WmUnit Guide

Version 2.0

Content

1. About WmUnit.

WmUnit is a unit testing framework for testing WebMethods Integration Server services. WmUnit test is simply JUnit test with additional methods (WmTestCase class is inherited from TestCase). It can be run as normal JUnit test from your favorite IDE or junit runner see junit.org for details. To use it you need client.jar in your classpath. That can be found in WebMethods distribution ({WmPath}IntegrationServer\lib). Simple examples can be found in test folder.

WmUnit is licensed under CPL. It uses dom4j (dom4j.org) and junit (junit.org) libs.
See our homepage at wmunit.sf.net.

2. About this Guide.

This guide presents a way of creating Test Classes using WmUnit. Examples are based on a "tutorial.catalogue:getProductData" service from WmSamples package. That is available in installation of webMethods Integration Server.

In fourth paragraph we describe format of configuration file wunitconf.xml. Use this file to configure your testing environment and your preference of creating test classes. In next paragraph you can see how to generate xml input files from developer saved input records. Sixth paragraph present how to use test class generator using console or [X]Windows. As a result you get test class with test methods ready to insert your assertions.

Finally we present how to use generated class in Eclipse IDE.

3. Installation

Download binary distribution from http://sourceforge.net/projects/wmunit. Unzip it in chosen directory. Copy webmethods files:

client61.jar – from {WmPath}/common/lib

client.jar, entbase.jar, entmisc.jar, server.jar, entssl.jar – from {WmPath}/IntegrationServer/lib

to lib directory.

And that's it.

When installing from source one needs same libraries. And one needs to run "ant".

4. Description of wunitconf.xml

WmUnit is configured in wunitconf.xml file. Changing values of elements you can set possible servers and brokers, on which you want to operate.

wunitconf.xml step by step:
There are four main elements in this file:

- <servers>
- <brokers>
- <xmlInSeparatedFile>
- <outputDir>
- <devDbg>

Which are described in section below.

4.1 Servers

Element <servers> contains a list of element <server>.
Each element <server> has two attributes: obligatory attribute “alias” describes alias of server, which is used in tests as an acronym used to identify server, and optional “default”, which value “true” determines the default server. There can be only one default server.
Additionally element <server> contains elements such as: <address> - describes the server ip address and port, <user> - name of the user to connect to Integration server, <password> - used to authorize user.
For example

```
<servers>
  <server alias="server1" default="true">
      <address>127.0.0.1:5555</address>
      <user>Administrator</user>
      <password>manage</password>
  </server>
  <server alias="server2">
      <address>192.168.1.1:3456</address>
      <user>Administrator</user>
      <password>mySimplePassword</password>
  </server>
</servers>
```

4.2 Brokers

Element <brokers> is extremely similar to element <servers>.

It contains elements <broker>, which has such attributes as
- alias (the same meaning as in section server)
- default
- is (describes to which IS broker is connected - in WmUnit 2.0 not used)

Element <broker> holds elements:
- address – describes ip address and port of broker
- name – name of broker
- clientGroup

Example

```
<brokers>
<broker alias="broker1" default="true" is="server1">
```

```
        <address>127.0.0.1</address>
        <name>my_broker</name>
        <clientGroup>IntegrationServer</clientGroup>
    </broker>
    </brokers>
```

4.3 XmlInSepareatedFile

You can also find element <xmlInSeparatedFile>, it determines whether you want to put your xml in separated file or no. That's why there are only two possible values for this element: "yes" and "no". Element is not obligatory, default value if it is not found is "yes".

4.4 Output Dir

Element <outputDir> describes the catalogue, where classes and xml files will be generated.

