2004

A Test Suite Generator For Struts Based Applications

Gregory M. Jackson

University of North Florida

A TEST SUITE GENERATOR FOR STRUTS BASED APPLICATIONS

By

Gregory M. Jackson

A project submitted to the
Department of Computer and Information Sciences
in partial fulfillment of the requirement for the degree of

Master of Science
in Computer and Information Sciences

UNIVERSITY OF NORTH FLORIDA
DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES

April 2004

APPROVAL BY THE PROJECT COMMITTEE

The project "A Test Suite Generator for Struts Based Applications" submitted by Gregory M. Jackson in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences has been approved by the Project Committee:

Sentence Deleted

4/22/04

Arturo Sánchez, Ph.D.
Project Director

Sentence Deleted

4/27/04

Judith Solano, PhD.
Chairperson of the Department

Sentence Deleted

4/22/04

Charles Winton, Ph.D.
Graduate Director

ACKNOWLEDGEMENTS

This project is dedicated to my father, Marshall Jackson, who always pushed me to work hard on everything I do and showed me that if I put my mind to it, anything is possible. My father played a key role in encouraging me to start working towards a Masters degree, but passed away due to cancer prior to its completion. He showed me how to be successful in any career I should choose while continuing to dedicate time to my family and loved ones.

I would also like to thank my wife, Arlyn. She has endured many nights and weekends alone, while I worked on papers and projects. Her patience and understanding has allowed me to achieve this important milestone in my life.

CONTENTS

LIST OF FIGURES

ABSTRACT

Testing web-based enterprise applications requires the use of automated testing frameworks. The testing framework's ability to run suites of test cases through development ensures enhancements work as required and have not caused defects in previously developed sub systems. Open source testing frameworks like JUnit and Cactus have addressed the requirements to test web-based enterprise applications, however they do not address the generation of test cases based on direct analysis of the code under test.

This paper presents a tool to generate test cases for web-based enterprise applications. The generator focuses on creating test cases used to test applications built on the Struts MVC framework for the J2EE platform. Using the Struts configuration files, test cases are generated to test each request path and response. The created test cases take advantage of the StrutsTestCase library and run using the JUnit and Cactus frameworks. The generated test cases follow a consistent pattern for the test cases and reduce the time required build the automated testing for the application.

Chapter 1

INTRODUCTION

The ability to generate test cases is a very important part of large-scale software development projects. With the size and complexity of today's applications increasing, the cost of manually testing and performing regression testing is often a either a large burden on the project or unfeasible. Testing frameworks are now available to assist developers in the testing of large enterprise systems. These frameworks allow test cases to be executed throughout the development cycle thereby allowing problems to be identified as early as possible.

This project looks at further reducing the testing burden by assisting developers in the automation of the test process by generating test case classes for Struts based (J2EE) applications. The generation of test classes reduces the development time when using a testing framework. In addition, by creating a simple process to generate the test cases, it is more likely that developers will employ testing.

Chapter 1 will describe the Struts framework and its associated testing frameworks. Chapter 2 will go into the architecture of the developed test suite generator and translation rules for creating the test cases. Chapter 3 covers the results produced by

the generator. The conclusion is covered in Chapter 4 followed by the future developments for the Struts test case generator in Chapter 5.

1.1 The Struts Framework

The Jakarta Struts Framework is an open source Jakarta project under the Apache software license [McClanahan00]. It has become one of the most widely used frameworks for building web applications utilizing the J2EE platform. The Struts framework is based on the Model-View-Controller (MVC) design paradigm using the Model 2 architecture. The functionality within Struts is centered on the controller portion of the MVC design paradigm.

In the Struts Framework, a Java Servlet acts as the (only) controller for the application. This servlet is known as the ActionServlet. It is responsible for receiving all requests from the clients of the application and acts as a single point of entry. Upon receiving a request it creates the appropriate Struts Action to handle the request. The Action will call upon other business objects to handle the request, and their response determines which page should be returned. A typical response sends data back to the presentation tier to be displayed to the user, often with the instantiation of a Java Server Page that is displayed in a browser. This process is represented in Figure 1.

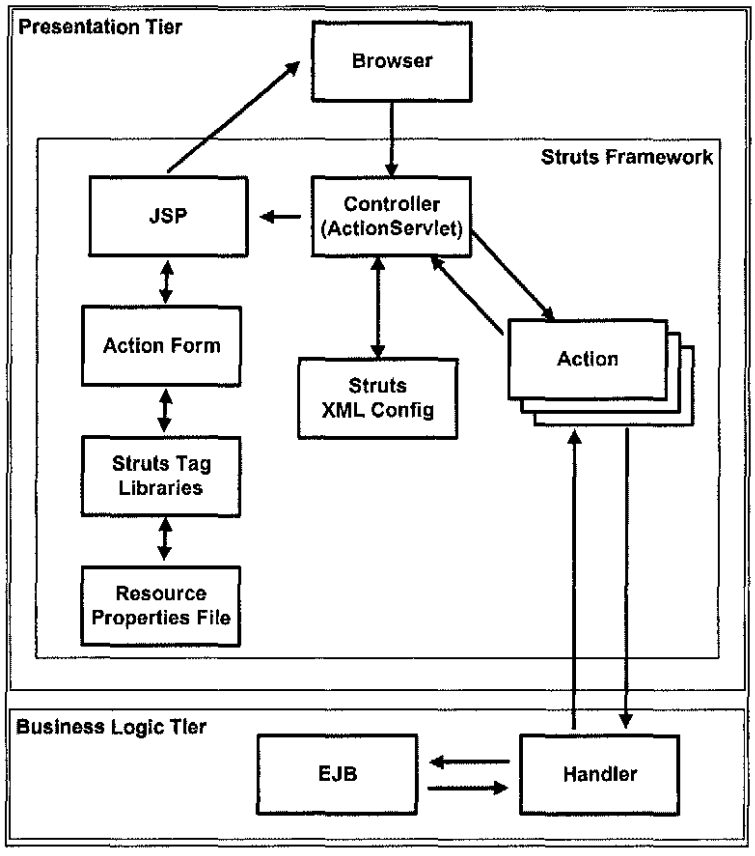

Figure 1: The Struts Framework

The focal point of all Struts applications is a collection of one or more Struts configuration files, which contain the mappings for each request that is made and the responses that can occur. A configuration file defines the request URL, the type of form submitted (ActionForm), and the class (Action) that will handle the request. The Struts ActionForm is a Java data transfer object with additional methods for data validation, population, and resetting the objects values. The ActionForm stores the data of the form submitted within an HTML or JSP and can be populated by an Action to transfer data back to the Java Server Page. The Struts Action determines what calls to make in the business tier and the appropriate response. Each response is known as an Action forward that maps the response to a Java Server Page to be displayed.

1.2 Associated Testing Frameworks

Testing frameworks have been developed to make unit and integration testing easy. Unit tests help ensure classes do what they are supposed to do in an isolated way, that is to say without taking into account interactions with other classes [Dudney03]. Integration testing takes into account interactions among classes. Struts based applications contain many Java classes that are built around the J2EE platform. The size and complexity of these applications call for frameworks to perform testing exercises on the implementation. Without an automated framework, testing this type of applications is difficult at best. The Struts Test Suite Generator described in Chapter 2 will utilize the JUnit, Cactus, and StrutsTestCase testing frameworks. The following sections will describe each of these frameworks and how they are used in the testing of Struts applications.

1.2.1 JUnit

JUnit is a simple framework for writing and running automated test for Java classes developed by Erich Gamma and Kent Beck [Gamma00] as an open source software project released under the Common Public License Version 1.0. Using an automated testing framework like JUnit is essential to a project's successful unit testing plan.

Since JUnit is nothing more that a Java API, in order to write a test using it, a subclass of junit.framework.TestCase is created. The test case contains methods used to setup the environment and create fixture objects, which will be used to test various methods

associated with the class under testing. After a test case has been exercised a teardown method returns the system back to its original state. Each test will assert conditions on the state of the system by calling methods of the Java classes. If an assertion fails, exceptions are generated and logged by the framework. The test cases can be grouped together in a test suite allowing multiple test cases to be run in one process. A test suite is a subclass of junit.framework.TestSuite. When the test suite runs, all test cases contained in the suite output their results to a log showing if the test passes or failed. If the test fails it logs which assertion caused the failure. Test logs can be used to review the results of tests and compared to results of tests performed repeatedly. The logs can then be reviewed and action taken where required. Additionally, JUnit provides developers with a GUI front end that shows a progress bar colored red if errors have been found and green otherwise. The front end summarizes the results by indicating the cases that pass and assertions that fail.

1.2.2 Cactus

While JUnit is a great tool for testing stand-alone Java classes, it falls short in the ability to test classes that interact with container objects. Cactus was developed to fill the gap between JUnit and server-side java code. Cactus is an open source Jakarta project under the auspices of the Apache Software Foundation [Seale01]. It is used to test applications that employ Java Servlets, Java Server Pages, Tag Libraries, Filters, Enterprise Java Beans, and other container objects. Cactus is an extension of JUnit supporting testing of server and web based applications. It performs two functions

first, acting as the client making a request, then within the server's container allowing the server environment to be setup to perform the test. Since Cactus is an extension of JUnit, test cases and test suites can also be created and all the features of JUnit are available (e.g. assertion checking predicates).

Struts applications are built around the ActionServlet, which is a container object. The container for a Struts application is the J2EE application server, where the application is deployed. In addition to the ActionServlet object; Struts applications often employ Tag Libraries, Filters, and may use Enterprise Java Beans in their business tiers. These characteristics make Cactus a good choice as part of an automated testing framework. Figure 2 shows how the Cactus framework interacts with a Struts application.

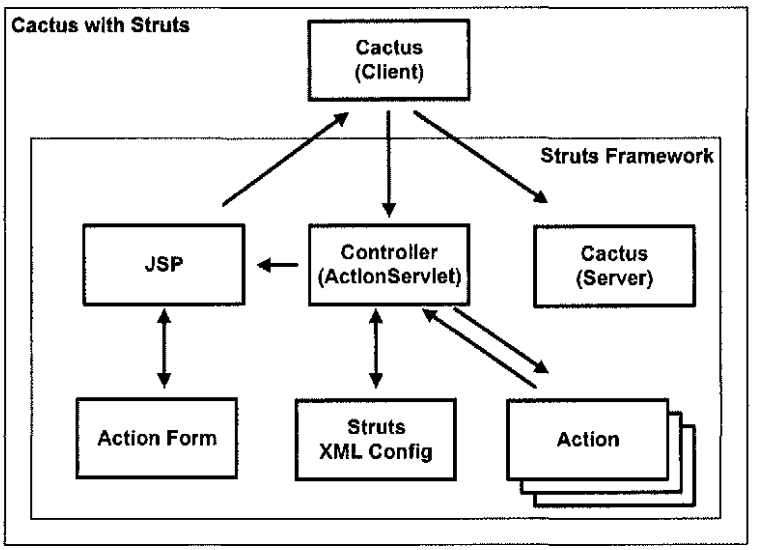

Figure 2: Cactus Interaction with Struts

1.2.3 StrutsTestCase

The StrutsTestCase framework is an open source software project that is an extension of the JUnit TestCase [Seale01]. It provides support for both mock servlet objects and Cactus container testing. The StrutsTestCase framework is important for testing Struts applications because it provides an API specific to testing the Struts framework. This project focuses on the CactusStrutsTestCase subclass of ServletTestCase, allowing container testing of Struts applications using the Cactus and JUnit frameworks. Using this approach, the Struts ActionServlet is running in the container and request can be made to test actions, mappings, action forms, and forward declarations. Chapter 2 describes how the tests for each of these are built based on the Struts configuration files.

