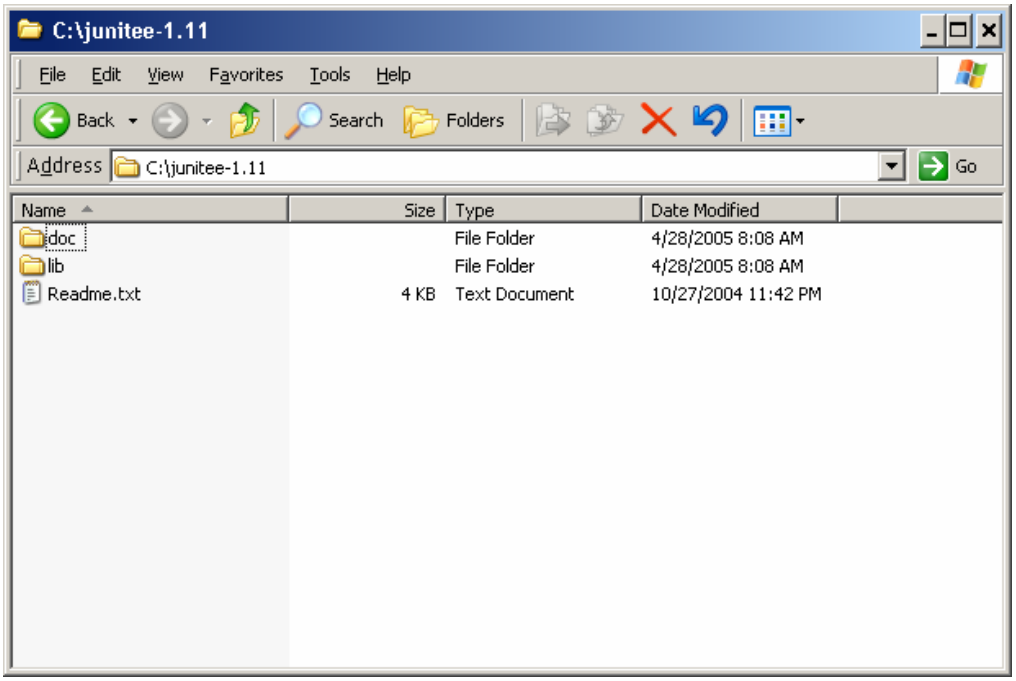
The Java development community has two primary options for integration unit test frameworks namely Jakarta Cactus and JUnitEE. Both frameworks are open source projects and both provide a servlet test runner that can be deployed to application servers to execute JUnit unit tests. I will not be comparing and contrasting the two frameworks, beyond that cursory background, as part of this paper. The remaining sections of this paper will provide an example of using the JUnitEE framework

Integration Unit Testing with JUnitEE

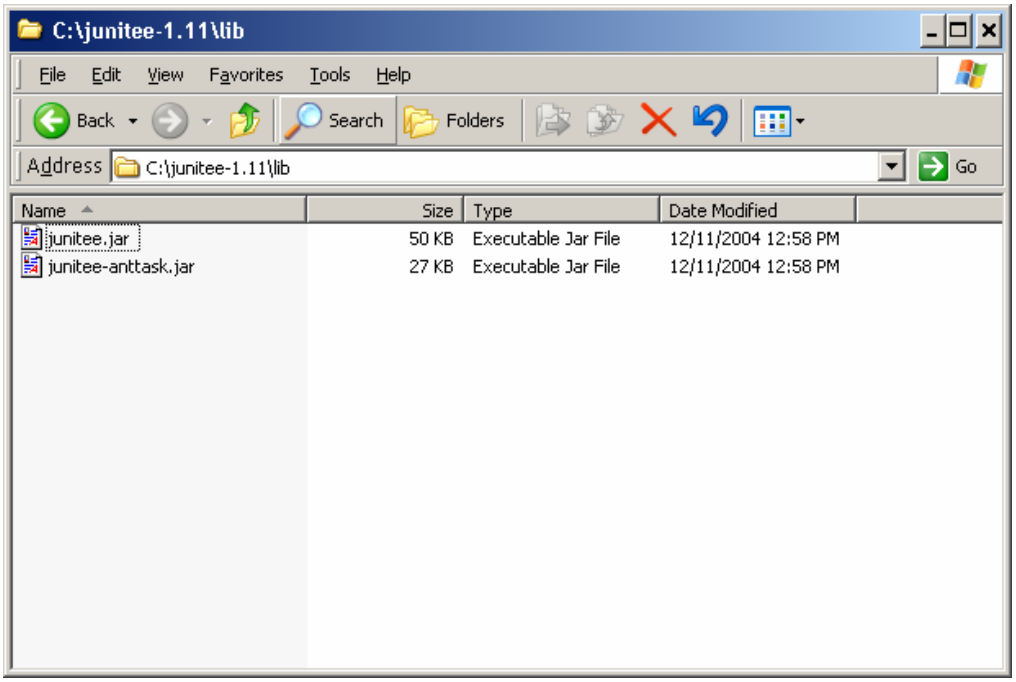
Now we will walk through an example of using JUnitEE to run unit tests within the context of the SAP NetWeaver Application Server. Note that this tutorial is assuming a local installation of the SAP NetWeaver Application Server (Java) 6.40 SP12 and SAP NetWeaver Developer Studio 2.0.12.

Downloading and Installing JUnitEE

First you will need to download JUnitEE. After downloading, extract the contents of the ZIP file to a local drive:



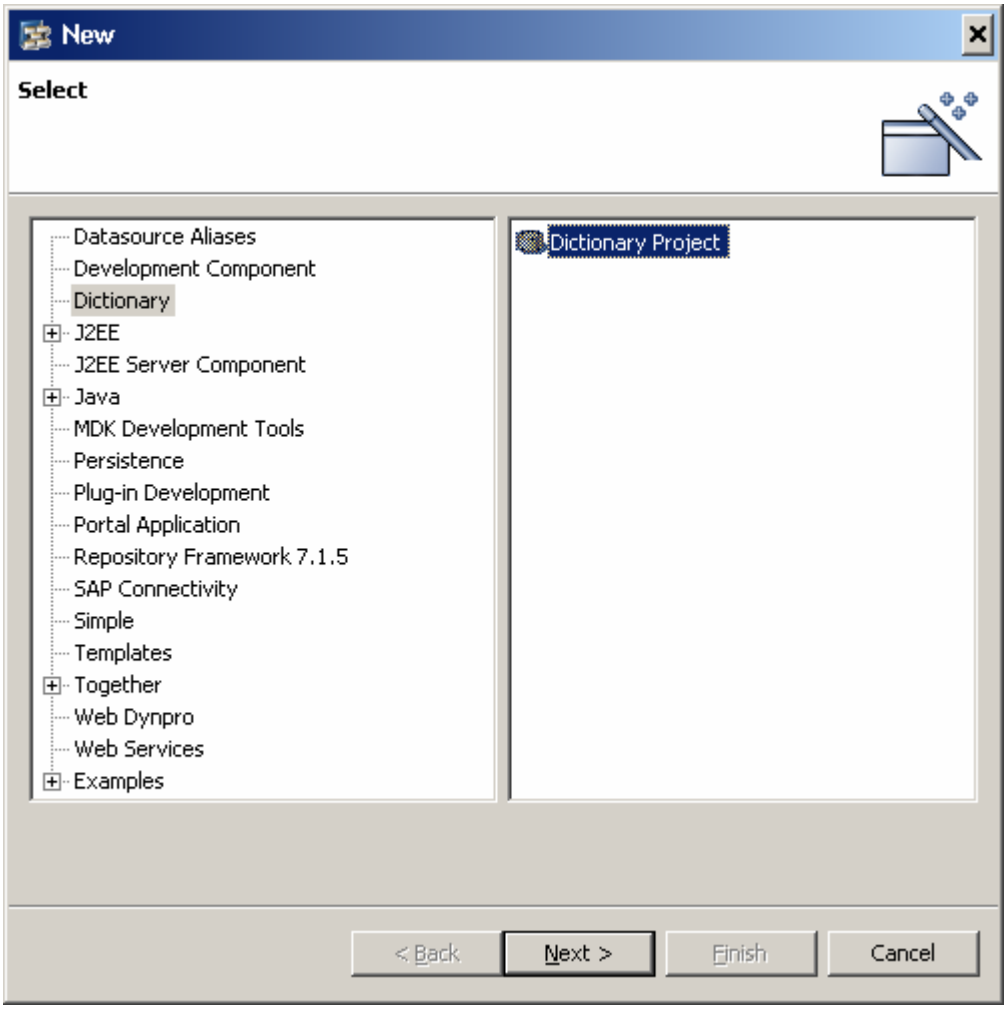
The main file that we are interested in is called junitee.jar file and it's located in the lib folder:



Create and Deploy a New Dictionary Project

To continue our example, we will now create a new Java Dictionary project. We will be extending our calculator example to store the results of the calculation in the Java Dictionary. To create the dictionary take the following steps:

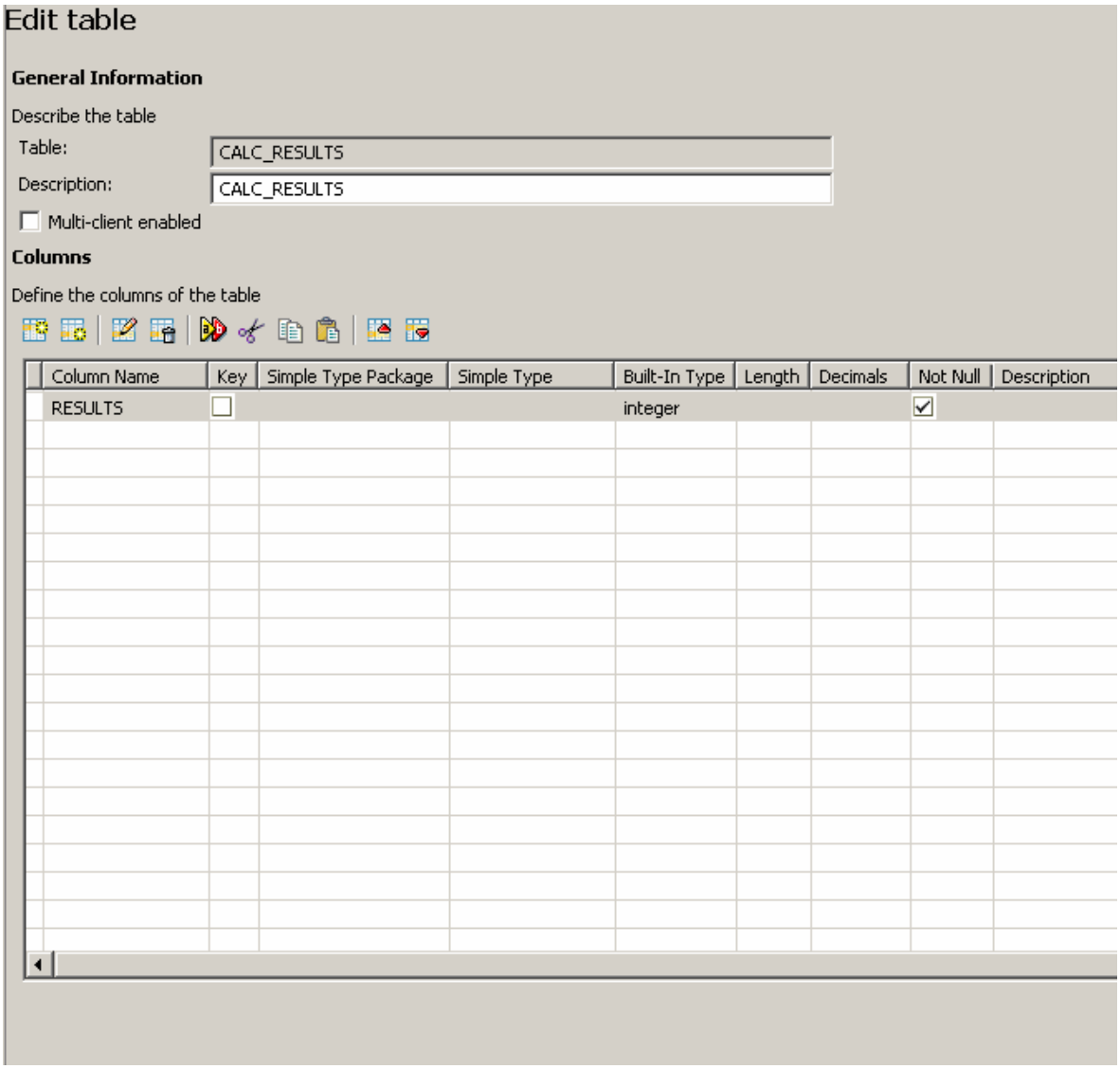
1) From the file menu of Developer Studio choose New → Project and select Dictionary



2) Name the project CalcResults and click finish. The dialog should automatically close and open the Dictionary Perspective of Developer Studio.

3) In the dictionary explorer, expand the CalcResults node and then the Local Dictionary node. Select the Database Tables node then right-click and choose “Create Table.” When prompted, name the table CALC_RESULTS and click OK. After clicking OK the table should automatically open in Edit Mode.

4) With the Edit Table view open for the CALC_RESULTS tables, click in the first row and add a column named RESULTS with a built-in data type of integer.



5) Next, open the file menu and click Save all metadata.

