WmTestSuite

Working with WmTestSuite

Version 2.2.1

April 2016

DOCUMENT ID: GWM-WWST-221-20160428

CONTENTS

1 Introduction

1.1 Purpose

This document provides information on service testing options of Integration Server using Software AG WmTestSuite.

1.2 Scope

The scope of this document is to introduce the suite, provide instructions to install, design, and execute test cases.

It is assumed that the user is familiar with the standard build and test tools such as, Ant and JUnit.

2 WmTestSuite

WmTestSuite is an Eclipse based testing tool that allows the developers to create unit tests for their development. These tests can be used to improve the overall development quality and helps to provide a mechanism to create automated tools for continuous integration and delivery.

WmTestSuite provides the following functionalities:

- Provides service unit testing and regression testing tool that allows service developers to assemble unit tests without requiring additional development
- Enables integration with JUnit so that a standard unit testing framework, which already works well with Continuous Integration tools, can be leveraged
- Enables ease of use for test development
- Provides a java API for advanced users to create JUnit test cases
- Provides a user interface that is integrated in Software AG Designer. Software AG Designer ensures that users do not switch between tools for services development and corresponding test cases.
- Provides a mechanism to repeatedly execute the service with same inputs and compare the results with an expected set of outputs.
- Provides a framework for mocking service execution for steps that cannot be executed during the testing. For details, see Mocks.

2.1 Capabilities

WmTestSuite has following capabilities:

- Unit Testing
- Mock Testing
- Regression Testing

2.1.1 Unit Testing

WmTestSuite is a unit-testing tool. Unit test cases can be designed, built, executed using Eclipse User Interface. You can also execute the test cases externally using Ant scripts.

2.1.2 Mock Testing

Mocking is a feature that mimics the functionality of services that are dependent on external resources. When a test case encounters a service that is mocked, it executes the service.

2.1.3 Regression Testing

You can save the test cases, along with their inputs and outputs, in xml files. Run the reusable artifacts to ensure that the latest changes do not re-introduce the errors fixed in the earlier versions.

2.2 Environment

Option	Description
Hardware Requirements	No additional hardware is required other than the ones that are already in use for Integration Server and Software AG Designer.
Software Requirements	• WmTestSuite Eclipse plug-in v 2.2.1 • WmTestSuite library extension for Integration Server • WmServiceMock package for Integration Server • Ant Build tool 1.7 (optional) • JUnit Tool (optional) • JDK 1.7 or later
Version Compatibilities	This suite depends on open source products like Ant and JUnit. Following are the supported versions: • Ant: 1.7 • JUnit: 3.8.2 WmTestSuite works with all the currently supported GA versions of Integration Server and Software AG Designer.

2.3 What it is not

- WmTestSuite is not an integration or system test platform. However, the application dependencies can be mocked and an integration or system test can be simulated using this suite.
- WmTestSuite is not a performance-testing tool. It cannot be used for performance, load, or volume testing.

3 Working with WmTestSuite

3.1 Installing the server

Follow these steps to install the WmTestSuite:

1. Shutdown the Integration Server.
2. Backup the *config/invokemanager.cnf* file in the Integration Server install folder.
3. Unzip the required version of *serviceMockServer.zip* file to the home folder of webMethods Integration Server , for example, *C:\SoftwareAG\IntegrationServer\instances\default*.

   A jar file in the s lib/jars folder and a new package WmServiceMock is installed on the server. The WmServiceMockSamples package is installed too and can be deleted, if you do not need it. This package has some services and test cases that can be used as samples.
4. A new entry appears in the *config/invokemanager.cnf* file that refers to *com.wm.ps.serviceMock.ServiceInterceptor* as shown below:

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

<IDataXMLCoder version="1.0">
  <record javaclass="com.wm.data.ISMemDataImpl">
    <array name="processorArray" type="value" depth="1">
      <value>com.wm.ps.serviceMock.ServiceInterceptor</value>
    </array>
  </record>