Chapter 2

STRUTS TEST SUITE GENERATOR

This chapter will describe the components of the Struts test suite generator. It will cover the configuration files, translation rules, the Java Beans representing the test suite, the XML test suite description, and the file writers.

2.1 Test Suite Generator Architecture

The test suite generator architecture includes a Java application that takes the Struts configuration file as input and outputs a set of test cases that are subclasses of the CactusStrutsTestCase class. An intermediate step in the process is a test suite XML file (test-suite.xml) that describes the test suite. The XML file will contain each test case; describing the request under test, any parameters, and all assertions being performed. The intermediate file is parsed by the generator and produces the final Java classes as output. The test case Java classes are placed into the Cactus client and server projects allowing the test to be executed by the JUnit, Cactus, and StrutsTestCase frameworks. The test suite generator process is represented in Figure 3.

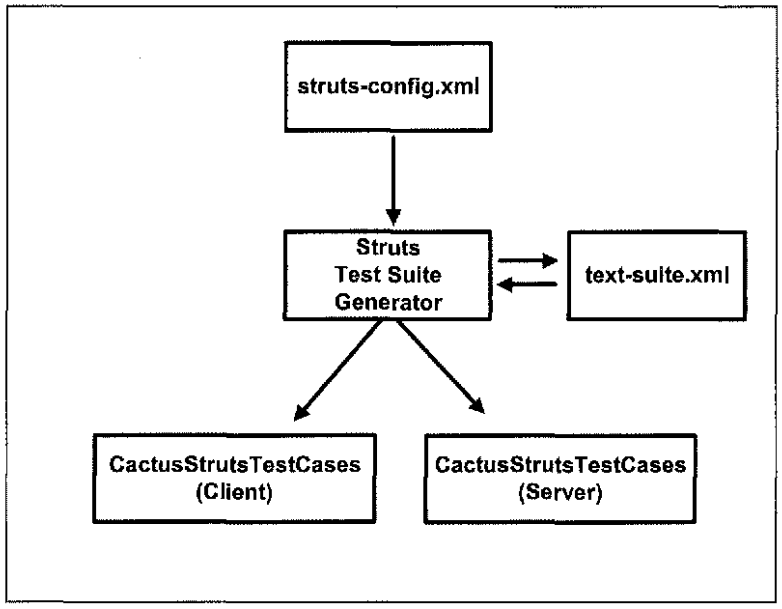

Figure 3: Struts Test Case Generator

2.2 Parsing the Struts Configuration File

The Struts framework uses one or more configuration files to load application specific components at startup. By default, there is only one Struts configuration file (struts-config.xml), but applications can be divided into multiple sub-applications with multiple configuration files. Each configuration file is a detailed map for the application containing the fields on the forms, where the JSP are located, every action that the application performs, and exactly what resources each of these actions need [Husted02]. When the application is started, the configuration files are parsed to create Java objects. Each element in the configuration file corresponds to a Java object. Figure 4 contains a list of the elements that are used by the generator. This list is a subset of elements in the Struts configuration file table taken from [Husted02].

Element	Description
<form-beans>	Describes the set of form bean descriptors for this application module.
<form-bean>	Describes an ActionForm subclass that can be referenced by an <action> element.
<action-mappings>	Describes a set of ActionMappings that are available to process requests that match the url-pattern of the ActionServlet registered with the container.
<action>	Describes an ActionMappings that is used to process a request for a specific module-relative URI.
<forward>	Describes an ActionForward that is to be made available to an Action as a return value.

Figure 4: The struts-config.xml elements used by the StrutsTestSuiteGenerator

The Struts framework uses the Commons Digester open source software project to parse the XML description and to translate it into Java objects [McClanahan99]. Once translated, the Struts configuration file is stored in a module configuration Java object (org.apache.struts.config.ModuleConfig). Sets of digester rules are created that map the XML to the module configuration. These rules are passed to the digester with the Struts configuration file to create the module configuration Java object.

Our Struts Test Suite Generator uses the information in the Struts configuration file to build the test cases. By importing the Struts digester rules and portions of the Struts framework, the generator is able to create a module configuration Java object. This object is then used to directly access properties of the Struts configuration file by calling its methods directly. In order for the generator to use portions of the Struts framework, the Struts JAR file must be included in the class path of the project. The Commons Digester framework is also required in the generator's class path, allowing

```
<struts-config>
...
<form-beans>
    <form-bean name="directorySearchForm"
       type="com.jacg0002.tds.action_forms.DirectorySearchForm">
    </form-bean>
</form-beans>

<action-mappings>
    <action name="directorySearchForm"
            path="/directorySearch"
            scope="request"
            type="com.jacg0002.tds.actions.DirectorySearchAction"
            input="/phonesearch.jsp"
            validate="true">
        <forward name="success" path="/searchresults.jsp"/>
        <forward name="fail" path="/phonesearch.jsp"/>
    </action>
    <action name="directorySearchForm"
            path="/newSearch"
            scope="request"
            type="com.jacg0002.tds.actions.NewSearchAction"
            input="/searchresults.jsp"
            validate="false">
        <forward name="success" path="/phonesearch.jsp"/>
    </action>
</action-mappings>
...
</struts-config>
```

Figure 5: A sample Struts configuration file (struts-config.xml)

it to be used to parse the Struts configuration file. A sample Struts configuration file is shown in Figure 5.

2.3 Test Suite Java Beans

The Struts Test Suite Generator uses a set of Java Beans to represent the test suite. The beans are created based on information extracted from the Struts configuration file. The test suite contains a collection of test cases, representing each request made to the Struts application. Each test case bean contains its package name, class name, request path, action type, action form name, action form type, and a collection of test methods. The test method bean contains the method name, forward name, forward path, a collection of assertion beans, and a collection of parameter beans. The assertion bean describes the assertions being performed and the parameter bean describes the parameters used when requesting the action. Figure 6 is a class diagram for the test suite Java beans.

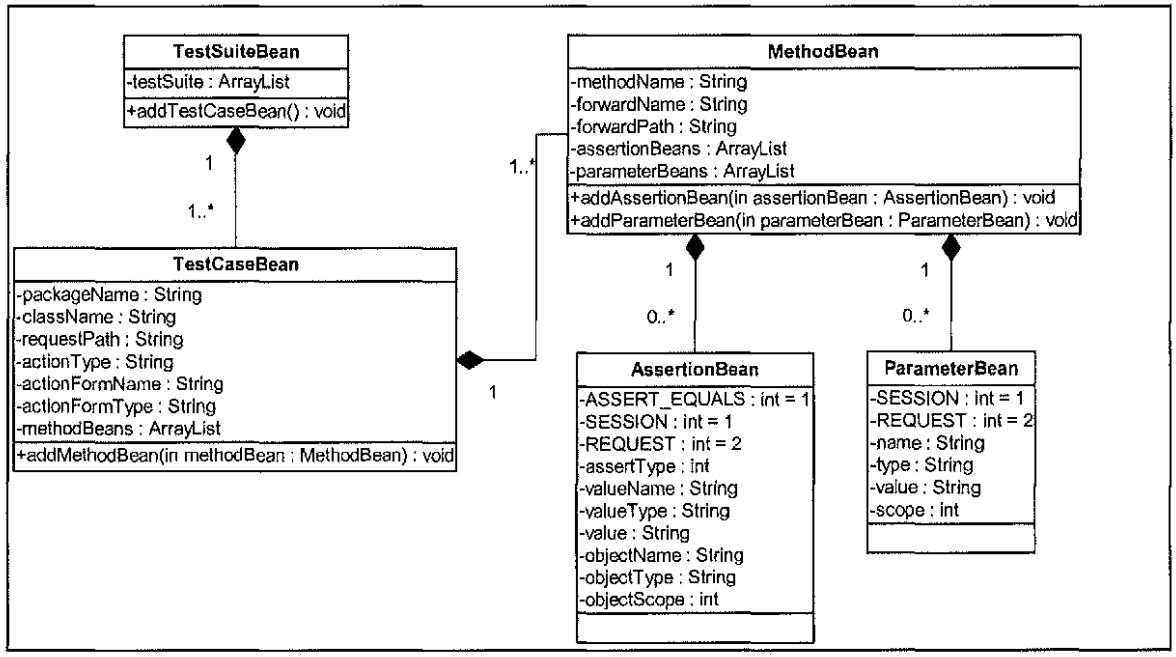

Figure 6: Test suite Java beans class diagram

2.4 Translation Rules

Building the test suite beans requires a set of translation rules to map the elements in the Struts configuration file to the beans. The following rules are applied to build the test suite Java beans:

- Each *action* element creates a new TestCaseBean.
- The *path* attribute of the *action* is used to create the TestCaseBean class name (i.e.: "/searchDirectory" would create a class name "SearchDirectoryTest") and is also used to set the request path used by the test case to make a request to the application.
- The test case package is set based on the package name supplied to the generator.

- The *type* attribute of the *action* is used to set the action type (the fully qualified Java classname of the Action subclass.)
- The *name* attribute of the *action* is used to set the action form name, which is used to lookup the ActionForm subclass.
- The *form-bean* element is found based on the action form name and its *type* attribute is used to set the action form type (the fully qualified Java classname of the ActionForm subclass).
- Each *forward* element found with the *action* element creates a MethodBean within the TestCaseBean.
- The *name* attribute of the *forward* element is used to create the method name (i.e.: "success" would create a method name of "testSuccess") and is used to set the forward name.
- The *path* attribute of the *forward* element is used to set the forward path.

These translation rules allow a test case to be created for each action mapping and a test method for each forward mapping. This simple combination creates a test for each path through the Struts application.

2.5 Test Suite XML Description

An intermediate step in the creation of the test suite and test cases is an XML file that describes the test suite. In addition to containing a description of the test suite, the XML file supports the addition of more methods, parameters, and assertions. This allows the generator to produce their corresponding test case Java classes. This is

important since the generator does not have the required information to know all the parameters and assertions needed to accurately test an application.