4.5 DevDbg

Element <devDbg> indicate the way of creation devDbg files. In this file service input xml is written in webMethods format. You can use this file to load it to webMethods Developer. Fields will be filled values from devDbg.xml

Possible values:

- none - file devDbg.xml is not created (default)
- overideFile - create one file devDbg.xml, is override by every run of test
- forEachService - create file in format devDbg+packageName+serviceName+.xml, which is overrided by every run of tests from packageName+serviceName
- forEveryTest - create files devDbg+packageName+serviceName+testName.xml
- ownFileName - create devDbg file with filename set as argument makeDevDbgFile

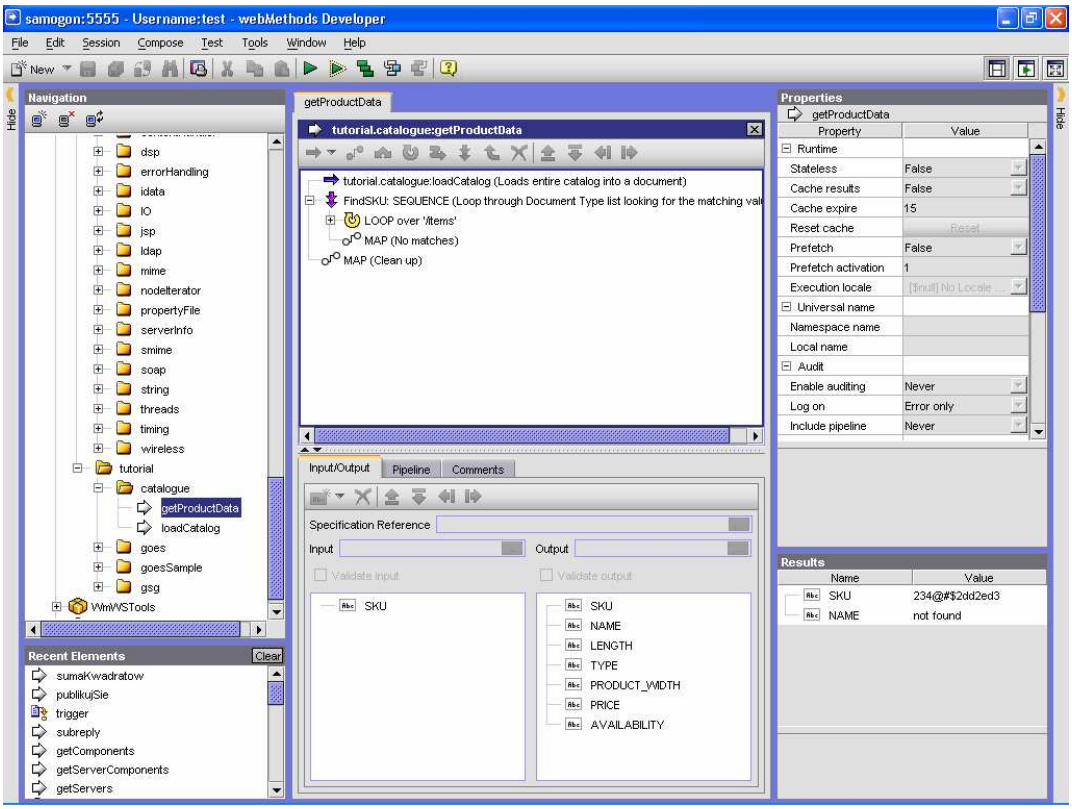
For example

<devDbg>none</ devDbg>

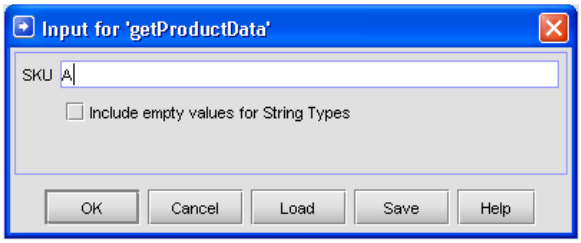
5. How to prepare XML input files.

You can invoke service using webMethods Developer with for example proper values, be sure about the results, and then use this input document to generate test case.