</IDataXMLCoder>
```

5. Restart the server.

3.2 Software AG Designer plugin installation

Follow these steps to install the Software AG Designer plugin

1. Navigate to the Software AG Designer install location and locate the eclipse folder.

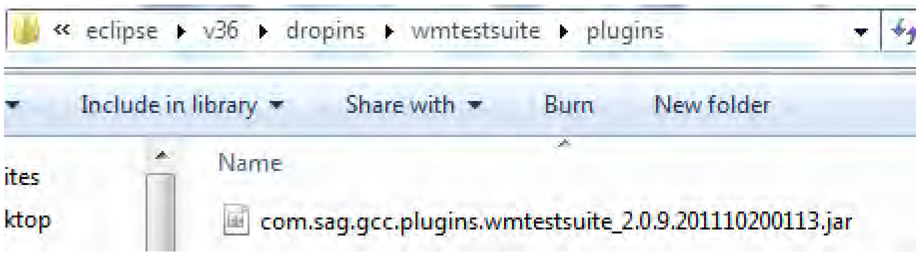
   For example,

   - If the Software AG Designer version is 9.0 and the install location is C:\SofwareAG for the webMethods development tools, then the Software AG Designer eclipse install is at *C:\SoftwareAG\eclipse\v36_90*
   - If the Software AG Designer version is 9.5, and the install location is C:\SofwareAG for the webMethods development tools, then the Software AG Designer eclipse install is at *C:\SoftwareAG\eclipse\v43_95*
   - If Software AG Designer version 9.6 or later and the install location is C:\SofwareAG for the webMethods development tools, then the Software AG Designer eclipse install is at *C:\SoftwareAG\Designer\eclipse*

   Note: Delete the earlier versions of WmTestSuite installed on Software AG Designer .
2. Create a folder named dropins at this location, if it does not already exist and then create a new folder under the dropins folder.

3. Unzip the *com.sag.gcc.plugins.wmtestsuite_xxx.zip* file under the newly created folder.

   A plugins folder is created with one jar file in it. An example installation is shown below:

   com.sag.gcc.plugins.wmtestsuite_2.0.9.201110200113.jar

4. Restart the Eclipse with ***-clean*** option after the installation. For example,

   <install_dir> /Designer/eclipse/eclipse.exe -clean.

   A new preference page called WmTestSuite is added to the Software AG Designer preferences. This preference page contains the preferences for setting up the server to develop against and some other related preferences.

   Note: If the installation fails, check the Software AG Designer logs to understand the cause of failure.

3.3 Getting Familiar

This section provides information on the terms and concepts used to understand WmTestSuite.

3.3.1 Services

The webMethods Integration Server hosts packages that contain services and related files. The server contains several packages.