The conversion of the test suite beans to XML is accomplished using the Betwixt open source software project [Strachan02]. The Betwixt library provides an XML introspection mechanism for serializing the beans to XML. The generator utilizes the Betwixt libraries to produce the test suite XML file (test-suite.xml) and to de-serialize the XML file back into the test suite beans.

2.6 Test Suite and Test Case Writers

Two Java classes are used to output the test suite and test case Java files. The TestCaseWriter class takes a test case bean and creates the CactusStrutsTestCase subclass code. The test case includes the constructor, a main method to run the test using the JUnit Swing user interface, a setup and teardown method, and a test method for each method bean. The TestSuiteWriter class takes a test suite bean and creates a Java class named AllTest. The AllTest class contains a JUnit TestSuite that contains all the CactusStrutsTestCase subclasses and a main method to run the test suite using the JUnit Swing user interface.

2.7 Sequence of Events

To create the test suite XML file, the Struts Test Suite Generator first parses the Struts configuration file. The StrutsConfigDigester class produces the ModuleConfig Java object. Applying the translation rules to the ModuleConfig creates the test suite and test case Java beans. The beans are then passed to the Betwixt bean writer to create the test suite XML file. Figure 7 shows the sequence diagram for generating the test-suite.xml file.

Creating the CactusStrutsTestCase subclasses starts by reading the test suite XML file. The Betwixt bean reader performs all the work to create the test case and test suite beans. The beans are then passed to the TestCaseWriter and TestSuiteWriter to produce the final output as shown in Figure 8.

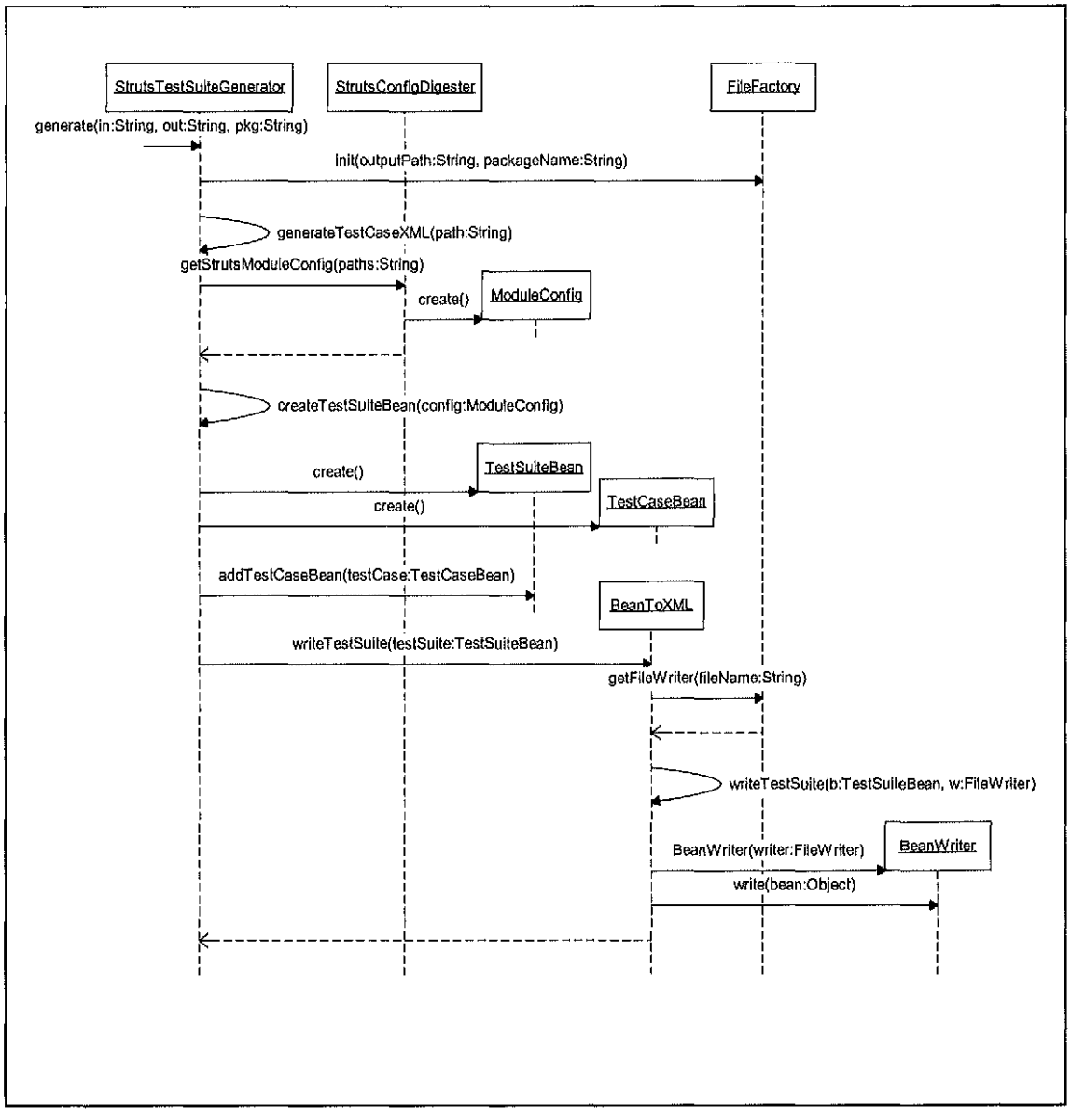

Figure 7: Sequence diagram for generating the test-suite.xml file

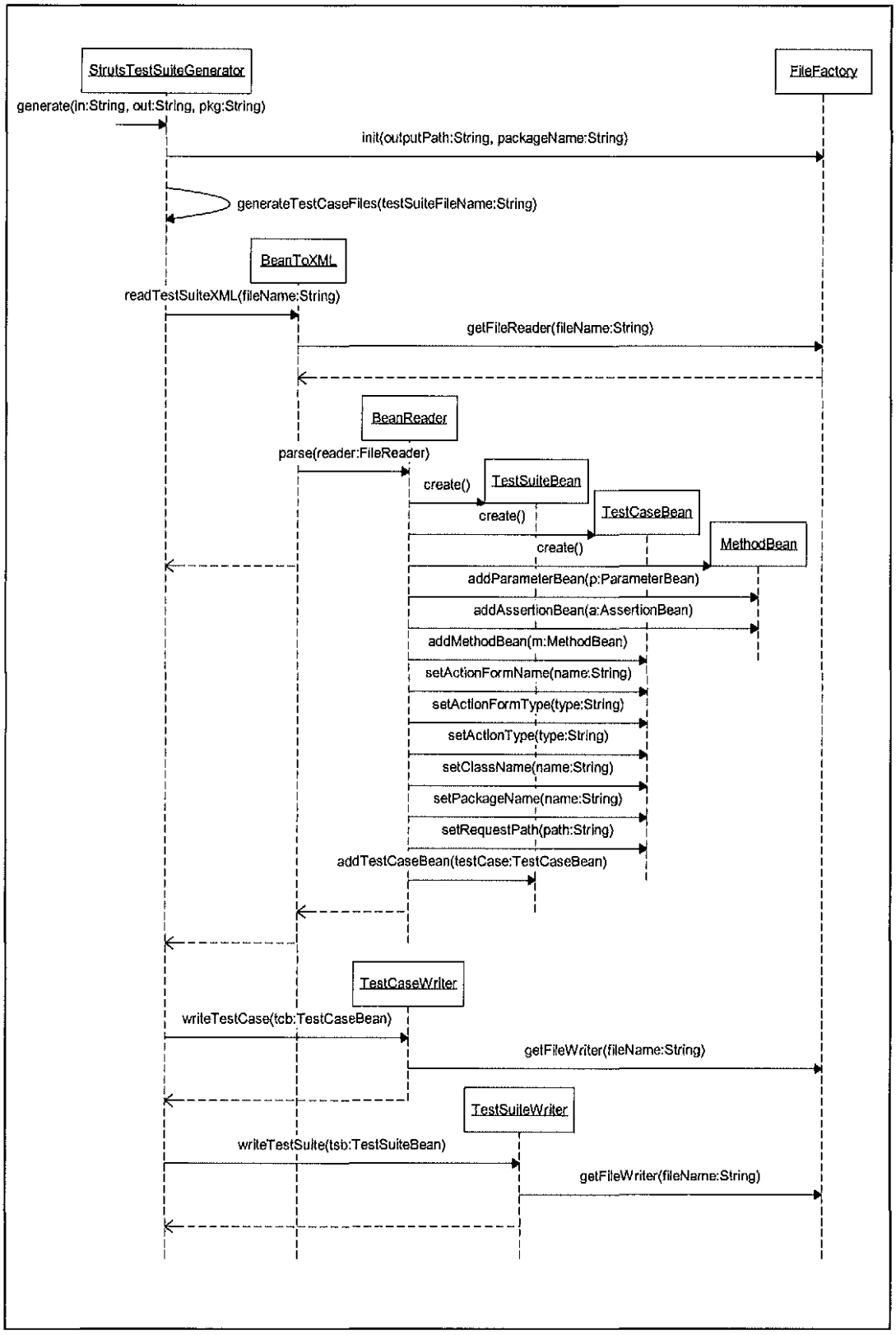

Figure 8: Sequence diagram for generating test case files

Chapter 3

RESULTS

3.1 StrutsTestSuiteGenerator Input

The input into the Struts Test Suite Generator includes a Struts configuration, the output directory, and the package name for the generated Java classes. The generator is run by passing the arguments to the main method of the StrutsTestSuiteGenerator class. The first argument is the fully qualified path of the Struts configuration file. The second argument is the fully qualified output directory. The directory must exist and the user account must have write access for the directory. The third argument is the package name. A sample set of arguments is listed below:

```
java StrutsTestSuiteGenerator C:\MyProject\WEB-INF\struts-config.xml
C:\MyProject\src  com.myproject.test
```

These arguments will be passed to the generate method of the Struts Test Suite Generator to create the test suite and test cases. After the classes are generated the application will terminate. A complete set of installation and execution instructions are located in Appendix B.