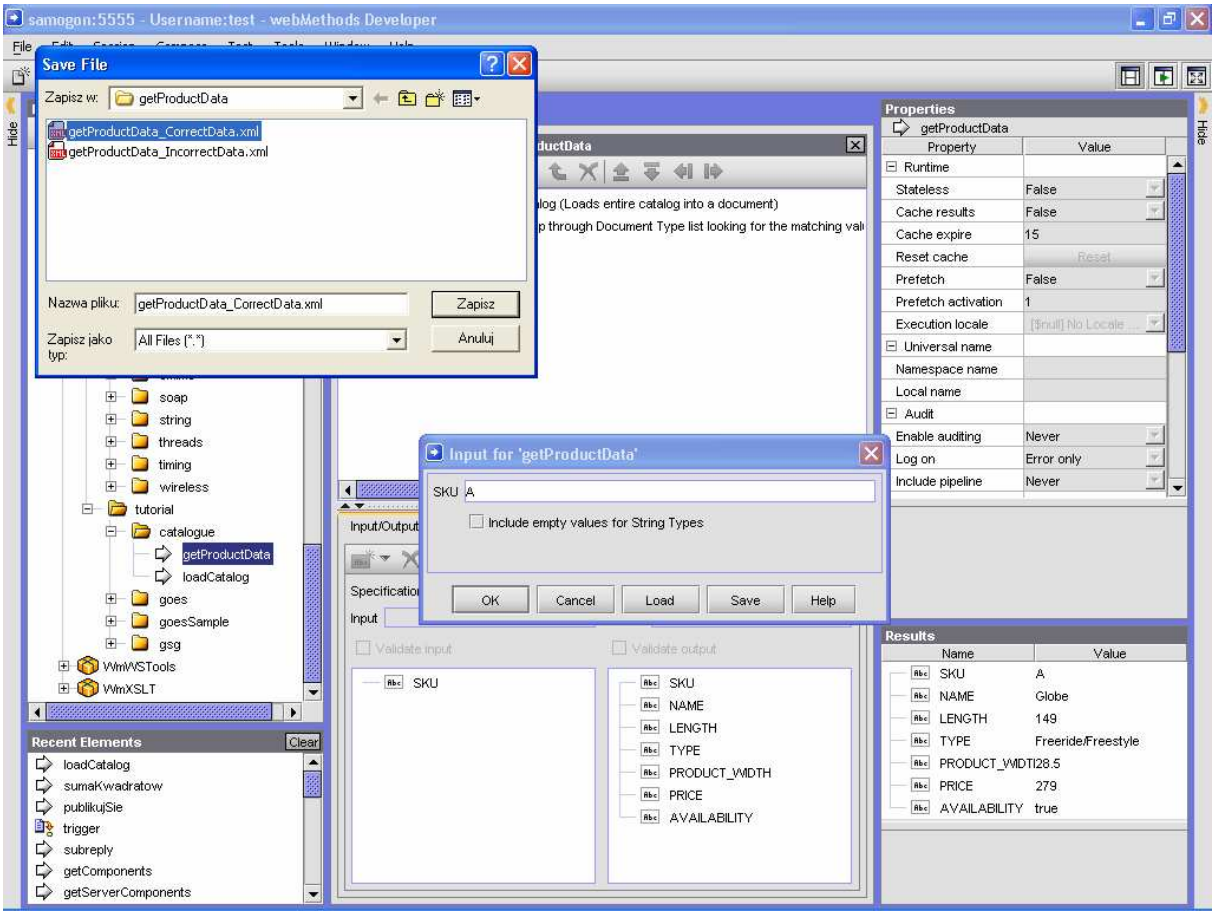
5.1. Run webMethods Developer and find service which you would like to test.



5.2. Click Test -> Run (or green Arrow) to run the service. Next fill input fields with data which you would like to use in your test case.



5.3. Click Save to save input document with filled values.



5.4. webMethods Developer will create XML file.

File example *getProductData_CorrectData.xml*:

```
<?xml version="1.0" encoding="UTF-8"?>

<IDataXMLCoder version="1.0">
 <record javaclass="com.wm.util.Values">
   <value name="SKU">A</value>
 </record>
</IDataXMLCoder>
```