Example: Packages that contain built-in services, which can be invoked from services or client applications and services that demonstrate features of the webMethods Integration Platform.

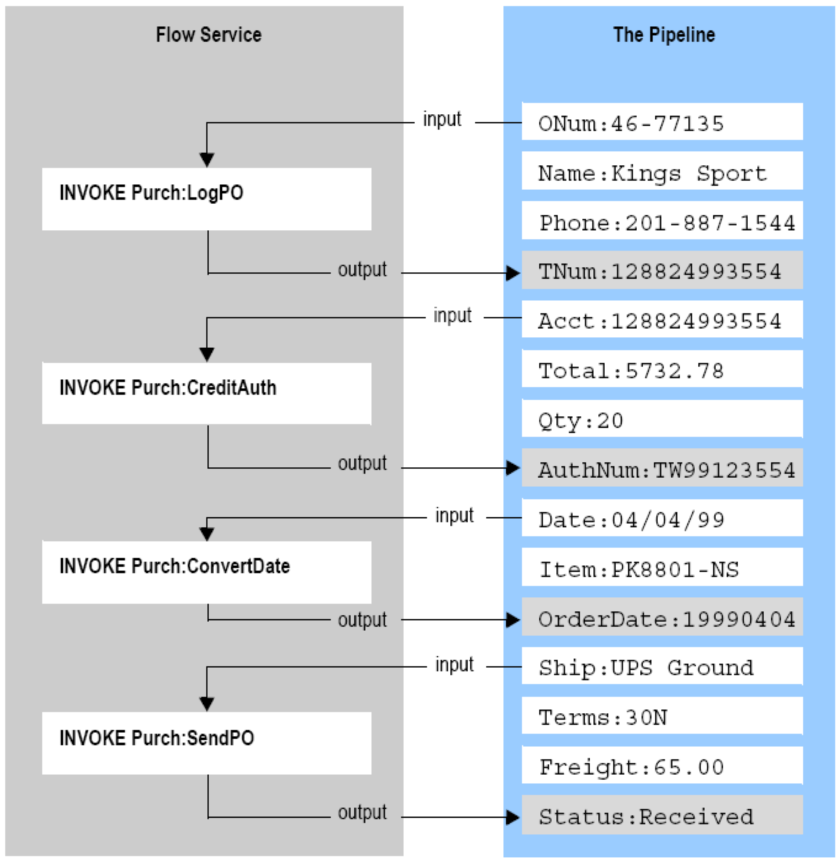
You can create additional packages to hold the services that your developers create. Developers can create services that perform functions, such as, integrating your business systems with those of your partners, retrieving data from legacy systems, and accessing and updating databases.

Integration Server provides an environment for the orderly, efficient, and secure execution of services. It decodes client requests, identifies the requested services, invokes the services, passes data to them in the expected format, encodes the output produced by the services, and returns output to the clients.

3.3.2 Pipeline

Pipeline refers to the data structure in which input and output values are maintained for a flow service. It allows services in the flow to share data.

The pipeline holds the input and output for a flow service



Pipeline starts with the input to the flow service and collects inputs and outputs from subsequent services in the flow. When a service in the flow executes, it has access to all data in the pipeline at that point.

3.3.3 Unit Testing

WmtestSuite uses the concepts of service execution, pipeline data, and the open source JUnit testing framework to provide unit testing functionality for Integration Server Flow and Java services. WmtestSuite provides the ability to create a suite of tests consisting of individual test cases. Each test case defines a service to be tested, the type of test to be performed, and provides a user interface to define input data to the test case through the pipeline. When the service execution is completed, the pipeline output is validated against the expected output defined in the test case.

3.3.4 Test Case

A test case is a unit of testing for a service that provides:

- service to be tested
- inputs to the service
- expected output from the service

A test case can also define expected output from a service as an exception or error. The service returns the defined errors when incorrect data is sent to it.

3.3.5 Test Suite