3.2 StrutsTestSuiteGenerator Output

The generator validates that the output directory exist then creates the package directory structure. The test suite XML file (test-suite.xml) is created in the package directory. A sample file is shown in Figure 9. Next the CactusStrutsTestCase subclasses and the Java class (AllTests.java) containing the test suite is output in the package directory. A sample test case is shown in Figure 10. A sample Struts configuration file and generated output is located in Appendix C.

```
<?xml version='1.0' ?>
<TestSuiteBean>
   <testCaseBeans>
      <testCaseBean>
         <packageName>edu.unf.jacg0002.cactus</packageName>
         <actionType> com.jacg0002.tds.actions.DirectorySearchAction
         </actionType>
         <requestPath>/directorySearch</requestPath>
         <actionFormName>directorySearchForm</actionFormName>
         <className>DirectorySearchTest</className>
         <actionFormType>
            com.jacg0002.tds.action_forms.DirectorySearchForm
         </actionFormType>
         <methodBeans>
            <methodBean>
               <forwardName>fail</forwardName>
               <methodName>testFail</methodName>
               <forwardPath>/phonesearch.jsp</forwardPath>
               <assertionBeans></assertionBeans>
               <parameterBeans></parameterBeans>
            </methodBean>
            <methodBean>
               <forwardName>success</forwardName>
               <methodName>testSuccess</methodName>
               <forwardPath>/searchresults.jsp</forwardPath>
               <assertionBeans></assertionBeans>
               <parameterBeans></parameterBeans>
            </methodBean>
         </methodBeans>
      </testCaseBean>
   </testCaseBeans>
</TestSuiteBean>
```

Figure 9: Sample test-suite.xml file

```
package edu.unf.jacg0002.cactus;
import servletunit.struts.CactusStrutsTestCase;

public class DirectorySearchTest extends CactusStrutsTestCase {

      public DirectorySearchTest(String testName){
            super(testName);
      }

      public static void main(String[] args){
           junit.swingui.TestRunner.run(DirectorySearchTest.class);
      }

      public void setUp() throws Exception {
            super.setUp();
      }

      public void tearDown() throws Exception {
            super.tearDown();
      }

      public void testFail() {
            setRequestPathInfo("/directorySearch");
            actionPerform();
            verifyForward("fail");
            verifyNoActionErrors();
      }

      public void testSuccess() {
            setRequestPathInfo("/directorySearch");
            actionPerform();
            verifyForward("success");
            verifyNoActionErrors();
      }
}
```

Figure 10: A generated test case

3.3 Execution of Generated Test

The generated test cases are now ready to be executed. While a JUnit test case requires little or no additional setup, Cactus test cases require the project under test to

be deployed and running in the container. There are two approaches for the server side testing code. One is to copy the generated test cases into the project's source code folder and re-deploy the project to the container. The second option is to create a separate project containing the test cases with references to the project under test. Using either option, a client project is created that also contains the generated test cases. The setup of a project to support the Cactus and StrutsTestCase frameworks is well documented on their corresponding project's web sites and in books like "Java Tools for extreme Programming: Mastering Open Source Tools including Ant, JUnit, and Cactus" [Hightower02]. Once the Cactus client project and server-side project are configured to support the Cactus and StrutsTestCase frameworks, the application is started and the test cases are launched using the main class in the test suite. The JUnit framework will log the test results and display each test cases success or failure.

Chapter 4

CONCLUSIONS

Today's Web applications undergo maintenance at a faster rate than other software systems, typically small incremental changes. Automated testing is critical for these small changes to be successful and to keep the application from losing previous functionality. The time spent creating automated test is returned many times over during regression testing of the application. The Struts Test Case Generator reduces the time a developer must spend to create the automated test classes. Less time is consumed reviewing the configuration of an application and creating the test case classes, allowing the focus on the data parameters and assertions required to fully test an application. Large applications have many request paths and possible forward mappings to Java Server Pages. Generating the test cases for these applications will save precious time during the development process. In addition to saving the time creating the test cases, the developer has greater assurance that all paths are covered using the generator.

Chapter 5

FUTURE DEVELOPMENTS

The generator is just the first step towards creating a complete Struts testing plug-in for the Eclipse platform. While this generator will allow the creation of simple test cases for each path in the Struts application, more complex test cases can be generated using reflection on the Action Forms. Adding this ability would allow the generator to add parameters that are passed as part of the request. A second feature would be a GUI interface for specifying additional assertions and test cases. This feature would allow the generator to first create an initial set of test cases. Each test case could have additional assertions added based on the available assertions within the StrutsTestCase, Cactus, and JUnit frameworks. If additional test cases are required for a single path, test cases could be copied and modified using the interface. In addition to adding the ability to modify the test cases, enabling the wizard to plug-in to the Eclipse platform would make the generator easier to use and encourage developers to take advantage of the testing frameworks.

REFERENCES

[Cavaness03]
Cavaness, C., Programming Jakarta Struts, O'Reilly, Sebastapol, 2003, pp.11.

[Dudney03]
Bill Dudney, Jonathan Lehr, Jakarta Pitfalls. Time-Saving Solutions for Struts, Ant, JUnit, and Cactus, Wiley Publishing, Inc., 2003.

[Gamma00]
Erich Gammam Kent Beck, JUnit Project, http://sourceforge.net/projects/junit, 2000.

[Hightower02]
Richard Hightower, Nicholas Lesiecki, Java Tools for eXtreme Programming. Mastering Open Source Tools Including Ant, JUnit, and Cactus, Wiley Computer Publishing, 2002.

[Husted02]
Husted, T., Struts in Action: Building Web Applications with the Leading Java Framework, Manning Publications Company, 2002.

[Massol00]
Vincent Massol, Cactus Project, http://jakarta.apache.org/cactus/, 2000.

[McClanahan99]
Craig McClanahan, Scott Sanders, Jean-Francois Arcand, Commons Digester Project, http://jakarta.apache.org/commons/digester, 1999.

[McClanahan00]
Craig McClanahan, Struts Project, http://jakarta.apache.org/struts/, 2000.

[Seale01]
StrutsTestCase Project, http://sourceforge.net/projects/strutstestcase, 2001

[Strachan02]
James Strachan, Robert Donkin, Martin van den Bemt, Commons Betwixt Project, http://jakarta.apache.org/commons/betwixt, 1999.

APPENDIX A

(Note: Source code was formatted to fit within the page margins. To compile, some wrapped lines will need to be reformatted)

Source Code: TestCaseGenerator.java

```
package edu.unf.jacg0002;

import java.util.Iterator;

import org.apache.struts.config.ActionConfig;
import org.apache.struts.config.FormBeanConfig;
import org.apache.struts.config.ForwardConfig;
import org.apache.struts.config.ModuleConfig;

/**
 * The StrutsTestSuiteGenerator was created in support of a project
 * submitted to the University of North Florida Department of
 * Computer and Information Sciences in partial fulfillment of the
 * requirement for the degree of Masters of Science in Computer
 * and Information Science by Gregory M. Jackson in April 2004.
 *
 *
 * @author Greg Jackson
 * @version 0.1
 */
public class StrutsTestSuiteGenerator
{
      /**
       * Launches the StrutsTestSuiteGenerator
       * Usage:java Main strutsConfigFile outputDirectory packageName
       * args[0]: strutsConfigFile - The fully qualified path and
       *                                  file name of the
       *                             struts-config.xml file
       * args[1]: outputDirectory  - The output directory for the
       *                             generated files
       * args[2]: packageName      - The package for the generated
       *                             classes
       *
       * Calls generate with the supplied parameters.
       */
      public static void main(String[] args)
      {
            if(args.length != 3)
            {
                  System.out.println("Usage: java Main
                      strutsConfigFile outputDirectory packageName");
                  System.out.println("strutsConfigFile - The path
                      and file name of the struts-config.xml file");
                  System.out.println("                   - i.e.:
                      C:\\temp\\struts-config.xml");
                  System.out.println("outputDirectory  - The output
                      directory for the generated files");
                  System.out.println("                   - i.e.:
```

```
                C:\\temp\\testcases");
            System.out.println("packageName      - The
                package for the generated classes");
            System.out.println("                 - i.e.:
                com.mycompany.test");
            System.exit(1);
        }
        String strutsConfigFile = args[0];
        String outputDirectory = args[1];
        String packageName = args[2];
        try
        {
            StrutsTestSuiteGenerator strutsTestSuiteGenerator =
                new StrutsTestSuiteGenerator();
            strutsTestSuiteGenerator.generate(strutsConfigFile,
                outputDirectory, packageName);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
}

/**
 * Launches the StrutsTestSuiteGenerator
 * Generates the directory for the packageName in the
 * outputDirectory.  Creates TestCases and a TestSuite
 * based on the strutsConfigFile.
 * @param strutsConfigFile - The fully qualified path
 * and file name of the struts-config.xml file
 * @param outputDirectory - The output directory for
 * the generated files
 * @param packageName - The package for the
 * generated classes
 */
public void generate(String inputFilePath,
                     String outputDirPath,
                     String packageName)
                     throws Exception
{
      FileFactory.init(outputDirPath, packageName);
      generateTestCaseXML(inputFilePath);
      generateTestCaseFiles(BeanToXML.TEST_SUITE_FILE_NAME);
}


/**
 * Converts the struts-config.xml file into TestCases.
 * Outputs the TestCases to test-cases.xml.
 */
private void generateTestCaseXML(String strutsConfigFilePath)
                                 throws Exception
{
      StrutsConfigDigester dig = new StrutsConfigDigester();
      ModuleConfig moduleConfig =
          dig.getStrutsModuleConfig(strutsConfigFilePath);

      TestSuiteBean testSuiteBean =
          createTestSuiteBean(moduleConfig);

      BeanToXML btx = new BeanToXML();
      btx.writeTestSuite(testSuiteBean);
}
```

```
/**
 * Reads in the test-suite.xml file and creates
 * TestCase java files.
 */
private void generateTestCaseFiles(String testSuiteFileName)
                                        throws Exception
{
      BeanToXML btx = new BeanToXML();
      TestSuiteBean testSuiteBean =
          btx.readTestSuiteXML(testSuiteFileName);

      Iterator it =
          testSuiteBean.getTestCaseBeans().iterator();
      while(it.hasNext())
      {
            TestCaseBean testCaseBean =
               (TestCaseBean) it.next();
            TestCaseWriter testCaseWriter =
                new TestCaseWriter();
            testCaseWriter.writeTestCase(testCaseBean);
      }

      TestSuiteWriter testSuiteWriter = new TestSuiteWriter();
      testSuiteWriter.writeTestSuite(testSuiteBean);
}


/**
 * Builds the TestSuiteBean based on the translation rules.
 */
private TestSuiteBean createTestSuiteBean(ModuleConfig
                        moduleConfig) throws Exception
{
      TestSuiteBean testSuiteBean = new TestSuiteBean();
      ActionConfig[] actionConfigs =
          moduleConfig.findActionConfigs();

      for(int i=0; i<actionConfigs.length; i++)
      {
            TestCaseBean testCaseBean = new TestCaseBean();

            testCaseBean.setClassName(buildClassName
               (actionConfigs[i].getPath()));
            testCaseBean.setPackageName(
                FileFactory.getPackageName());
            testCaseBean.setRequestPath(
                actionConfigs[i].getPath());
            testCaseBean.setActionType(
                actionConfigs[i].getType());
             testCaseBean.setActionFormName(
                actionConfigs[i].getName());
            FormBeanConfig formBeanConfig =
                moduleConfig.findFormBeanConfig(
                actionConfigs[i].getName());
            if(formBeanConfig!=null)
                  testCaseBean.setActionFormType(
                      formBeanConfig.getType());

            ForwardConfig[] forwardConfigs =
                actionConfigs[i].findForwardConfigs();
            if(forwardConfigs.length == 0)
            {
```

```
                        MethodBean methodBean = new MethodBean();
                        methodBean.setMethodName("test");
                        testCaseBean.addMethodBean(methodBean);
                }
                else
                {
                        for(int x=0; x<forwardConfigs.length; x++)
                        {
                                MethodBean methodBean =
                                    new MethodBean();
                                methodBean.setMethodName(
                                    buildMethodName(
                                    forwardConfigs[x].getName()));
                                methodBean.setForwardName(
                                    forwardConfigs[x].getName());
                                methodBean.setForwardPath(
                                    forwardConfigs[x].getPath());
                                testCaseBean.addMethodBean(methodBean);
                        }
                }

                testSuiteBean.addTestCaseBean(testCaseBean);
        }

        return testSuiteBean;
    }

    private String buildClassName(String requestPathInfo)
    {
        String className = "";
        if(requestPathInfo != null && requestPathInfo.length()>1)
        {
                className = className +
                    requestPathInfo.substring(1,2).toUpperCase();
                if(requestPathInfo.length()>2)
                        className = className +
                            requestPathInfo.substring(
                            2,requestPathInfo.length());
        }
        className = className + "Test";

        return className;
    }

    private String buildMethodName(String forward)
    {
        String methodName = "test";
        if(forward!=null && forward.length()>0)
        {
                methodName = methodName +
                    forward.substring(0,1).toUpperCase();
                if(forward.length()>1)
                        methodName = methodName +
                            forward.substring(1);
        }
        return methodName;
    }


} // *** END StrutsTestSuiteGenerator.java **************************
```