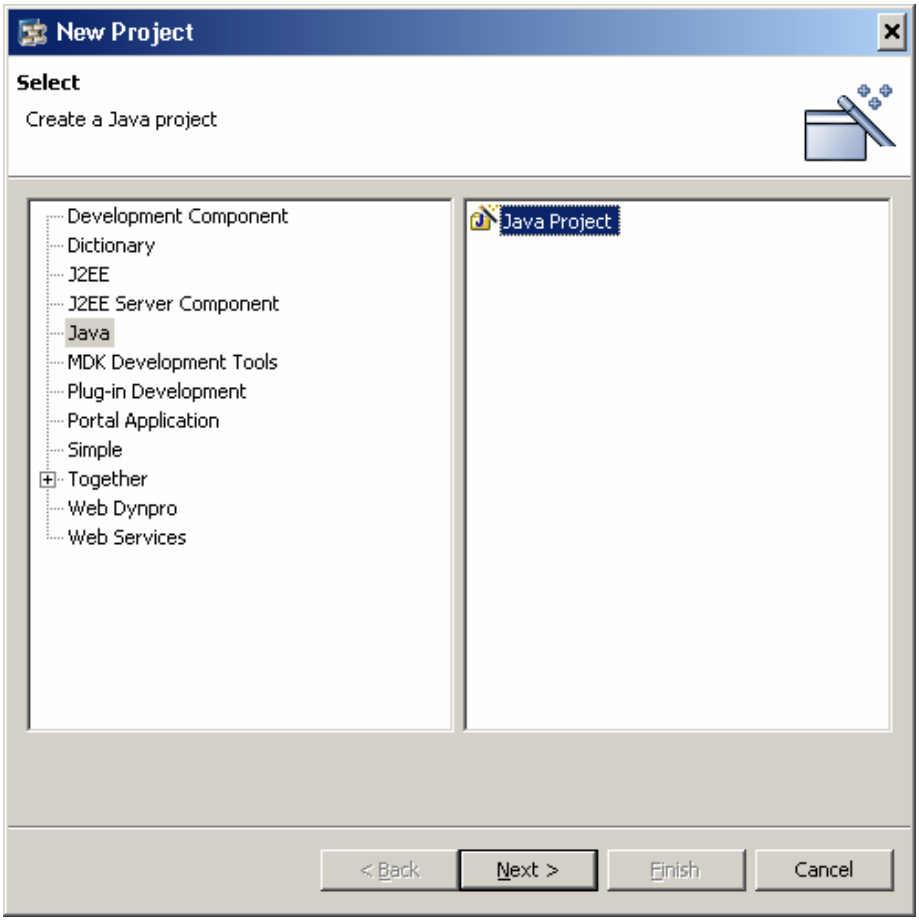
6) In the dictionary explorer, right-click the CalcResults project and choose Build. Then right-click the CalcResults project and choose Create Archive. Finally, right-click the CalcResults project and choose Deploy Archive.

We now have a Java Dictionary table we can use to store our calculation result.

Create a New Java Project

Next, we will create a new Java project and add a calculator class that stores the calculation in the Java Dictionary.

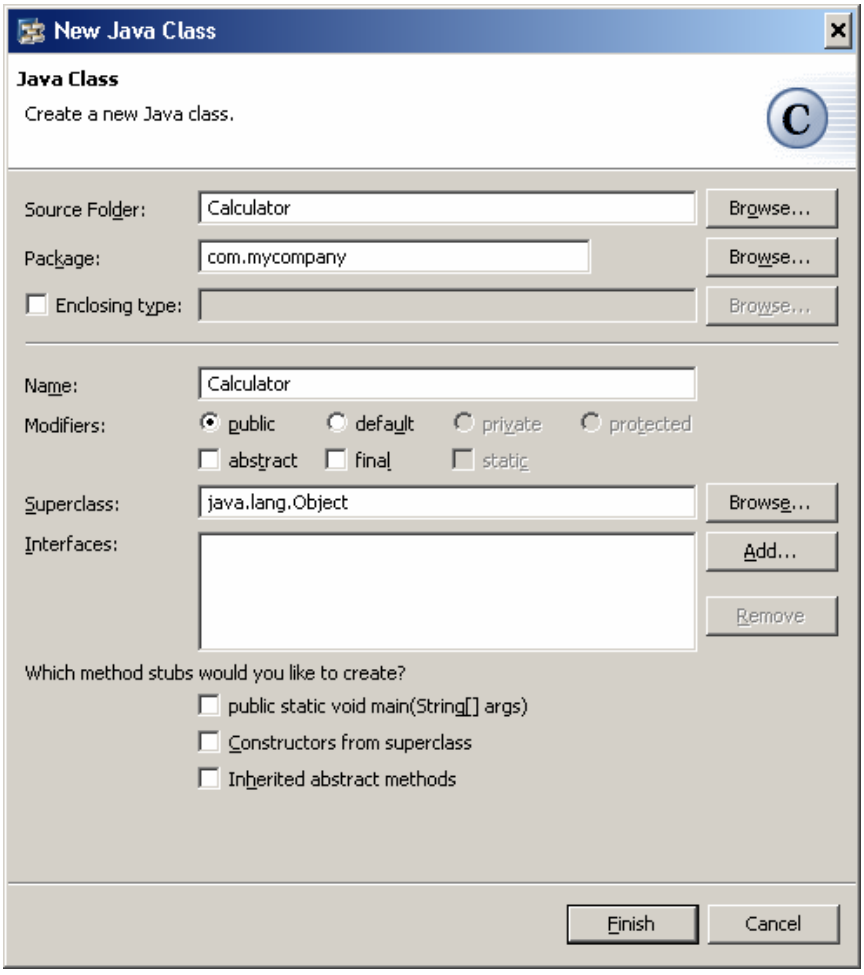
1) From the file menu in Developer Studio choose New → Project and select Java.



2) Name the project Calculator and click Finish.

3) In the package explorer view of the Java perspective right-click the Calculator project and choose New → Package. When prompted, name the package com.mycompany and click OK.

4) Next, in the package explorer, right-click the newly created com.mycompany package and choose New→ Class. When prompted, name the class Calculator and click Finish.



5) Next we will modify the code in the Java class that we just created to include an Add function that looks-up a JDBC datasource defined on the SAP NetWeaver Application Server and uses the datasource connection to store the calculation result in the Java Dictionary. Here's the complete content of the Calculator java class:

```
import javax.sql.*;
import java.sql.*;
import javax.naming.*;

public class Calculator
{
  public Calculator()
  {
  }
```

```
    public int Add(int v1, int v2) throws SQLException, NamingException
    {
          // calculate the result
          int result = v1 + v2;

          // do a local lookup of the datasource alias.
          InitialContext context = new InitialContext();
          DataSource dataSource = (DataSource)context.lookup("jdbc/CALC_DS");
          Connection conn = dataSource.getConnection();


          String sql = "INSERT INTO CALC_RESULTS(RESULTS) VALUES(?)";
          PreparedStatement statement = conn.prepareStatement(sql);
          statement.setInt(1, result);

          return result;
    }
}
```

6) Next choose File → Save All.

7) Finally, right-click the Calculator project in the package explorer view of the Java perspective and choose Build Project.

Configure JDBC Connection

The Calculator class created in the previous section includes an Add function which does a lookup for a JDBC Connection configured on the SAP NetWeaver Application Server. This section will show you how to configure that datasource using the SAP Visual Administrator.