A test suite is one or more test cases grouped together. Test suites are used to organize test cases into sets of related tests. For example, a service may provide a variety of capabilities based upon the inputs provided to it. A complete test suite should include test cases that provide inputs that fully test all of the possible outputs of the service, including errors or exceptions.

3.3.6 Mock

Mocks provide a means of simulating interaction with resources that are unavailable or the data provided by these resources or systems is not consistent for test purposes. Mocks also have a lifetime that can be either limited to the test case in which they are defined or applied to all of the test cases that follow within a test suite from the point of the definition. Mocks allow for selection of scope of ***session***, ***user***, or ***server*** to control how broadly the mock intercept will be effective. It is recommended to set the scope to ***session*** for most uses and is only applicable for the test session. If you set the scope to ***user***, all the sessions for the particular user will be affected by the mock. If you set the scope to ***server***, all user sessions gets affected. Mocks can be enabled or disabled for test case or test suite execution.

3.3.7 Service Mock

A service mock is used to replace the call to a service with a call to a different service. Any call to the mocked service is intercepted and the alternate service defined in the mock is called instead. The output of the mocked service is then returned to the calling service. This kind of mock is useful when the output of the mocked service needs to be dynamic based on some logic that can be created in the service.

3.3.8 Exception Mock

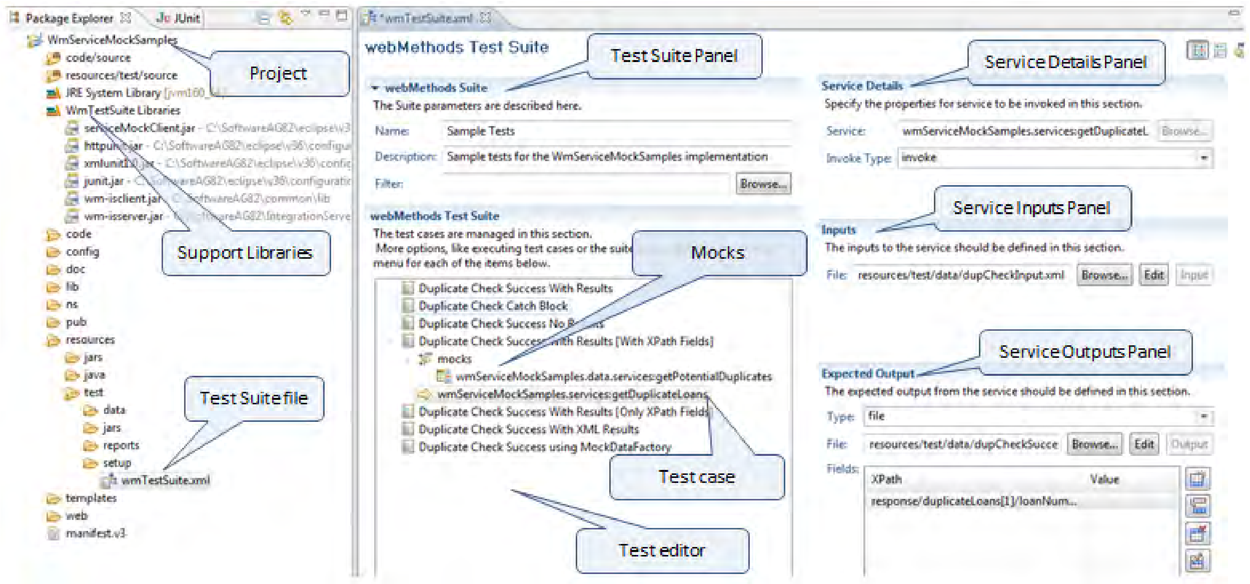
An exception mock is used to return an error or exception to the calling service and can be useful for testing error handling in a service. As with the other mocks described, any call to the service defined in the mock is intercepted and the exception defined in the mock is returned instead. This kind of mock is useful to simulate behavior that can cause exceptions in the normal flow.

3.3.9 Factory Mock