Source Code: StrutsConfigDigester.java

```
package edu.unf.jacg0002;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.MalformedURLException;
import java.net.URL;

import javax.servlet.UnavailableException;

import org.apache.commons.digester.Digester;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.struts.action.ActionServlet;
import org.apache.struts.config.ConfigRuleSet;
import org.apache.struts.config.ModuleConfig;
import org.apache.struts.config.ModuleConfigFactory;
import org.apache.struts.util.MessageResources;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;

/**
 * Modified version of the Struts project StrutsConfigDigester.
 * Return a ModuleConfig object to a Java application
 * when passed the path of the Struts configuration file.
 *
 * @author Greg Jackson
 */
public class StrutsConfigDigester
{
    /**
     * Comma-separated list of context-relative path(s) to
     * our configuration resource(s) for the default module.
     */
    protected String config = "/WEB-INF/struts-config.xml";


    /**
     * The Digester used to produce ModuleConfig objects from a
     * Struts configuration file.
     * @since Struts 1.1
     */
    protected Digester configDigester = null;

    /**
     * The resources object for our internal resources.
     */
    protected MessageResources internal = null;


    /**
     * The Java base name of our internal resources.
     * @since Struts 1.1
     */
    protected String internalName =
       "org.apache.struts.action.ActionResources";
```

```
/**
 * Commons Logging instance.
 * @since Struts 1.1
 */
protected static Log log =
    LogFactory.getLog(ActionServlet.class);


/**
 * The set of public identifiers, and corresponding
 * resource names, for the versions of the configuration
 * file DTDs that we know about.
 * There <strong>MUST</strong> be an even number of
 * Strings in this list!
 */
protected String registrations[] = {
    "-//Apache Software Foundation//DTD Struts Configuration
        1.0//EN",
    "/org/apache/struts/resources/struts-config_1_0.dtd",
    "-//Apache Software Foundation//DTD Struts Configuration
        1.1//EN",
    "/org/apache/struts/resources/struts-config_1_1.dtd",
    "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN",
    "/org/apache/struts/resources/web-app_2_2.dtd",
    "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN",
    "/org/apache/struts/resources/web-app_2_3.dtd"
};


  /**
   * Builds a ModuleConfig based on the struts-config.xml file
   * supplied in the path.
   *
   * @param paths, The path of the struts configuration file.
   * @return ModuleConfig
   */
  public ModuleConfig getStrutsModuleConfig(String paths)
         throws Exception
  {
    config = paths;
    ModuleConfig moduleConfig = initModuleConfig("", config);
    moduleConfig.freeze();
    destroyConfigDigester();
    return moduleConfig;
  }

protected ModuleConfig initModuleConfig(String prefix,
                                        String paths)
                                        throws Exception
{
    // Parse the configuration for this module
    ModuleConfigFactory factoryObject =
        ModuleConfigFactory.createFactory();
    ModuleConfig config =
        factoryObject.createModuleConfig(prefix);


    // Configure the Digester instance we will use
    Digester digester = initConfigDigester();

    // Process each specified resource path
    while (paths.length() > 0) {
```

```
            digester.push(config);
            String path = null;
            int comma = paths.indexOf(',');
            if (comma >= 0) {
                path = paths.substring(0, comma).trim();
                paths = paths.substring(comma + 1);
            } else {
                path = paths.trim();
                paths = "";
            }

            if (path.length() < 1) {
                break;
            }

            this.parseModuleConfigFile(prefix, paths, config,
                                       digester, path);
        }

        return (config);

    }

    /**
     * Parses one module config file.
     * @param prefix
     * @param paths
     * @param config
     * @param digester Digester instance that does the parsing
     * @param path The path to the config file to parse.
     * @throws UnavailableException
     */
    private void parseModuleConfigFile(
        String prefix,
        String paths,
        ModuleConfig config,
        Digester digester,
        String path)
        throws Exception {

        InputStream input = null;
        try {
            InputSource is = new InputSource();
            File file = new File(path);
            input = new FileInputStream(file);
            is.setByteStream(input);
            digester.parse(is);
        } catch (MalformedURLException e) {
            handleConfigException(paths, e);
        } catch (IOException e) {
            handleConfigException(paths, e);
        } catch (SAXException e) {
            handleConfigException(paths, e);
        } finally {
            if (input != null) {
                try {
                    input.close();
                } catch (IOException e) {
                    throw new UnavailableException(e.getMessage());
                }
            }
        }
    }
```

```
    /**
     * Simplifies exception handling in the parseModuleConfigFile()
     * method.
     * @param paths
     * @param e
     * @throws UnavailableException
     */
    private void handleConfigException(String paths, Exception e)
        throws Exception {
        log.error(internal.getMessage("configParse", paths), e);
        throw new Exception(internal.getMessage("configParse",
                            paths));
    }

    /**
     * <p>Create (if needed) and return a new Digester instance that
     * has been
     * initialized to process Struts module configuraiton files and
     * configure a corresponding ModuleConfig object (which must be
     * pushed on to the evaluation stack before parsing begins).</p>
     *
     * @exception ServletException if a Digester cannot be configured
     * @since Struts 1.1
     */
    protected Digester initConfigDigester()
    {

        // Do we have an existing instance?
        if (configDigester != null) {
            return (configDigester);
        }
       boolean validating = true;

        // Create a new Digester instance with standard capabilities
        configDigester = new Digester();
        configDigester.setNamespaceAware(true);
        configDigester.setValidating(validating);
        configDigester.setUseContextClassLoader(true);
        configDigester.addRuleSet(new ConfigRuleSet());
        for (int i = 0; i < registrations.length; i += 2) {
            URL url= this.getClass().getResource(registrations[i+1]);
            if (url != null) {
                configDigester.register(registrations[i],
                                        url.toString());
            }
        }

        // Return the completely configured Digester instance
        return (configDigester);
    }

    /**
     * Gracefully release any configDigester instance that
     * we have created.
     * @since Struts 1.1
     */
    protected void destroyConfigDigester() {

        configDigester = null;
    }

} // *** END StrutsConfigDigester.java  ****************************
```

Source Code: TestSuiteBean.java

```
package edu.unf.jacg0002;

import java.util.ArrayList;

/**
 * Contains all the TestCaseBeans in the TestSuite
 *
 * @author Greg Jackson
 */
public class TestSuiteBean
{
	private ArrayList testSuite;

	public TestSuiteBean()
	{
		testSuite = new ArrayList();
	}

	/**
	 * Adds the TestCaseBean to the testSuite ArrayList.
	 * @param testCaseBean
	 */
	public void addTestCaseBean(TestCaseBean testCaseBean)
	{
		if(testCaseBean != null)
			testSuite.add(testCaseBean);
	}

	/**
	 * @return ArrayList of edu.unf.jacg0002.TestCaseBean objects.
	 */
	public ArrayList getTestCaseBeans()
	{
		return testSuite;
	}

} // *** END TestSuiteBean.java  ************************************
```

Source Code: TestCaseBean.java

```
package edu.unf.jacg0002;

import java.util.ArrayList;

/**
 * Represents a TestCase, containing all required
 * information to write the a TestCase class.
 *
 * @author Greg Jackson
 */
public class TestCaseBean
{
	private String packageName;
	private String className;
	private String requestPath;
	private String actionType;
	private String actionFormName;
	private String actionFormType;
	private ArrayList methodBeans;

	public TestCaseBean()
	{
		methodBeans = new ArrayList();
	}

	/**
	 * Adds a MethodBean to the TestCaseBean
	 */
	public void addMethodBean(MethodBean methodBean)
	{
		if(methodBean!=null)
			methodBeans.add(methodBean);
	}

	/**
	 * Returns the actionFormName.
	 * @return String
	 */
	public String getActionFormName()
	{
		return actionFormName;
	}

	/**
	 * Returns the actionFormType.
	 * @return String
	 */
	public String getActionFormType()
	{
		return actionFormType;
	}

	/**
	 * Returns the actionType.
	 * @return String
	 */
	public String getActionType()
	{
		return actionType;
```

```
}

/**
 * Returns the methodBeans.
 * @return ArrayList
 */
public ArrayList getMethodBeans()
{
      return methodBeans;
}

/**
 * Returns the requestPath.
 * @return String
 */
public String getRequestPath()
{
      return requestPath;
}

/**
 * Sets the actionFormName.
 * @param actionFormName The actionFormName to set
 */
public void setActionFormName(String actionFormName)
{
      this.actionFormName = actionFormName;
}

/**
 * Sets the actionFormType.
 * @param actionFormType The actionFormType to set
 */
public void setActionFormType(String actionFormType)
{
      this.actionFormType = actionFormType;
}

/**
 * Sets the actionType.
 * @param actionType The actionType to set
 */
public void setActionType(String actionType)
{
      this.actionType = actionType;
}

/**
 * Sets the methodBeans.
 * @param methodBeans The methodBeans to set
 */
public void setMethodBeans(ArrayList methodBeans)
{
      this.methodBeans = methodBeans;
}

/**
 * Sets the requestPath.
 * @param requestPath The requestPath to set
 */
public void setRequestPath(String requestPath)
{
      this.requestPath = requestPath;
```

```
        }

        /**
         * Returns the className.
         * @return String
         */
        public String getClassName()
        {
              return className;
        }

        /**
         * Returns the packageName.
         * @return String
         */
        public String getPackageName()
        {
              return packageName;
        }

        /**
         * Sets the className.
         * @param className The className to set
         */
        public void setClassName(String className)
        {
              this.className = className;
        }

        /**
         * Sets the packageName.
         * @param packageName The packageName to set
         */
        public void setPackageName(String packageName)
        {
              this.packageName = packageName;
        }

} // *** END TestCaseBean.java  ************************************
```