5.5. You should repeat step 2.2 and 2.3 for every planned test case.

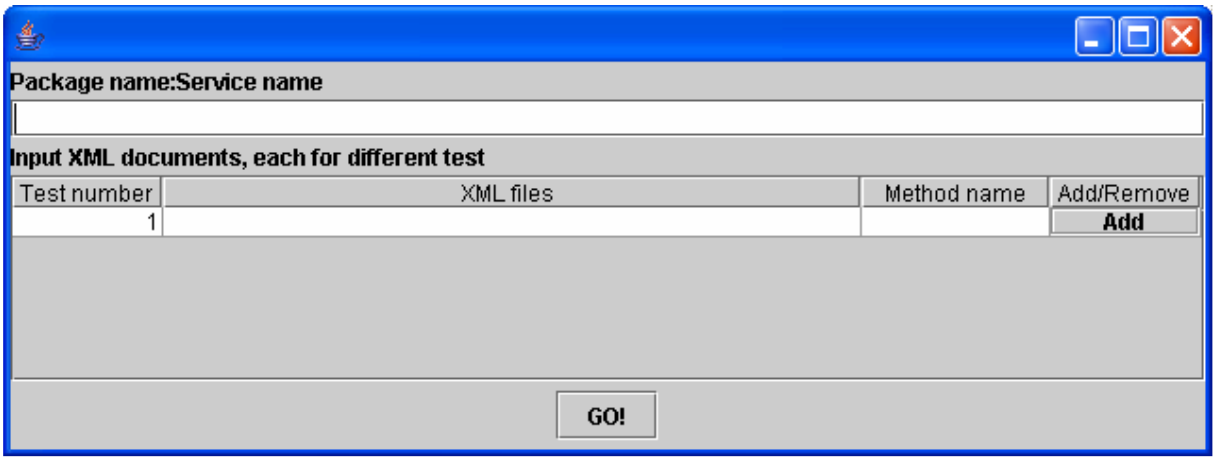
6. How to generate test class.

6.1. Console

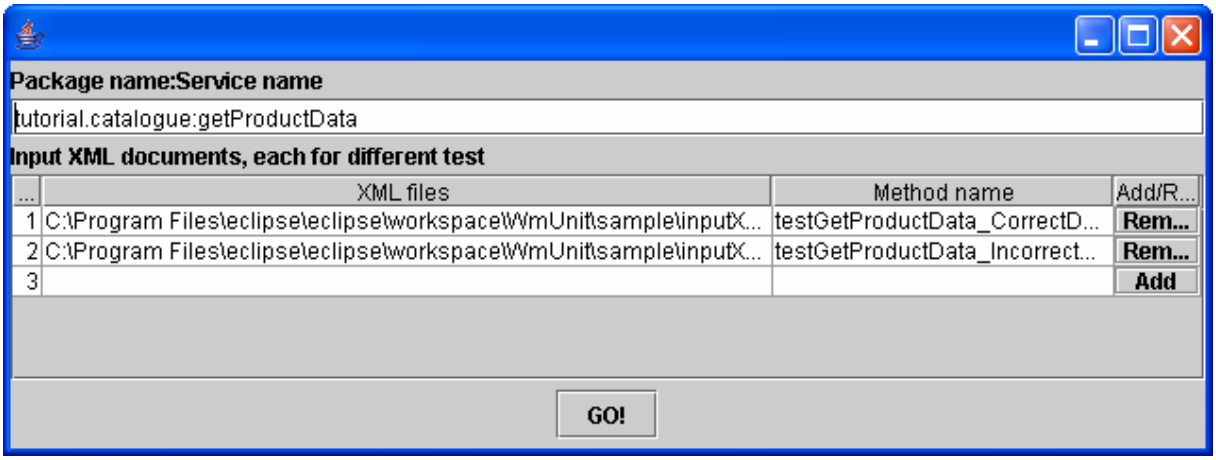
Run PrepereTest class (windows *PrepereTest.bat*) and provide arguments in proper order: <package name>, <service name>, [paths to xml files]. Arguments <package name>, <service name> are obligatory. Without [paths to xml files] PrepereTest will create only test class without test methods.

6.2. [X]windows.

6.2.1. In windows run TestCaseGenerator.bat, in Xwindows test_case_generator.sh



6.2.2. Provide name of package and service in format: <package name>:<service name>. One can use ctrl+c on package tree in WebMethods Developer to obtain service name in that form. Click Add and pick all files which you generated in step 2. Name of the test method will be automatically generated from name of input xml file adding prefix “test”. You can also write path to xml file in column XML files and then name of test method. If you don’t add prefix test it will be added automatically. It is obligatory to write both xml file path and method name. If wmunit exit it will delete this files. If you would like to keep them you should save them in different folder.