A factory mock is used when multiple different results can be produced by a call to a service based on the input provided. A factory mock is implemented as a Java class. The WmServiceMockSamples project provides a Sample Mock Factory. Any call to the class defined in a factory mock is intercepted and the input is passed instead to the factory which evaluates the input and returns the appropriate results. While both the factory mock and the service mock options provide dynamic output simulation, the factory mock does not require an extra test service to be created on the server and relies on a lightweight java implementation.

3.4 Layout

The figure below shows various components of Software AG Designer with a test suite file open for editing.



webMethods Suite
The Suite parameters are described here.
Name: Sample Tests
Description: Sample tests for the WmServiceMockSamples implementation
Filter:

webMethods Test Suite
The test cases are managed in this section.
More options, like executing test cases or the suite ... menu for each of the items below.

Duplicate Check Success With Results
Duplicate Check Catch Block
Duplicate Check Success No R...
Duplicate Check Success with Results [With XPath Fields]
mocks
wmServiceMockSamples.data.services:getPotentialDuplicates
wmServiceMockSamples.services:getDuplicateLoans
Duplicate Check Success With Results [Only XPath Fields]
Duplicate Check Success With XML Results
Duplicate Check Success using MockDataFactory

Service Details
Specify the properties for service to be invoked in this section.
Service: wmServiceMockSamples.services:getDuplicateL
Invoke Type: invoke

Inputs
The inputs to the service should be defined in this section.
File: resources/test/data/dupCheckInput.xml

Expected Output
The expected output from the service should be defined in this section.
Type: file
File: resources/test/data/dupCheckSucce
Fields:

XPath	Value
response/duplicateLoans[1]/loanNum...	

Use the , , and icons on the toolbar to tailor the layout.

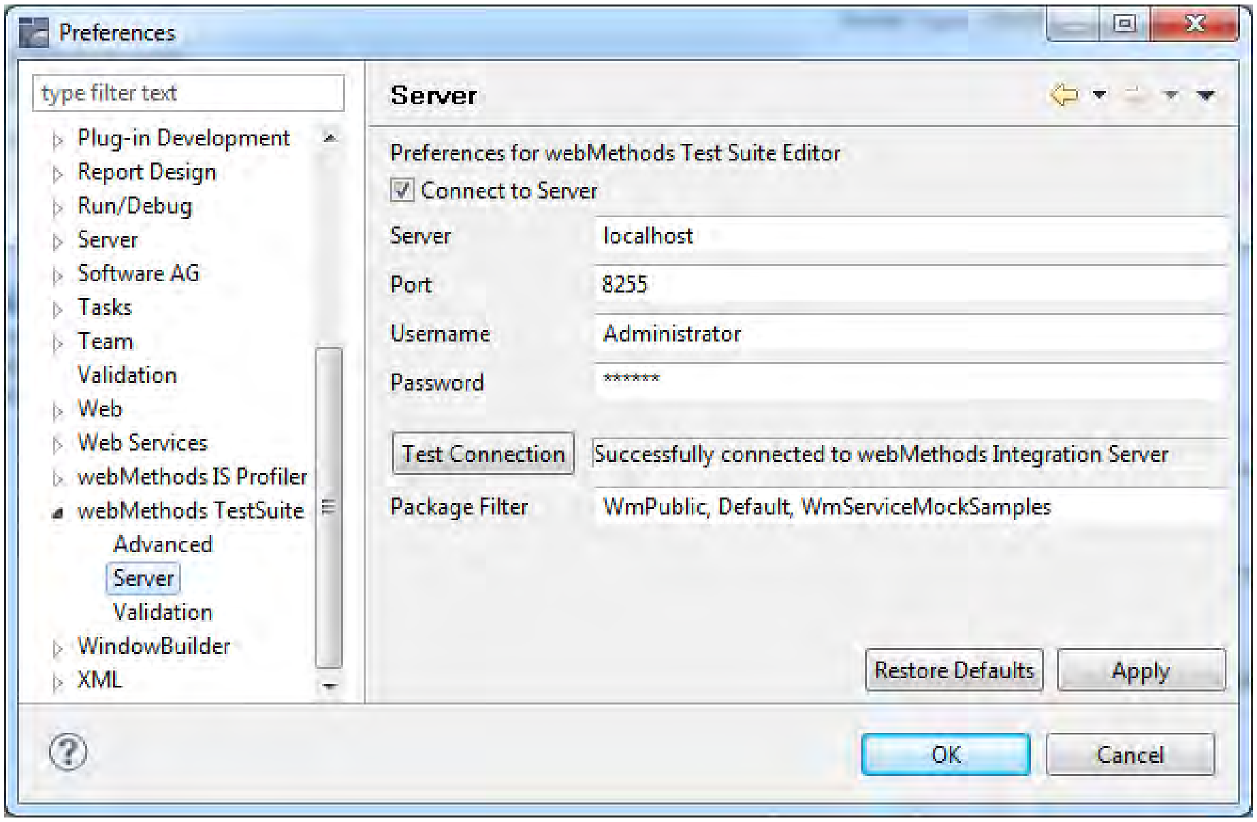
- The icon allows you to place the Master and Details views next to each other, master on the right and details on the left.
- The icon allows you make the master appear at the top of the display, and the details view gets aligned underneath it.