Source Code: MethodBean.java

```
package edu.unf.jacg0002;

import java.util.ArrayList;

/**
 * Represents a testXXX() methods in a TestCase
 *
 * @author Greg Jackson
 */
public class MethodBean
{
	private String methodName;
	private String forwardName;
	private String forwardPath;
	private ArrayList assertionBeans;
	private ArrayList parameterBeans;

	/**
	 * Constructor
	 * Initializes assertionBeans and paramterBeans collections.
	 */
	public MethodBean()
	{
		assertionBeans = new ArrayList();
		parameterBeans = new ArrayList();
	}

	/**
	 * Adds a AssertionBean to the assertionBeans collection.
	 */
	public void addAssertionBean(AssertionBean assertionBean)
	{
		if(assertionBean!=null)
			assertionBeans.add(assertionBean);
	}

	/**
	 * Adds a ParamterBean to the parameterBeans collection.
	 */
	public void addParameterBean(ParameterBean parameterBean)
	{
		if(parameterBean!=null)
			parameterBeans.add(parameterBean);
	}

	/**
	 * Returns the assertionBeans.
	 * @return ArrayList
	 */
	public ArrayList getAssertionBeans()
	{
		return assertionBeans;
	}

	/**
	 * Returns the forwardName.
	 * @return String
	 */
	public String getForwardName()
```

```
{
      return forwardName;
}

/**
 * Returns the forwardPath.
 * @return String
 */
public String getForwardPath()
{
      return forwardPath;
}

/**
 * Returns the methodName.
 * @return String
 */
public String getMethodName()
{
      return methodName;
}

/**
 * Returns the parameterBeans.
 * @return ArrayList
 */
public ArrayList getParameterBeans()
{
      return parameterBeans;
}

/**
 * Sets the assertionBeans.
 * @param assertionBeans The assertionBeans to set
 */
public void setAssertionBeans(ArrayList assertionBeans)
{
      this.assertionBeans = assertionBeans;
}

/**
 * Sets the forwardName.
 * @param forwardName The forwardName to set
 */
public void setForwardName(String forwardName)
{
      this.forwardName = forwardName;
}

/**
 * Sets the forwardPath.
 * @param forwardPath The forwardPath to set
 */
public void setForwardPath(String forwardPath)
{
      this.forwardPath = forwardPath;
}

/**
 * Sets the methodName.
 * @param methodName The methodName to set
 */
public void setMethodName(String methodName)
```

```
        {
                this.methodName = methodName;
        }

        /**
         * Sets the parameterBeans.
         * @param parameterBeans The parameterBeans to set
         */
        public void setParameterBeans(ArrayList parameterBeans)
        {
                this.parameterBeans = parameterBeans;
        }

} // *** END MethodBean.java ***************************************
```

Source Code: ParameterBean.java

```
package edu.unf.jacg0002;

/**
 * Represents a parameter used by a MethodBean.
 *
 * @author Greg Jackson
 */
public class ParameterBean
{
	public static final int SESSION = 1;
	public static final int REQUEST = 2;

	private String name;
	private String type;
	private String value;
	private int scope;

	/**
	 * Constructor
	 */
	public ParameterBean() {}

	/**
	 * Constructor with all required fields.
	 * @param name, the variable name used to define the parameter.
	 * @param type, the fully qualified class name for the
	 * parameter.
	 * @param value, the string value of the parameter.
	 * @param scope, the scope of the parameter (use class
	 * constants).
	 */
	public ParameterBean(String name, String type,
	                     String value, int scope)
	{
		this.name = name;
		this.type = type;
		this.value = value;
		this.scope = scope;
	}


	/**
	 * Returns the variable name used to define the paramter.
	 */
	public String getName()
	{
		return name;
	}

	/**
	 * Returns the scope of the parameter (use class constants).
	 */
	public int getScope()
	{
		return scope;
	}

	/**
	 * Returns the fully qualified class name for the parameter.
```

```
     */
    public String getType()
    {
        return type;
    }

    /**
     * Returns the string value of the parameter.
     */
    public String getValue()
    {
        return value;
    }

    /**
     * @param name, the variable name used to define the parameter.
     */
    public void setName(String name)
    {
        this.name = name;
    }

    /**
     * @param scope, the scope of the parameter (use class
     * constants).
     */
    public void setScope(int scope)
    {
        this.scope = scope;
    }

    /**
     * @param type, the fully qualified class name for the
     * paramter.
     */
    public void setType(String type)
    {
        this.type = type;
    }

    /**
     * @param value, the string value of the parameter.
     */
    public void setValue(String value)
    {
        this.value = value;
    }

} // *** END ParameterBean.java  ***********************************
```

Source Code: AssertionBean.java

```
package edu.unf.jacg0002;

/**
 * Describes an assertion performed in a TestCase
 *
 * @author Greg Jackson
 */
public class AssertionBean
{
	public static final int ASSERT_EQUALS = 1;

	public static final int SESSION = 1;
	public static final int REQUEST = 2;

	private int assertType;

	private String valueName;
	private String valueType;
	private String value;

	private String objectName;
	private String objectType;
	private int objectScope;

	public AssertionBean() {}

	/**
	 * Constructor with all required fields.
	 *
	 * @param assertType, the type of assertion
	 *                    (use class constants)
	 * @param valueName, the name of the variable used
	 *                    to store the value used in assertions.
	 * @param valueType, the fully qualified class name of
	 *                    the value used in the assertions.
	 * @param value, the string value used in the assertion.
	 * @param objectName, the name of the variable used to
	 *                     store the object returned by the
	 *                     request.
	 * @param objectType, the fully qualified class name of
	 *                     the object returned by the request.
	 * @param objectScope, the scope of the object returned
	 *                     by the request.
	 */
	public AssertionBean(int assertType, String valueName,
	                     String valueType, String value,
	                     String objectName, String objectType,
	                     int objectScope)
	{
		this.assertType = assertType;
		this.valueName = valueName;
		this.valueType = valueType;
		this.value = value;
		this.objectName = objectName;
		this.objectType = objectType;
		this.objectScope = objectScope;
	}
```

```
/**
 * Returns the the type of assertion.
 */
public int getAssertType()
{
      return assertType;
}

/**
 * Returns the name of the variable used
 * to store the object returned by the request.
 */
public String getObjectName()
{
      return objectName;
}

/**
 * Returns the the scope of the object
 * returned by the request.
 */
public int getObjectScope()
{
      return objectScope;
}

/**
 * Returns the fully qualified class
 * name of the object returned by the request.
 */
public String getObjectType()
{
      return objectType;
}

/**
 * Returns the string value used in
 * the assertion.
 */
public String getValue()
{
      return value;
}

/**
 * Returns the name of the variable used
 * to store the value used in assertions.
 */
public String getValueName()
{
      return valueName;
}

/**
 * Returns the fully qualified classname
 * of the value used in the assertions.
 */
public String getValueType()
{
      return valueType;
}
```

```
    /**
     * Sets the assertion type using class
     * constants for available assertion types.
     */
    public void setAssertType(int assertType)
    {
        this.assertType = assertType;
    }

    /**
     * Sets the name of the variable used
     * to store the object returned by the request.
     */
    public void setObjectName(String objectName)
    {
        this.objectName = objectName;
    }

    /**
     * Sets the the scope of the object
     * returned by the request.
     */
    public void setObjectScope(int objectScope)
    {
        this.objectScope = objectScope;
    }

    /**
     * Sets the fully qualified class name of
     * the object returned by the request.
     */
    public void setObjectType(String objectType)
    {
        this.objectType = objectType;
    }

    /**
     * Sets the string value used in the assertion.
     */
    public void setValue(String value)
    {
        this.value = value;
    }

    /**
     * Sets the name of the variable used to
     * store the value used in assertions.
     */
    public void setValueName(String valueName)
    {
        this.valueName = valueName;
    }

    /**
     * Sets the fully qualified class name of
     * the value used in the assertions.
     */
    public void setValueType(String valueType)
    {
        this.valueType = valueType;
    }

} // *** END AssertionBean.java  ************************************
```

Source Code: FileFactory.java

```
package edu.unf.jacg0002;

import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.StringTokenizer;

/**
 * Perform basic file operations.
 *
 * @author Greg Jackson
 */
public class FileFactory
{
	private static String outputPath;
	private static String packageName;
	private static String packageOutputPath;
	private static String separator;

	/**
	 * Initializes the FileFactory.
	 * Verifies outputPath is a directory.
	 * Creates the sub directories for the package.
	 */
	public static void init(String outputPath,
				String packageName)
				throws Exception
	{
		setOutputPath(outputPath);
		setPackageName(packageName);
	}

	/**
	 * Creates a java.io.File for the fileName
	 * supplied.File is created in the supplied
	 * outputPath + package directory.
	 */
	public static File getFile(String fileName)
	{
		File f = new File(packageOutputPath + fileName);
		return f;
	}

	/**
	 * Creates a java.io.FileWriter for the
	 * fileName supplied. A FileWriter is created
	 * for the File in the outputPath + package + fileName.
	 * If the file does not exist, it is created.
	 * If the file exist, it is opened with
	 * append set to false
	 */
	public static FileWriter getFileWriter(String fileName)
				throws IOException
	{
		FileWriter fw = new FileWriter(getFile(fileName));

		return fw;
	}
```

```
    /**
     * Creates a java.io.FileReader for the
     * fileName supplied.
     */
    public static FileReader getFileReader(String fileName)
                                throws IOException
    {
          FileReader fr = new FileReader(getFile(fileName));
          return fr;
    }

    private static void setOutputPath(String path) throws Exception
    {
          File f = new File(path);
          if(!f.isDirectory())
             throw new Exception("Output Path is not a Directory");
          outputPath = path;
          separator = f.separator;
    }

    private static void setPackageName(String name)
                         throws Exception
    {
          String pop = outputPath.trim();

          if( pop.charAt(pop.length()-1) != separator.charAt(0))
                pop = pop + separator;

          if(name == null || name.length()==0)
          {
                packageName = name;
                packageOutputPath = pop;
                return;
          }

          StringTokenizer tok = new StringTokenizer(name,".");
          while(tok.hasMoreTokens())
          {
                pop = pop + tok.nextToken() + separator;
          }

          packageName = name;
          packageOutputPath = pop;
                      File f = new File(pop);
          if(!f.exists())
          {
                if(!f.mkdirs())
                {
                     throw new Exception("Unable to create
                                          package directory");
                }
          }
    }

    /**
     * Returns the package name;
     */
    public static String getPackageName()
    {
          return packageName;
    }
} // *** END FileFactory.java  ************************************
```