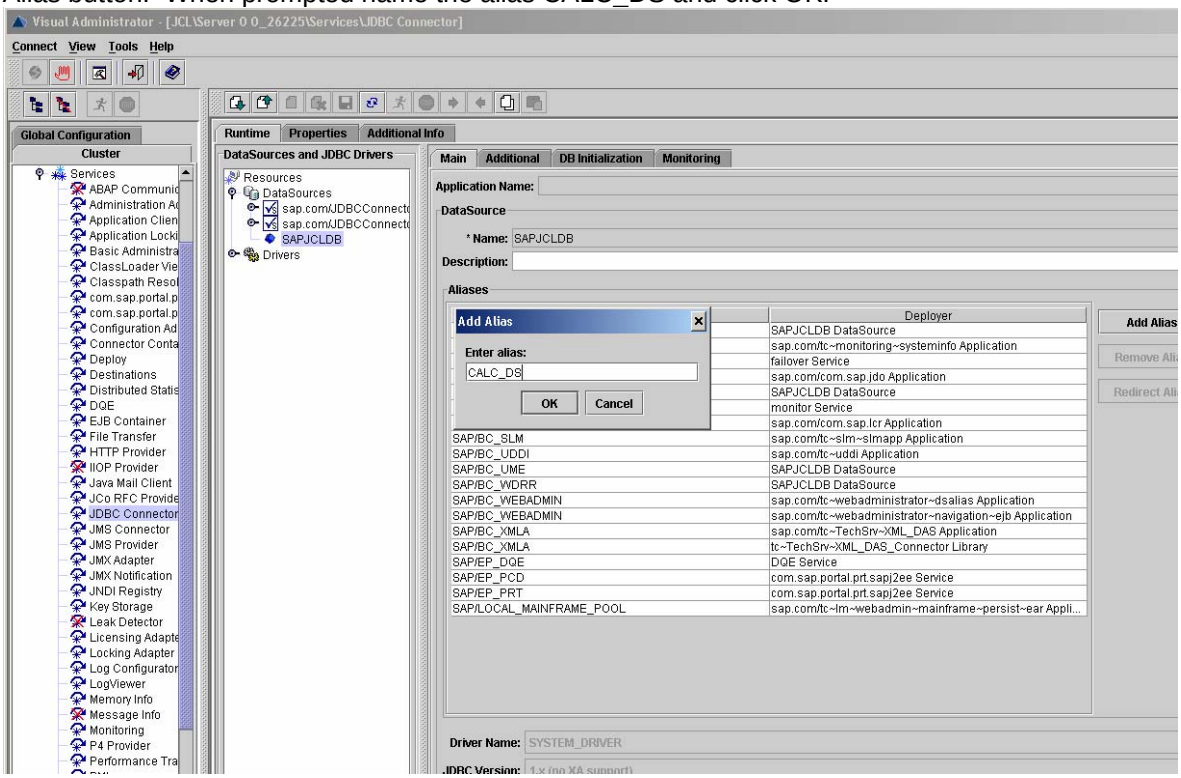
1) Launch the SAP Visual Administrator.

2) Login using the administrator account.

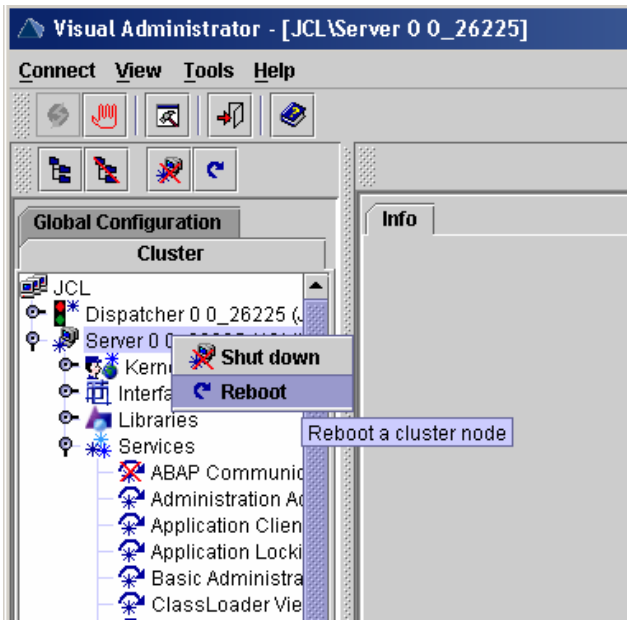
3) After the tree in the left pane of the Visual Administrator finishes loading, expand the Server node, then expand the Services node.

4) Next, select the JDBC Connections node listed under Services.

5) Expand the Datasources node to display the default system datasource. Select it and click the Add Alias button. When prompted name the alias CALC_DS and click OK.



6) Finally, in order to make the alias available you must restart the SAP Server. To do this right-click the server and choose Reboot.

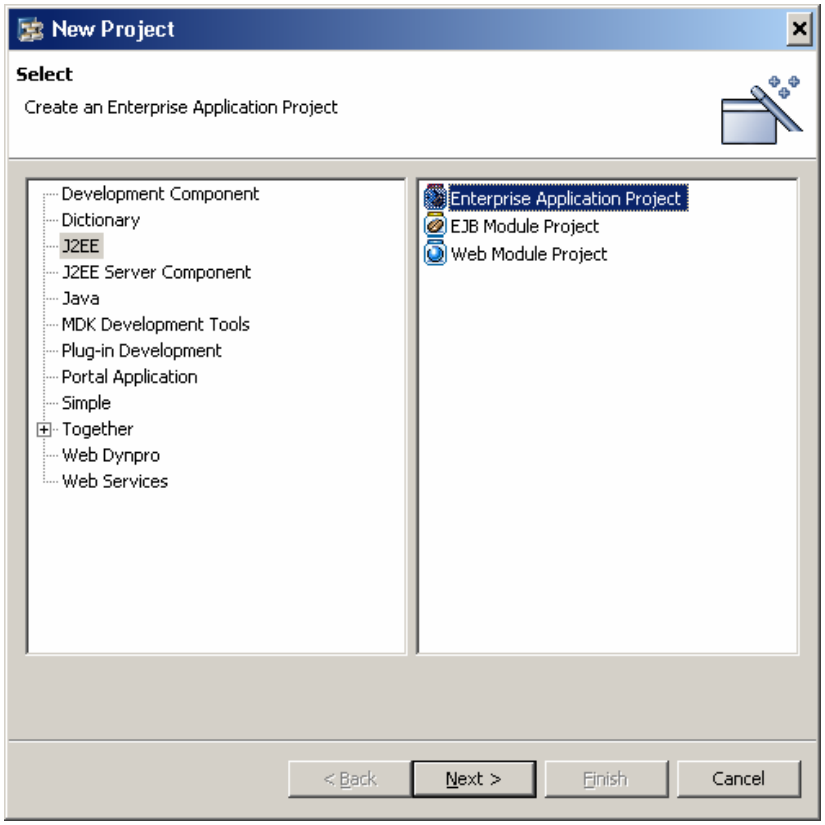


Configure and Deploy the Integration Unit Test and the Integration Unit Test Servlet

Finally we will create and deploy our integration unit test servlet and an integration unit test that tests the Calculator java class we constructed in the previous section. Creating and deploying a servlet on the SAP web application server requires the creation of both a web module project and an enterprise application project which is used as the deployment mechanism for the servlet.