6.2.3. Click **GO!** and your test class with test cases will be created. TestCaseGenerator will create java source file name of a class is a name of a

service. Class is in a package that name is same as webmethods folder for a service. And java files are generated in source folder set in *wunitconf.xml*.

Example of java source file for service `tutorial.catalogue:getProductData`
Generated file sample/TestClassess/`tutorial/catalogue`/*GetProductData.java*:

a) If element <xmlInSeparatedFile> is set to „no”:

```
package tutorial.catalogue;

import org.dom4j.io.DocumentSource;
import org.dom4j.Document;
import org.dom4j.io.SAXReader;
import java.text.ParseException;
import java.io.File;
import com.wm.app.b2b.client.ServiceException;
import pl.infovide.wmunit.*;
import pl.infovide.wmunit.utils.*;

public class GetProductData extends WTestCase
{
	/**
	 * Test
	 * @throws Throwable
	 */
	public void testGetProductData_CorrectData()
			throws Throwable
	{
		String xml = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
				"" +
				"<WUnitIn>" +
				"  <SKU>A</SKU>" +
				"</WUnitIn>" +
				"";
		Document  inDoc = makeData(xml);
		Document outDoc = null;
		try{
			outDoc =   invoke
			("tutorial.cataloque","getProductData",inDoc,getDefaultServer())
			;

			//assertions go here

		}catch(Throwable t){
			if(inDoc!=null){
				prettyPrint(inDoc);
				makeDevDbgFile(inDoc);
			}if(outDoc!=null){
				prettyPrint(outDoc);
			}
			throw t;
		}
	}
	/**
	 * Test
	 * @throws Throwable
	 */
	public void testGetProductData_IncorrectData()
			throws Throwable
	{
		String xml = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
				"" +
				"<WUnitIn>" +
				"  <SKU>234@#$2dd2ed3</SKU>" +
				"</WUnitIn>" +
				"";
		Document  inDoc = makeData(xml);
		Document outDoc = null;
		try{
			outDoc = invoke
			("tutorial.cataloque","getProductData",inDoc,getDefaultServer())
			;

			//assertions go here

		}catch(Throwable t){
			if(inDoc!=null){
				prettyPrint(inDoc);
				makeDevDbgFile(inDoc);
			}if(outDoc!=null){
```

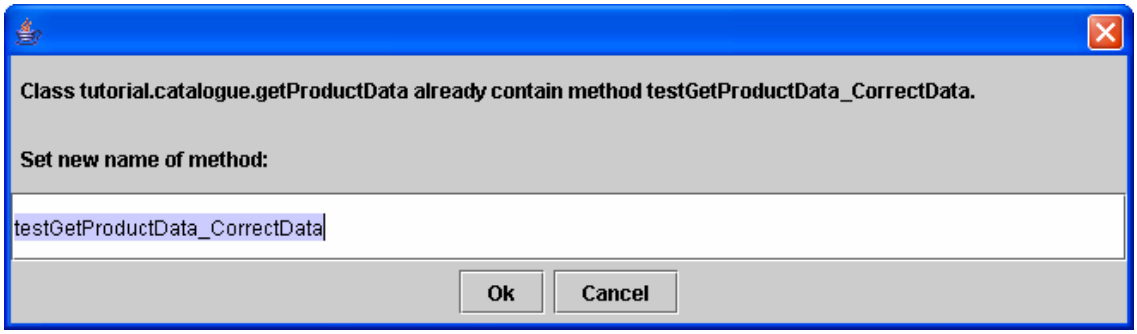
```
                        prettyPrint(outDoc);
                    }
                    throw t;
                }
        }
}
```