Source Code: BeanToXML.java

```
package edu.unf.jacg0002;

import java.io.FileReader;
import java.io.FileWriter;
import java.io.Writer;

import org.apache.commons.betwixt.io.BeanReader;
import org.apache.commons.betwixt.io.BeanWriter;

/**
 * Converts TestSuiteBean to XML using the
 * org.apache.commons.betwixt.io.BeanReader and
 * org.apache.commons.betwixt.io.BeanWirter.
 *
 * @author Greg Jackson
 */
public class BeanToXML
{
  public static final String TEST_SUITE_FILE_NAME = "test-suite.xml";

      /**
       * Reads the test-suite.xml file and
       * converts to a TestSuiteBean class
       */
      public TestSuiteBean readTestSuiteXML(String fileName)
                          throws Exception
      {
            FileReader fileReader =
                FileFactory.getFileReader(fileName);
            BeanReader beanReader = new BeanReader();
        beanReader.getXMLIntrospector()
                  .setAttributesForPrimitives(false);
        beanReader.setMatchIDs(false);

        // Register beans so that betwixt knows what the
        //XML is to be converted to
        beanReader.registerBeanClass(TestSuiteBean.class);

        // Now we parse the xml
        TestSuiteBean testSuite = (TestSuiteBean)
                    beanReader.parse(fileReader);

            fileReader.close();

            return testSuite;
      }

      /**
       * Writes the TestSuiteBean to the
       * test-suite.xml file.
       */
      public void writeTestSuite(TestSuiteBean testSuiteBean)
                  throws Exception
      {
            FileWriter fileWriter =
                FileFactory.getFileWriter(TEST_SUITE_FILE_NAME);
            writeProlog(fileWriter);
            writeTestSuite(testSuiteBean, fileWriter);
            fileWriter.flush();
```

```
            fileWriter.close();
        }

        private void writeTestCase(TestCaseBean testCaseBean,
                                   Writer writer) throws Exception
        {
            BeanWriter beanWriter = new BeanWriter(writer);
            beanWriter.getXMLIntrospector()
                      .setAttributesForPrimitives(false);
            beanWriter.setWriteIDs(false);
            beanWriter.enablePrettyPrint();
            beanWriter.write(testCaseBean);
        }

        private void writeTestSuite(TestSuiteBean testSuiteBean,
                                    Writer writer) throws Exception
        {
            BeanWriter beanWriter = new BeanWriter(writer);
            beanWriter.getXMLIntrospector()
                      .setAttributesForPrimitives(false);
            beanWriter.setWriteIDs(false);
            beanWriter.enablePrettyPrint();
            beanWriter.write(testSuiteBean);
        }


        private void writeProlog(Writer writer) throws Exception
        {
            writer.write("<?xml version='1.0' ?>");
        }

} // *** END BeanToXML.java  ************************************
```

Source Code: TestSuiteWriter.java

```
package edu.unf.jacg0002;

import java.io.FileWriter;
import java.util.Iterator;

/**
 * Creates a TestSuite class named AllTestbased
 * on the information in the TestCaseBean.
 *
 * @author Greg Jackson
 *
 */
public class TestSuiteWriter
{
      private FileWriter fileWriter;
      public static final String TEST_SUITE_NAME = "AllTests.java";

      /**
       * Creates a class containing a TestSuite used to run
       * all the TestCases contained the the TestSuiteBean.
       */
      public void writeTestSuite(TestSuiteBean testSuiteBean)
                  throws Exception
      {
            fileWriter = FileFactory.getFileWriter(TEST_SUITE_NAME);
            fileWriter.write("package " +
                              FileFactory.getPackageName() +
                              ";" + "\n\n");
            fileWriter.write("import junit.framework.Test;" +
                              "\n\n");
            fileWriter.write("import junit.framework.TestSuite;" +
                              "\n\n");

            fileWriter.write("/**" + "\n");
            fileWriter.write(" * Used to run all TestCases generated"
                            + "\n");
            fileWriter.write(" */" + "\n");

            fileWriter.write("public class AllTests" + "\n");
            fileWriter.write("{" + "\n");

            fileWriter.write("\t" +
                              "public static void main(String[] args)"
                            + "\n");
            fileWriter.write("\t" + "{" + "\n");
            fileWriter.write("\t" +
                              "junit.swingui.TestRunner
                                          .run(AllTests.class);" +
                              "\n");
            fileWriter.write("\t" + "}" + "\n");

            fileWriter.write("\n");

            fileWriter.write("\t" +
                              "public static TestSuite suite()" +
                              "\n");
            fileWriter.write("\t" + "{" + "\n");
            fileWriter.write("\t" +
                              "TestSuite suite = new TestSuite();" +
```

```
                                "\n");

            for(Iterator it =
                testSuiteBean.getTestCaseBeans().iterator();
                it.hasNext(); )
            {
                  TestCaseBean testCaseBean =
                      (TestCaseBean) it.next();
                  fileWriter.write("\t" +
                                   "suite.addTest(new TestSuite(" +
                                   testCaseBean.getClassName() +
                                   ".class));" + "\n");
            }

            fileWriter.write("\t" + "     return suite;" + "\n");
            fileWriter.write("\t" + "}" + "\n");

            fileWriter.write("}" + "\n");

            fileWriter.flush();
            fileWriter.close();
      }

} // *** END TestSuiteWriter.java  **********************************
```

Source Code: TestCaseWriter.java

```
package edu.unf.jacg0002;

import java.io.FileWriter;
import java.util.Iterator;

/**
 * Creates a CactusStrutsTestCase class based
 * on the information in the TestCaseBean.
 *
 * @author Greg Jackson
 */
public class TestCaseWriter
{
      private FileWriter fileWriter;

      /**
       * Generates the Test Case class in the output directory.
       * @return the name of the class generated
       */
      public void writeTestCase(TestCaseBean testCaseBean)
                  throws Exception
      {
            String fileName = testCaseBean.getClassName() + ".java";
            fileWriter = FileFactory.getFileWriter(fileName);
            writePackageName(testCaseBean.getPackageName());
            writeImportStatements();
            writeClassName(testCaseBean.getClassName());
            writeCommonMethods(testCaseBean);
            writeTestMethods(testCaseBean);
            fileWriter.flush();
            fileWriter.close();
      }

      private String buildClassName(String requestPathInfo,
                                    String forward)
      {
            String className = "Test";
            if(requestPathInfo != null && requestPathInfo.length()>1)
            {
                  className = className +
                        requestPathInfo.substring(1,2).toUpperCase();
                  if(requestPathInfo.length()>2)
                        className = className +
                                    requestPathInfo.substring(2,
                                    requestPathInfo.length());
            }
            if(forward != null && forward.length()>0)
            {
                  className = className +
                              forward.substring(0,1).toUpperCase();
                  if(forward.length()>1)
                        className = className +
                              forward.substring(1,forward.length());
            }
            return className;
      }

      private String buildMethodName(String className)
      {
```

```
        String methodName = "test";
        if(className.length()>4)
              methodName = methodName + className.substring(4);
        return methodName;
}

private void writePackageName(String packageName)
              throws Exception
{
        fileWriter.write("package " + packageName + ";" + "\n");
}

private void writeImportStatements() throws Exception
{
        fileWriter.write("\n");

        fileWriter.write("import" +
                    "servletunit.struts.CactusStrutsTestCase;" +
                    "\n");
}

private void writeClassName(String className) throws Exception
{
        fileWriter.write("\n");

        fileWriter.write("public class " +
                         className +
                         " extends CactusStrutsTestCase" + "\n");
        fileWriter.write("{" + "\n");
}

private void writeCommonMethods(TestCaseBean testCaseBean)
              throws Exception
{
        fileWriter.write("\n");

        fileWriter.write("\t" + "/**" + "\n");
        fileWriter.write("\t" + " * Constructor" + "\n");
        fileWriter.write("\t" + " * @param testName" + "\n");
        fileWriter.write("\t" + " */" + "\n");
        fileWriter.write("\t" + "public " +
                         testCaseBean.getClassName() +
                         "(String testName)" + "\n");
        fileWriter.write("\t" + "{" + "\n");
        fileWriter.write("\t" + "     super(testName);" + "\n");
        fileWriter.write("\t" + "}" + "\n");

        fileWriter.write("\n");

        fileWriter.write("\t" + "/**" + "\n");
        fileWriter.write("\t" + " * Runs the test using the " +
                         "junit.swingui.TestRunner" + "\n");
        fileWriter.write("\t" + " */" + "\n");
        fileWriter.write("\t" +
                         "public static void main(String[] args)"
                       + "\n");
        fileWriter.write("\t" + "{" + "\n");
        fileWriter.write("\t" +
                         "      junit.swingui.TestRunner.run(" +
                         testCaseBean.getClassName() +
                         ".class);" + "\n");
        fileWriter.write("\t" + "}" + "\n");
```

```
        fileWriter.write("\n");

        fileWriter.write("\t" + "/**" + "\n");
        fileWriter.write("\t" +
                         " * If you override setUp()," +
                         "you must expicitly call super.setUp()"
                       + "\n");
        fileWriter.write("\t" + " */" + "\n");
        fileWriter.write("\t" +
                         "public void setUp() throws Exception" +
                         "\n");
        fileWriter.write("\t" + "{" + "\n");
        fileWriter.write("\t" + "     super.setUp();" + "\n");
        fileWriter.write("\t" + "}" + "\n");

        fileWriter.write("\n");

        fileWriter.write("\t" + "/**" + "\n");
        fileWriter.write("\t" +
                         " * If you override tearDown(), you must
                          expicitly call super.tearDown()" +
                         "\n");
        fileWriter.write("\t" + " */" + "\n");
        fileWriter.write("\t" + "public void tearDown()
                         throws Exception" + "\n");
        fileWriter.write("\t" + "{" + "\n");
        fileWriter.write("\t" + "     super.tearDown();" + "\n");
        fileWriter.write("\t" + "}" + "\n");

}

private void writeTestMethods(TestCaseBean testCaseBean)
              throws Exception
{
      for(Iterator it =
             testCaseBean.getMethodBeans()
                         .iterator(); it.hasNext(); )
      {
            MethodBean methodBean = (MethodBean) it.next();

            fileWriter.write("\n");

            fileWriter.write("\t" + "/**" + "\n");
            fileWriter.write("\t" +
                             " * Test the Action path \""+
                             testCaseBean.getRequestPath()+
                             "\" for the forward \""+
                             methodBean.getForwardName()+
                             "\"." + "\n");
            fileWriter.write("\t" +
                             " * Forward path: "+
                             methodBean.getForwardPath()+
                             "\n");
            fileWriter.write("\t" +
                             " * ActionForm: "+
                             testCaseBean.getActionFormType()+
                             "\n");
            fileWriter.write("\t" + " */" + "\n");
            fileWriter.write("\t" + "public void " +
                             methodBean.getMethodName() +
                             "()" + "\n");
            fileWriter.write("\t" + "{" + "\n");
```

```
                fileWriter.write("\t" +
                                 " setRequestPathInfo(\"" +
                                 testCaseBean.getRequestPath() +
                                 "\");" + "\n");
                Iterator paramIt =
                      methodBean.getParameterBeans().iterator();
                while(paramIt.hasNext())
                {
                      ParameterBean parameterBean =
                           (ParameterBean) paramIt.next();
                      fileWriter.write("\t\t" +
                                       "addRequestParameter(\"" +
                                       parameterBean.getName() +
                                       "\", \"" +
                                       parameterBean.getValue() +
                                       "\");" + "\n");
                }
                fileWriter.write("\t" +
                                 "      actionPerform();" + "\n");
                fileWriter.write("\t" +
                                 "      verifyForward(\"" +
                                 methodBean.getForwardName() +
                                 "\");" + "\n");
                fileWriter.write("\t" +
                                 "      verifyNoActionErrors();" +
                                 "\n");

                Iterator assertIt =
                     methodBean.getAssertionBeans().iterator();
                while(assertIt.hasNext())
                {
                      AssertionBean assertionBean =
                            (AssertionBean) it.next();
                      writeAssertStatement(assertionBean);
                }
                fileWriter.write("\t" + "}" + "\n");
          }
          fileWriter.write("\n");
          fileWriter.write("}");
    }

    private void writeAssertStatement(AssertionBean assertionBean)
                 throws Exception
    {
          String statement = "";
          String getObject = "";

          if(assertionBean.getObjectScope()==AssertionBean.SESSION)
          {
                getObject = assertionBean.getObjectType() +
                            " " +
                            assertionBean.getObjectName() +
                            " = " +
                            "request.getSession().getAttribute(\""
                          + assertionBean.getObjectName() + "\");";
          }
          else if (assertionBean.getObjectScope()
                   ==AssertionBean.REQUEST)
          {
                getObject = assertionBean.getObjectType() + " " +
                            assertionBean.getObjectName() + " = " +
                            "request.getAttribute(\"" +
                            assertionBean.getObjectName() + "\");";
```

```
            }
            else
            {
                  //Throw Exception
            }


      if(assertionBean.getAssertType()==AssertionBean.ASSERT_EQUALS)
            {
                  statement = "assertEquals(\"" +
                              assertionBean.getValue() + "\", " +
                              assertionBean.getObjectName() +
                              ");";
            }
            else
            {
                  //Throw Exception
            }

            fileWriter.write("\t\t" + getObject);
            fileWriter.write("\t\t" + statement);
      }

} // *** END TestCaseWriter.java  **********************************
```