Create the Enterprise Application Project

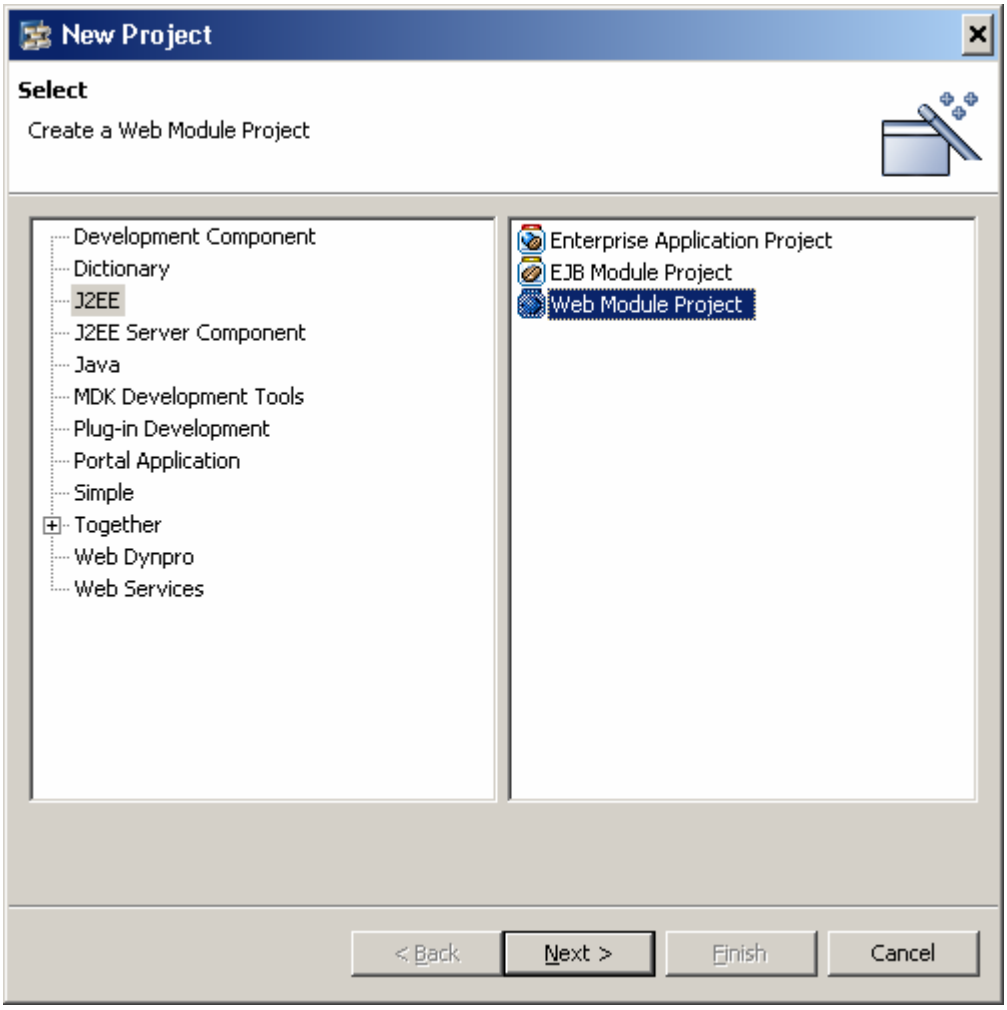
1) From the Developer Studio file menu choose New → Project and select J2EE then Enterprise Application Project and click Next.



2) Name the project JUnitEE and click finish.

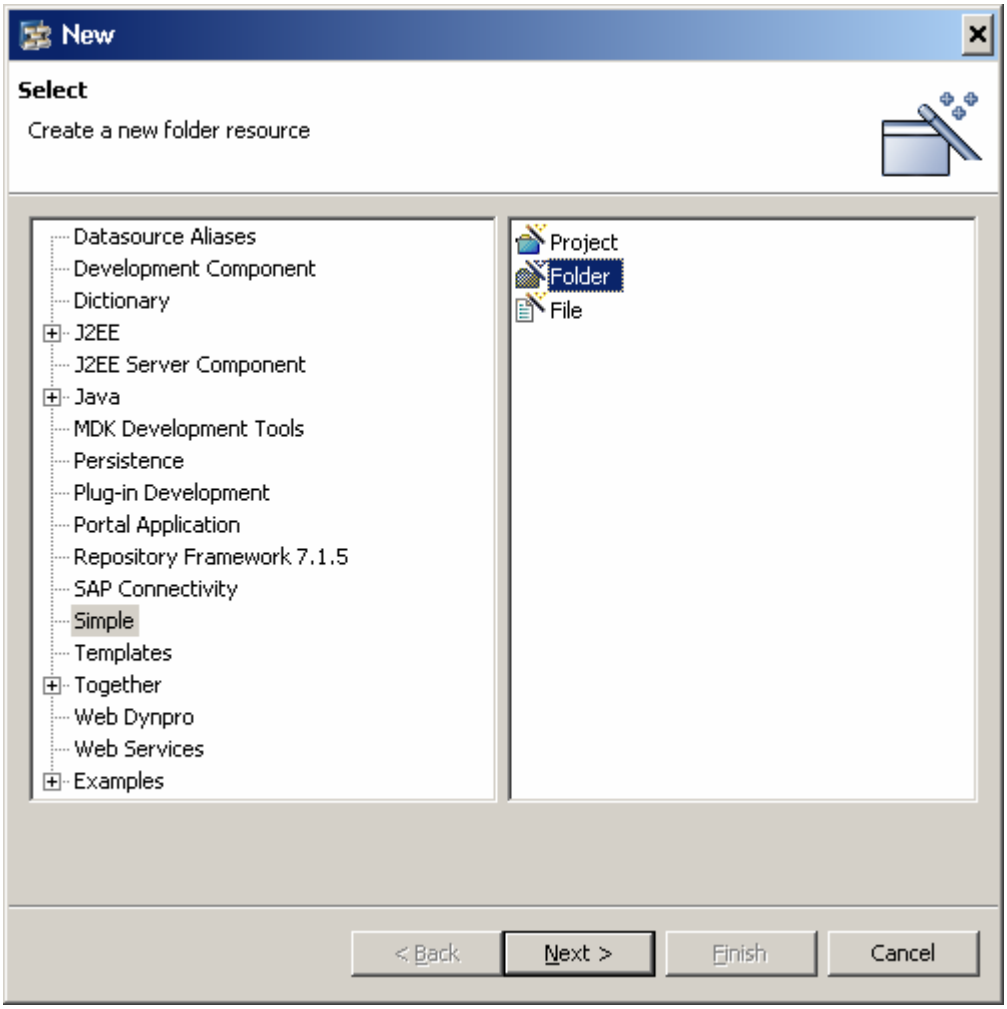
Create the Web Module Project

1) From the Developer Studio file menu choose New → Project and select J2EE then Web Module Project then click next.



2) Name the project JUnitEEFramework and click Finish.