- The icon allows you to toggle the display between the master and details views, each occupying the entire display. Click again to return to the original layout.

3.5 WmTestSuite Preferences

To display the preferences dialog, select **Window**>**Preferences**>**webMethods Tests**.

3.5.1 Server

The server settings are shown in the figure below:



Server

Preferences for webMethods Test Suite Editor

[✓] Connect to Server

Server: localhost

Port: 8255

Username: Administrator

Password: ******

Test Connection: Successfully connected to webMethods Integration Server

Package Filter: WmPublic, Default, WmServiceMockSamples

The Server preferences describe the connection to webMethods Integration Server for WmTestSuite.

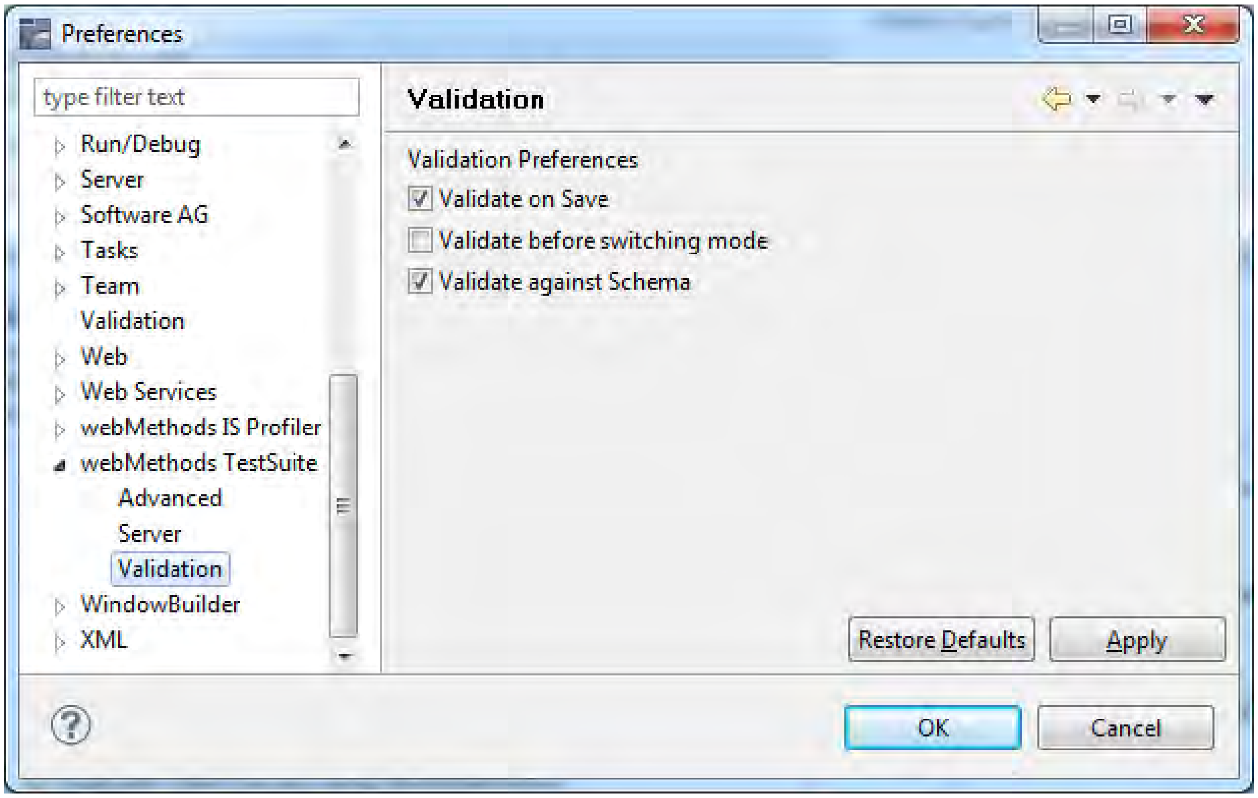
Select **Connect to Server** to allow WmTestSuite to execute tests against a specific Integration Server. Additionally, provide the Server, Port, Username, and Password and click **Test Connection** to ensure connectivity to the selected Integration Server.

Package Filter allows you to optionally enter a comma-separated list of packages for WmTestSuite to load to the Service Browser. When a large number of packages exist on the Integration Server, this feature will conserve memory in WmTestSuite by only loading those services for which test cases are developed.

The editor does not always require an active connection to the Integration Server for test development. But some of the introspection features which allow for service lookup and service signature are not available if a connection is not available and the user would have to enter these manually

3.5.2 Validation

The Validation preferences describe when and how WmTestSuite should validate information you enter.



Validation

Validation Preferences

- [✓] Validate on Save
- [ ] Validate before switching mode
- [✓] Validate against Schema

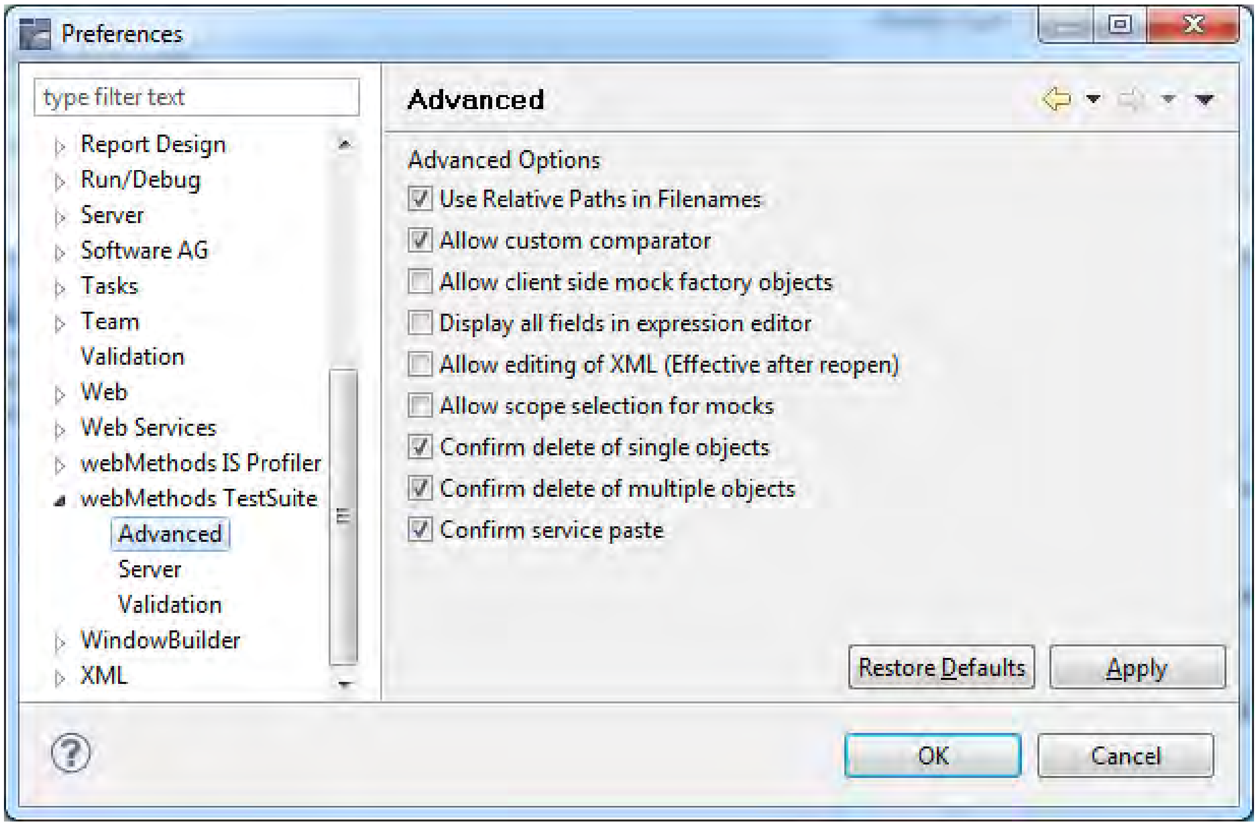
Select **Validate on Save** to allow WmTestSuite to validate the test suite prior to saving it to the file system. This option is enabled by default.

Select **Validate before switching mode** to allow WmTestSuite to perform validation of the test suite prior to switching from the XML editor mode to the test suite editor.