b) If value <xmlInSeparatedFile> is set to „yes”:

```
package tutorial.catalogue;

import org.dom4j.io.DocumentSource;
import org.dom4j.Document;
import org.dom4j.io.SAXReader;
import java.text.ParseException;
import java.io.File;
import java.io.InputStream;
import com.wm.app.b2b.client.ServiceException;
import pl.infovide.wmunit.*;
import pl.infovide.wmunit.utils.*;

public class GetProductData extends WTestCase
{
        /**
         * Test
         * @throws Throwable
         */

        public void testGetProductData_CorrectData()
                        throws Throwable
        {
                InputStream is =  ClassLoader.getSystemResourceAsStream("tutorial/
                                catalogue/getProductData_CorrectData.xml");
                Document inDoc = makeDocument(is);

                Document outDoc = null;
                try{
                        outDoc = invoke
                        ("tutorial.catalogue","getProductData",inDoc,getDefaultServer())
                        );

                        //assertions go here

                }catch(Throwable t){
                        if(inDoc!=null){
                                prettyPrint(inDoc);
                                makeDevDbgFile(inDoc);
                        }if(outDoc!=null){
                                prettyPrint(outDoc);
                        }
                        throw t;
                }

        }
}
```

Function `getSystemResourceAsStream(String arg0)` will load xml input file from folder where your test class were generated.

6.2.4. It is also possible to add test cases to existing test class. You can pick the same xml files but you should provide different name of test methods. Automatically name of test method is generated from name of input xml file but you can change it in column “Method name”. If you provide the same method name for more then one test case you will be asked to write different name of test method.



If you press **Cancel** test method will not be created and list of xml files for which test method wasn't created will be return to TestCaseGenerator. If all class and all test cases will be created successfully program will exit.

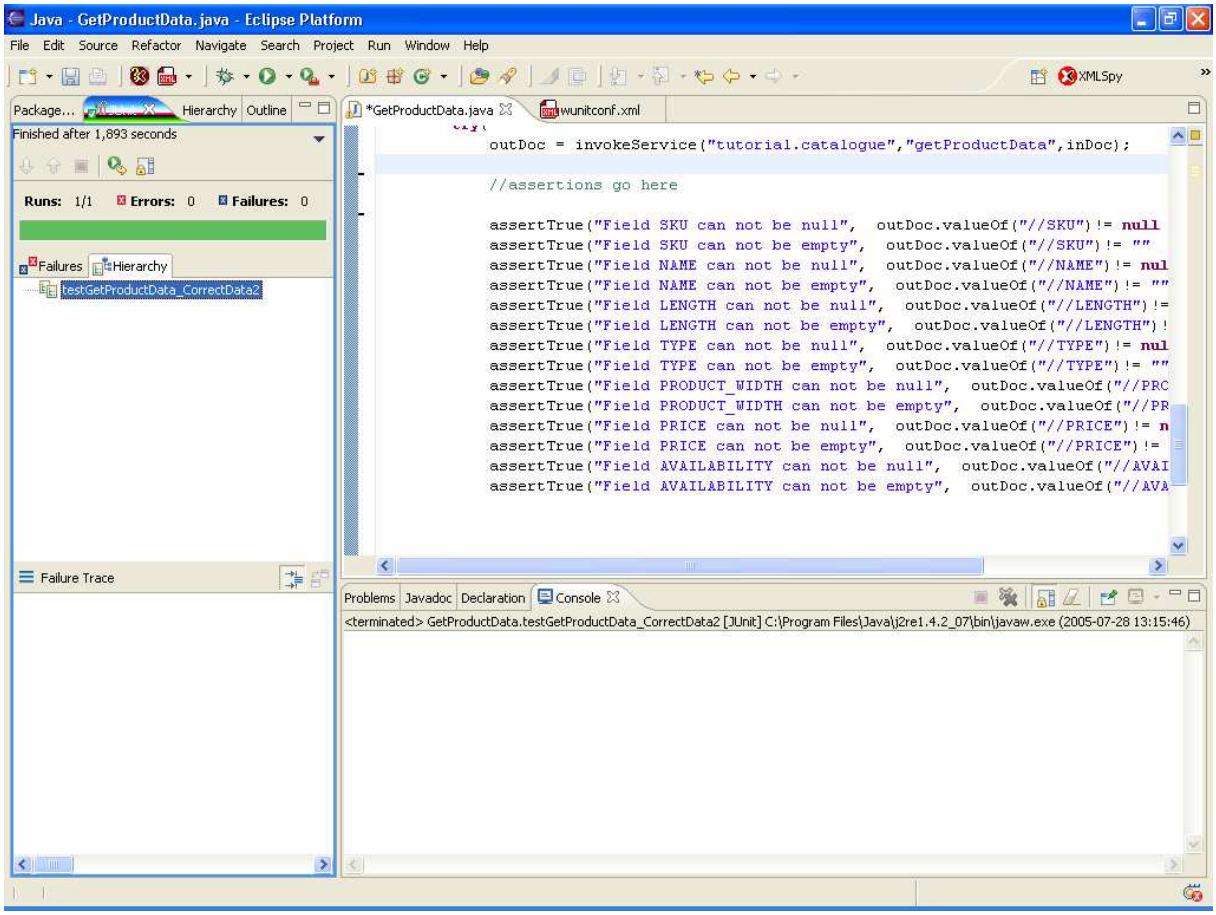
7. Sample JUnit test method using Eclipse IDE.