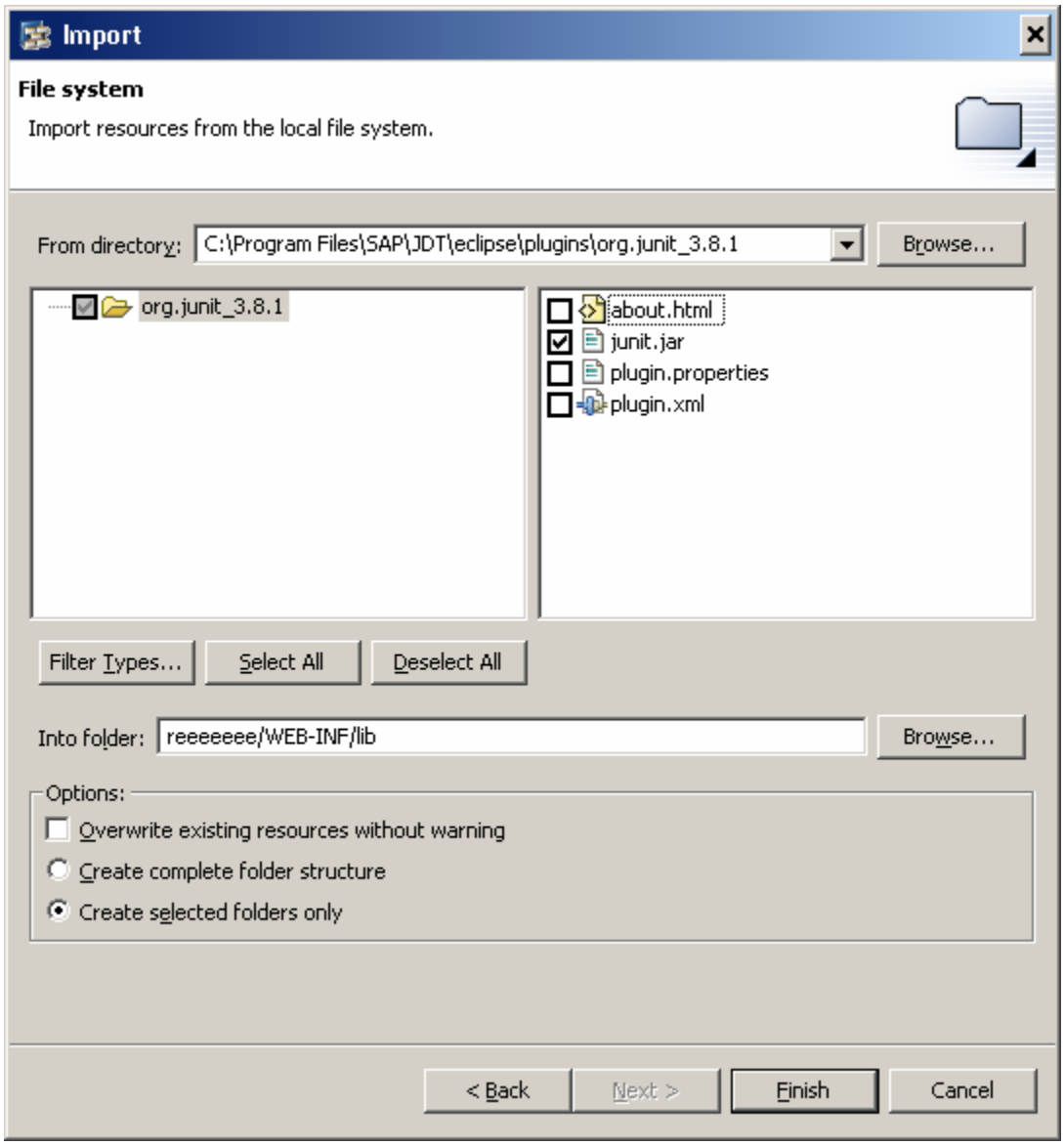
3) Open the Navigator view, locate the JUnitEEFramework project and right-click the WEB-INF and choose New → Other. When the dialog appears, select Simple then choose Folder.



4) When prompted name the folder lib and click finish.

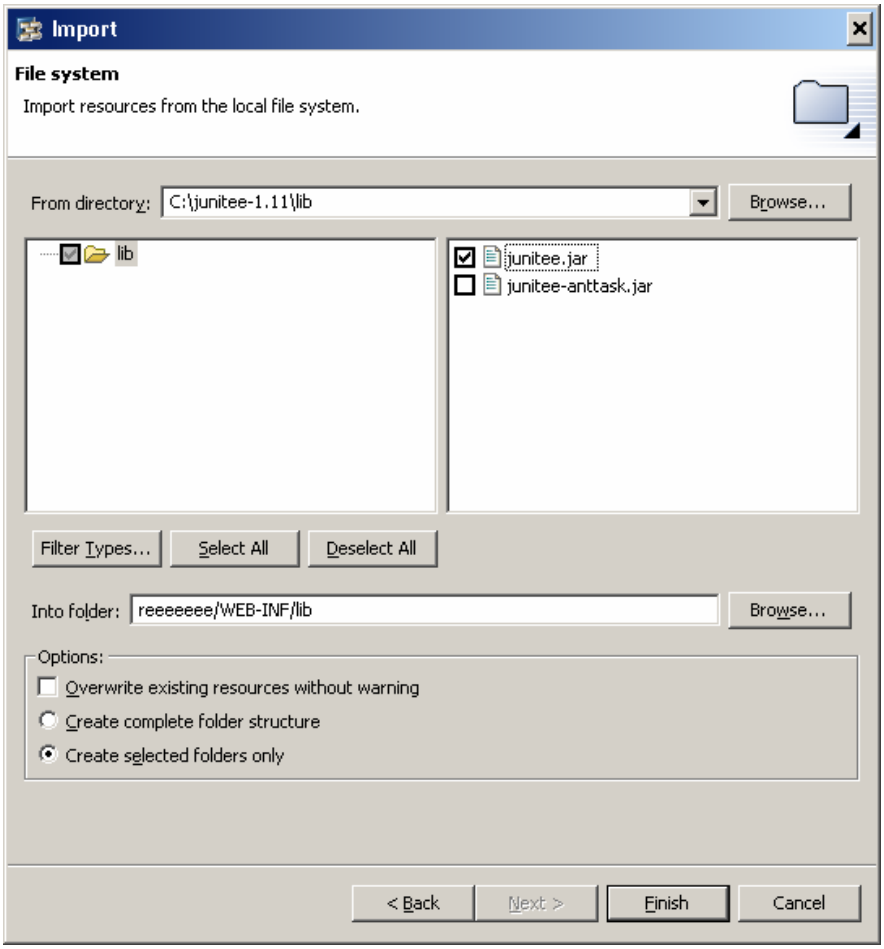
5) Right-click the new folder and choose Import. When prompted, select File System then click next.

6) Browse for the junit.jar file. For a typical Developer Studio installation this file can be found in the following folder: C:\Program Files\SAP\JDT\eclipse\plugins\org.junit_3.8.1. Select the junit.jar file and click finish.



7) Next, right-click the lib folder and again choose import. When prompted, select file system and click next.