APPENDIX B

Installation Instructions

The StrutsTestSuiteGenterator is a stand-alone Java application. It requires the JRE version 1.3.1 or higher. To run the applications simply place the StrutsTestSuiteGenterator.jar file in a directory on the computer with the JRE. Open a command prompt and change to the directory containing the JAR file. Type the following statement: to produce a usage explanation about the parameters.

```
java StrutsTestSuiteGenerator C:\MyProject\WEB-INF\struts-config.xml
C:\MyProject\src com.myproject.test
```

The first argument is the location and name of the Struts configuration file. The second is the output directory used for the generated files. The third is the package name for the generated test cases.

APPENDIX C

Sample Input File: struts-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 1.0//EN"
"http://jakarta.apache.org/struts/dtds/struts-config_1_0.dtd">
<struts-config>
      <data-sources>
      </data-sources>
      <form-beans>
            <form-bean name="directorySearchForm"
            type="com.jacg0002.tds.action_forms.DirectorySearchForm">
            </form-bean>
      </form-beans>
      <global-forwards>
            <forward name="error" path="/error.jsp"></forward>
      </global-forwards>
      <action-mappings>
            <action
            name="directorySearchForm"
            path="/directorySearch"
            scope="request"
            type="com.jacg0002.tds.actions.DirectorySearchAction"
            input="/phonesearch.jsp"
            validate="true">
                  <forward name="success" path="/searchresults.jsp"/>
                  <forward name="fail" path="/phonesearch.jsp"/>
            </action>
            <action
            name="directorySearchForm"
            path="/newSearch"
            scope="request"
            type="com.jacg0002.tds.actions.NewSearchAction"
            input="/searchresults.jsp"
            validate="false">
                  <forward name="success" path="/phonesearch.jsp"/>
            </action>
      </action-mappings>
      </struts-config>
```

Sample Output File: test-suite.xml

```
<?xml version='1.0' ?>
<TestSuiteBean>
  <testCaseBeans>
    <testCaseBean>
      <packageName>edu.unf.jacg0002.cactus</packageName>
      <actionType>
            com.jacg0002.tds.actions.NewSearchAction
      </actionType>
      <requestPath>/newSearch</requestPath>
      <actionFormName>directorySearchForm</actionFormName>
      <className>NewSearchTest</className>
      <actionFormType>
            com.jacg0002.tds.action_forms.DirectorySearchForm
      </actionFormType>
      <methodBeans>
        <methodBean>
          <forwardName>success</forwardName>
          <methodName>testSuccess</methodName>
          <forwardPath>/phonesearch.jsp</forwardPath>
          <assertionBeans/>
          <parameterBeans/>
        </methodBean>
      </methodBeans>
    </testCaseBean>
    <testCaseBean>
      <packageName>edu.unf.jacg0002.cactus</packageName>
      <actionType>
            com.jacg0002.tds.actions.DirectorySearchAction
      </actionType>
      <requestPath>/directorySearch</requestPath>
      <actionFormName>directorySearchForm</actionFormName>
      <className>DirectorySearchTest</className>
      <actionFormType>
            com.jacg0002.tds.action_forms.DirectorySearchForm
      </actionFormType>
      <methodBeans>
        <methodBean>
          <forwardName>fail</forwardName>
          <methodName>testFail</methodName>
          <forwardPath>/phonesearch.jsp</forwardPath>
          <assertionBeans/>
          <parameterBeans/>
        </methodBean>
        <methodBean>
          <forwardName>success</forwardName>
          <methodName>testSuccess</methodName>
          <forwardPath>/searchresults.jsp</forwardPath>
          <assertionBeans/>
```

```
            <parameterBeans/>
          </methodBean>
        </methodBeans>
      </testCaseBean>
    </testCaseBeans>
  </TestSuiteBean>
```

Sample Output File: AllTests.java

```
package edu.unf.jacg0002.cactus;

import junit.framework.Test;

import junit.framework.TestSuite;

/**
 * Used to run all TestCases generated
 */
public class AllTests
{
	public static void main(String[] args)
	{
		junit.swingui.TestRunner.run(AllTests.class);
	}

	public static TestSuite suite()
	{
		TestSuite suite = new TestSuite();
		suite.addTest(new TestSuite(NewSearchTest.class));
		suite.addTest(new TestSuite(DirectorySearchTest.class));
		return suite;
	}
}
```

Sample Output File: DirectorySearchTest.xml

```
package edu.unf.jacg0002.cactus;

import servletunit.struts.CactusStrutsTestCase;

public class DirectorySearchTest extends CactusStrutsTestCase
{

	/**
	 * Constructor
	 * @param testName
	 */
	public DirectorySearchTest(String testName)
	{
		super(testName);
	}

	/**
	 * Runs the test using the junit.swingui.TestRunner
	 */
	public static void main(String[] args)
	{
		junit.swingui.TestRunner.run(DirectorySearchTest.class);
	}

	/**
	 * If you override setUp(),
	 * you must expicitly call super.setUp()
	 */
	public void setUp() throws Exception
	{
		super.setUp();
	}

	/**
	 * If you override tearDown(),
	 * you must expicitly call super.tearDown()
	 */
	public void tearDown() throws Exception
	{
		super.tearDown();
	}

	/**
	 * Test the Action path "/directorySearch" for the
	 * forward "fail".
	 * Forward path: /phonesearch.jsp
	 * ActionForm:
	 * com.jacg0002.tds.action_forms.DirectorySearchForm
	 */
	public void testFail()
	{
		setRequestPathInfo("/directorySearch");
		actionPerform();
		verifyForward("fail");
		verifyNoActionErrors();
	}

	/**
	 * Test the Action path "/directorySearch" for the
```

```
         * forward "success".
         * Forward path: /searchresults.jsp
         * ActionForm:
         * com.jacg0002.tds.action_forms.DirectorySearchForm
         */
        public void testSuccess()
        {
              setRequestPathInfo("/directorySearch");
              actionPerform();
              verifyForward("success");
              verifyNoActionErrors();
        }

}
```

Sample Output File: NewSearchTest.xml

```
package edu.unf.jacg0002.cactus;

import servletunit.struts.CactusStrutsTestCase;

public class NewSearchTest extends CactusStrutsTestCase
{

	/**
	 * Constructor
	 * @param testName
	 */
	public NewSearchTest(String testName)
	{
		super(testName);
	}

	/**
	 * Runs the test using the junit.swingui.TestRunner
	 */
	public static void main(String[] args)
	{
		junit.swingui.TestRunner.run(NewSearchTest.class);
	}

	/**
	 * If you override setUp(),
	 * you must expicitly call super.setUp()
	 */
	public void setUp() throws Exception
	{
		super.setUp();
	}

	/**
	 * If you override tearDown(),
	 * you must expicitly call super.tearDown()
	 */
	public void tearDown() throws Exception
	{
		super.tearDown();
	}

	/**
	 * Test the Action path "/newSearch" for the forward "success".
	 * Forward path: /phonesearch.jsp
	 * ActionForm:
	 * com.jacg0002.tds.action_forms.DirectorySearchForm
	 */
	public void testSuccess()
	{
		setRequestPathInfo("/newSearch");
		actionPerform();
		verifyForward("success");
		verifyNoActionErrors();
	}

}
```

VITA

Gregory M. Jackson graduated from Florida State University with his Bachelors of Science in Biology. He began working with Northrop Grumman Corporation as an Industrial Engineer, but quickly transitioned into Software Engineering supporting the business and manufacturing systems. Greg will graduate from the University of North Florida in April 2004 with a Masters Degree in Computer and Information Science. His focus has been on Software Engineering under the direction of his project advisor, Dr. Arturo Sánchez of the University of North Florida.