Load test class as Eclipse project and add all jars. You can run all test class or methods as JUnit Test.

Asserts should be putted after invoking service. Test method after putting few simple asserts which check existence of elements and if they are not empty:

```
public void testGetProductData_CorrectData2()
                        throws Throwable
        {
                String xml = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
                                "" +
                                "<WUnitIn>" +
                                "  <SKU>A</SKU>" +
                                "</WUnitIn>" +
                                "";

                Document  inDoc = makeData(xml);
                Document outDoc = null;
                try{
                        outDoc =
                        invoke("tutorial.cataloque","getProductData",inDoc,getDefaultSer
                        ver());

                                assertTrue("Field SKU can not be null",
                                        outDoc.valueOf("//SKU")!= null  );
                                assertTrue("Field SKU can not be empty",
                                        outDoc.valueOf("//SKU")!= ""  );
                                assertTrue("Field NAME can not be null",
                                        outDoc.valueOf("//NAME")!= null  );
                                assertTrue("Field NAME can not be empty",
                                        outDoc.valueOf("//NAME")!= ""  );
                                assertTrue("Field LENGTH can not be null",
                                        outDoc.valueOf("//LENGTH")!= null  );
                                 assertTrue("Field LENGTH can not be empty",
                                        outDoc.valueOf("//LENGTH")!= ""  );
                                 assertTrue("Field TYPE can not be null",
                                        outDoc.valueOf("//TYPE")!= null  );
                                assertTrue("Field TYPE can not be empty",
                                        outDoc.valueOf("//TYPE")!= ""  );
                                assertTrue("Field PRODUCT_WIDTH can not be null",
                                        outDoc.valueOf("//PRODUCT_WIDTH")!= null  );
                                assertTrue("Field PRODUCT_WIDTH can not be empty",
                                        outDoc.valueOf("//PRODUCT_WIDTH")!= ""  );
                                assertTrue("Field PRICE can not be null",
                                        outDoc.valueOf("//PRICE")!= null  );
                                assertTrue("Field PRICE can not be empty",
                                        outDoc.valueOf("//PRICE")!= ""  );
                                assertTrue("Field AVAILABILITY can not be null",
                                        outDoc.valueOf("//AVAILABILITY")!= null  );
                                assertTrue("Field AVAILABILITY can not be empty",
                                        outDoc.valueOf("//AVAILABILITY")!= ""  );



                }catch(Throwable t){
                        if(inDoc!=null){
                                prettyPrint(inDoc);
                                makeDevDbgFile(inDoc);
                        }if(outDoc!=null){
                                prettyPrint(outDoc);
                        }
                        throw t;
                }
        }
```

If every thing is ok you will get green bar, otherwise red. In good style is to put message to every assert because it is very helpful for person who use your JUnit tests to fix bugs.



8. Additional features of WmUnit 2.0.

New methods available:

a) publish(String serverAlias, String docType, Document doc) – this method publishes Document doc of type docType on broker, which is connected to server serverAlias
b) subscribe(String serverAlias, String brokerAlias, String docType) – subscribes on document of docType on broker – brokerAlias.
c) unsubscire(String docType) – we assumed that, you will subscribe on specified type of document on only one broker.
d) getDocument(String docType) – returns the last published document(during the life time of object) – you have to subscribe on this type before
e) getDocuments(String docType) – returns list of published documents of this type.
f) documentPublished(String docType) – returns true if the document of appropriate type has been published during life time of object