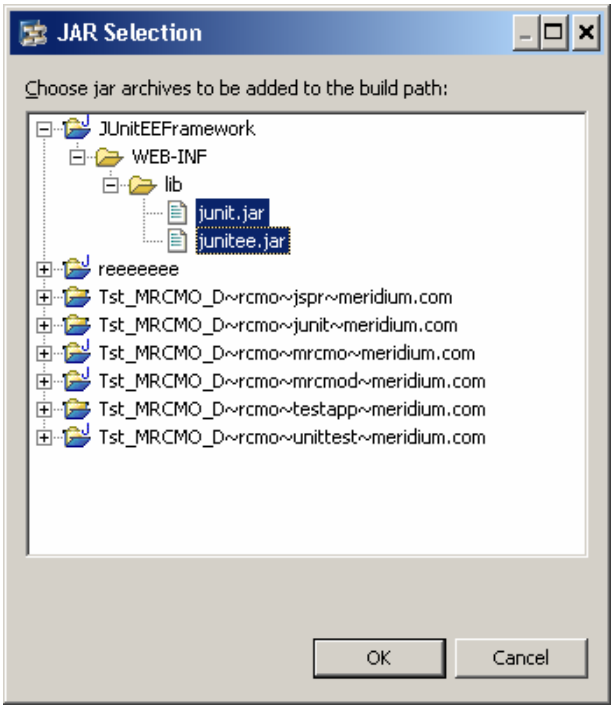
8) Browse for the junitee.jar file that you downloaded and extracted to your local drive as outlined previously in this paper.



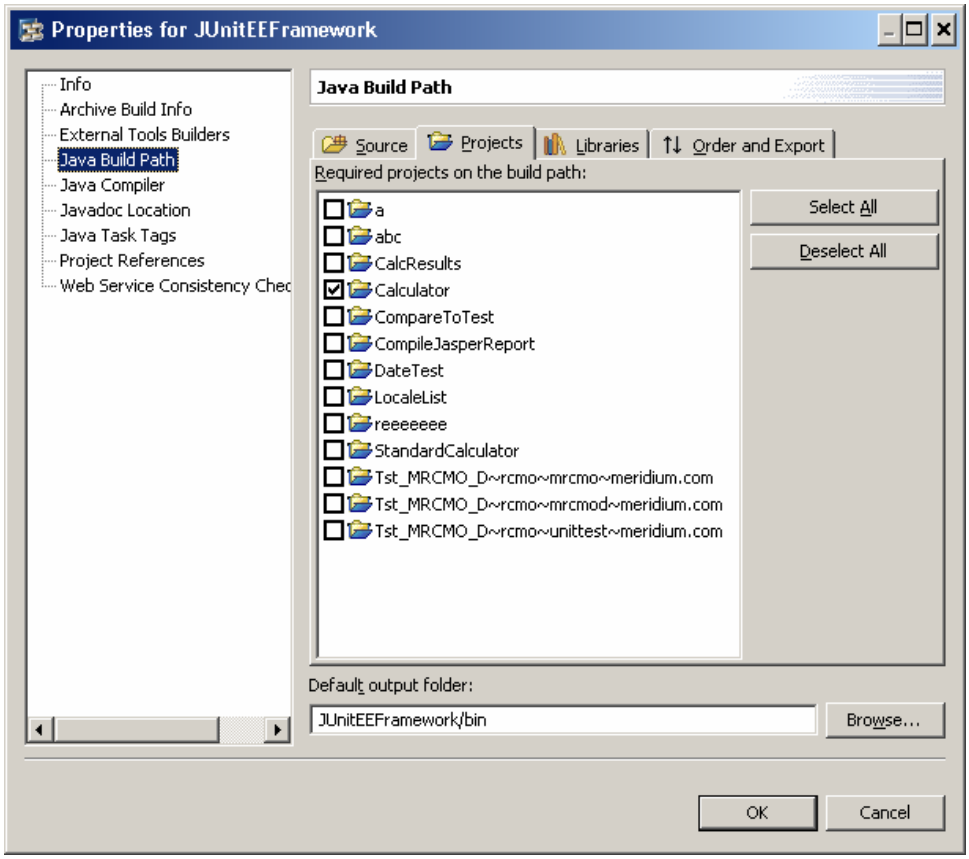
9) Open the J2EE Development Perspecitive, open the J2EE explorer then right-click the JUnitEEFramework project and choose Properties.

10) In the properties dialog open the Libraries tab and click Add Jars.

11) When prompted, choose the JUnitEEFramework project, expand the WEB-INF folder, expand the lib folder and select both junit.jar and junitee.jar then click OK.



12) Before closing the properties dialog, select the projects tab and choose the Calculator project then click OK.



13) Open the web.xml file and choose the source tab. Replace the source with the following:

```
<web-app>
    <display-name>JUnitEE Unit Test Servlet</display-name>
    <servlet>
        <servlet-name>UnitTestServlet</servlet-name>
        <description>JUnitEE Unit Test Servlet</description>
        <servlet-class>org.junitee.servlet.JUnitEEServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>UnitTestServlet</servlet-name>
        <url-pattern>/UnitTestServlet/*</url-pattern>
    </servlet-mapping>
</web-app>
```

14) Open the J2EE Explorer view and right-click the source folder of the JUnitEEFramework project then select New → Package. When prompted name the package com.mycompany and click Finish.

15) Expand the package then select mycompany, right-click and choose New→Java Class. Name the class CalculatorTest and click finish. Replace the class contents with the following code:

```
package com.mycompany;
import junit.framework.TestCase;
public class CalculatorTest extends TestCase
{
   public CalculatorTest(String name)
   {
         super(name);
   }
   public void testAdd()
   {
         int value1 = 99;
         int value2 = 1;
         int result = 0;

         Calculator calc = new Calculator();
         try
         {
               result = calc.Add(value1, value2);
         }
         catch(Exception ex)
         {
               TestCase.fail(ex.getMessage());
         }
         TestCase.assertEquals(result, 100);
   }
}
```

16) From the file menu, choose File → Save All.

17) Open the navigator view and right-click the WEB-INF folder for the JUnitEEFramework folder and choose New → Other.

18) Select Simple then File and click next. When prompted, name the file testCase.txt and click finish.

19) Add the following line to the file then save it: `com.mycompany.CalculatorTest`

20) Open the J2EE explorer and right-click the JUnitEEFramework project then select add to ear project.

21) Select the JUnitEE project from the list and click OK.

22) Finally, right-click the JUnitEEFramework project and choose build web archive.

Deploying the Integration Unit Test Servlet

To deploy the integration unit test servlet take the following steps:

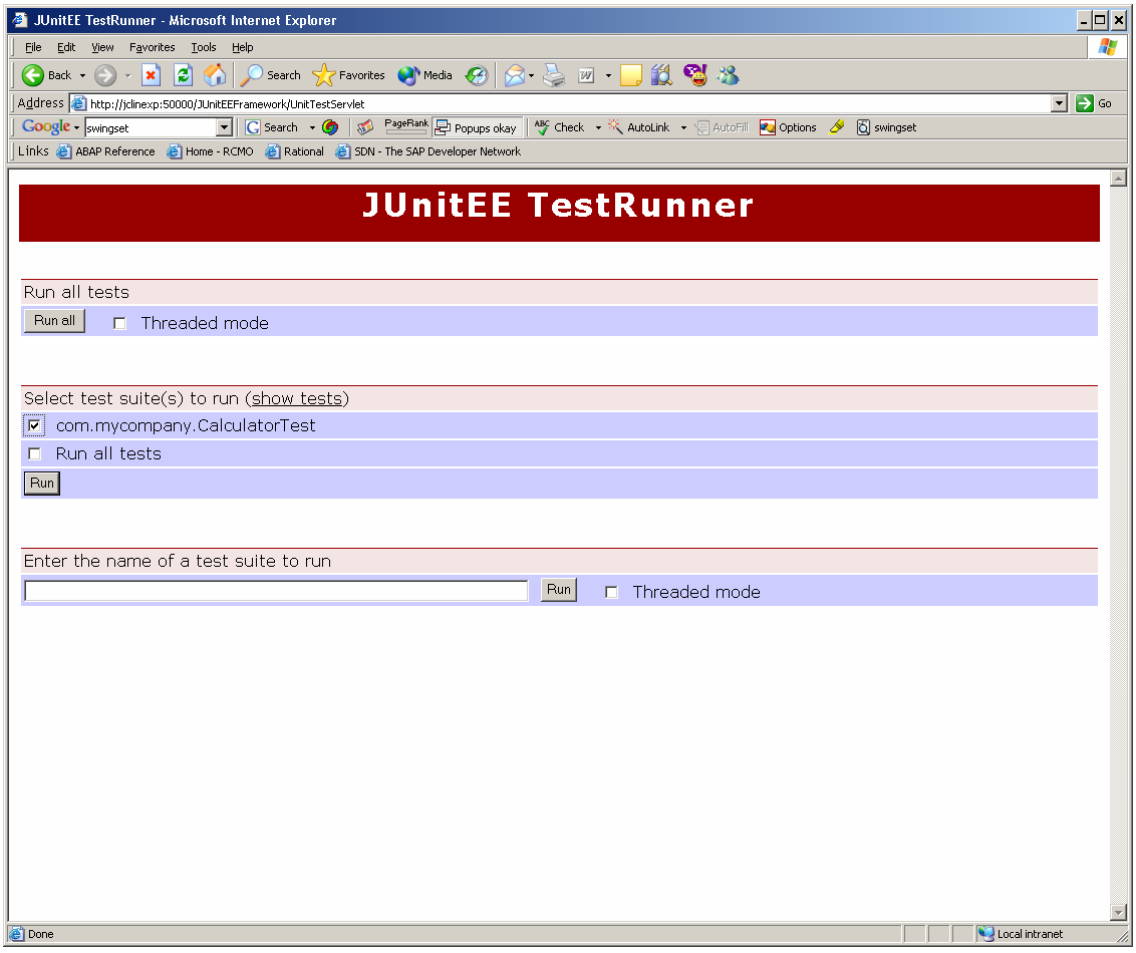
1) After building the web archive as outlined in the previous section, open the J2EE explorer view and right-click the JUnitEE project then choose Build Application Archive.

2) The build process should generate an EAR file. Right-click the generated EAR file and choose Deploy to J2EE engine.

Executing the Integration Unit Test

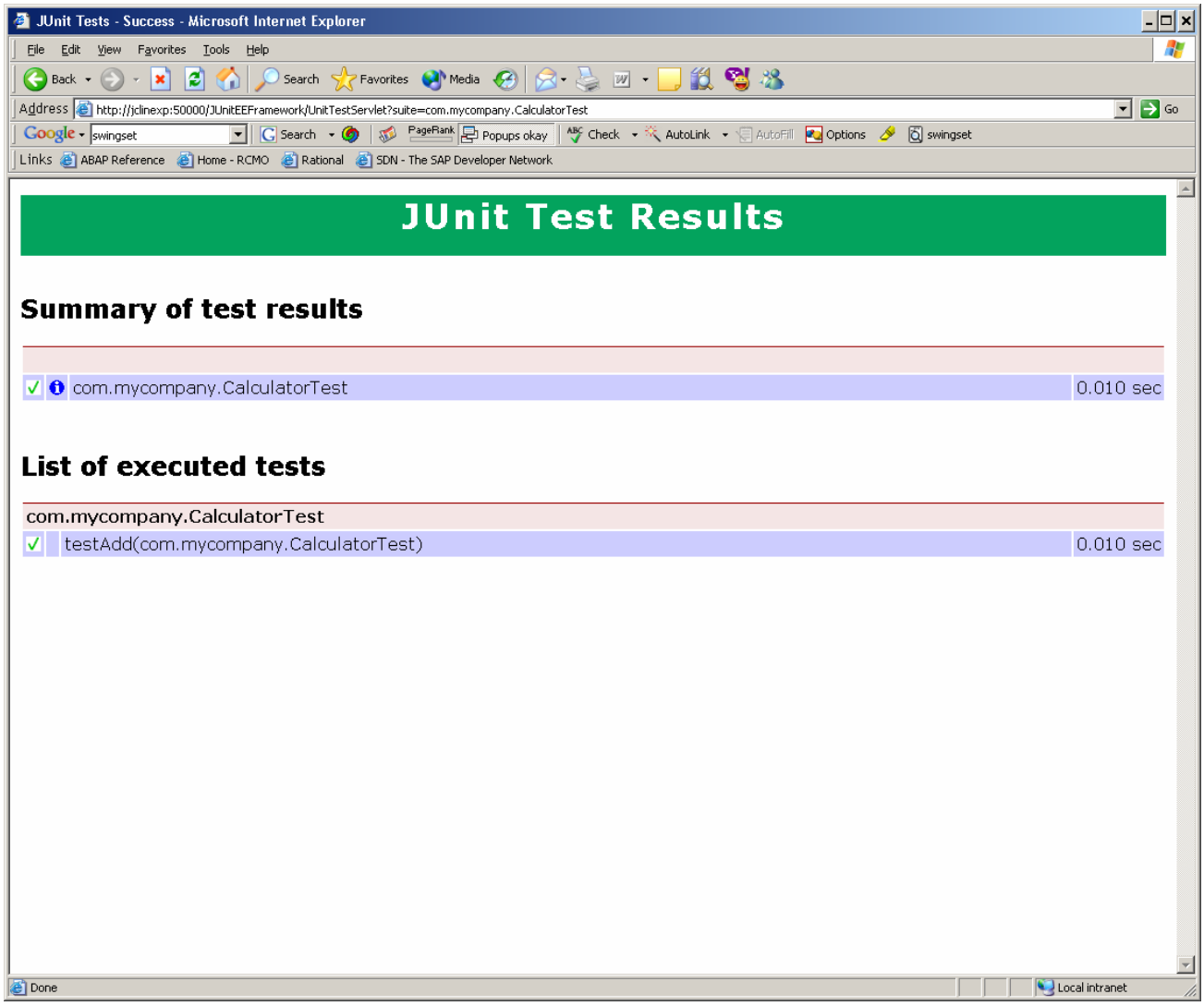
To launch the Integration Unit Test runner point your browser to:

http://server:port/JUnitEEFramework/UnitTestServlet



Select the com.mycompany.CalculatorTest test case from the list of tests and click run.

Upon successful execution of the unit test the following “green” screen will appear.



In the unlikely event ☺ that your code has bugs, the results screen will become red and the errors and corresponding call stack reported.

Summary

Integration unit testing is a great way to increase the quality of your SAP NetWeaver projects by providing a repeatable set of tests that can be executed daily.

Author Bio

Jason Cline is a Software Architect with Meridium, Inc. and has actively been involved with the design and implementation of Meridium's RCMO product built upon the SAP NetWeaver platform. You can contact him at jcline@meridium.com.

Disclaimer & Liability Notice