Select **Validate against schema** to allow WmTestSuite to validate the test suite against the XML schema. This is useful if the XML Source tab is used for entering details about one or more test cases in the suite. This option is also enabled by default.

3.5.3 Advanced

The Advanced preferences control other behaviors of WmTestSuite.



Advanced Options

- [x] Use Relative Paths in Filenames
- [x] Allow custom comparator
- [ ] Allow client side mock factory objects
- [ ] Display all fields in expression editor
- [ ] Allow editing of XML (Effective after reopen)
- [ ] Allow scope selection for mocks
- [x] Confirm delete of single objects
- [x] Confirm delete of multiple objects
- [x] Confirm service paste

Select **Use Relative Paths in Filenames** to make file name references relative to the Eclipse project containing the test suite rather than using the full path. Using relative names helps in executing tests when the test suite and data files are moved between environments and systems. This option is enabled by default.

Select **Allow custom comparator** to enable the option for the test creator to specify a custom comparator for expected output other than provided with the plug-in. The default setting is appropriate for most test case scenarios but may be needed for the advanced user. This setting only controls the display of the field that allows the selection of the comparator and does not allow/disallow this during execution.

Select **Allow client side mock factory objects** to enable the option of creating client side factory object that do not need to be deployed on the server before executing test cases. This option controls the display of the field that controls whether the mock factory objects are needed on the server or can be pushed during the execution of the tests.

Select to enable additional fields in the Expected Output section of a test case. This can be useful if several output conditions need to be checked together.

Select Allow editing of **Display all fields in the expression editor** XML (Effective after reopen) to enable editing of the XML source for the test suite. This option requires that the open test suite file is closed and reopened before its behavior reflects in the editor.

Select **Allow scope selection for mocks** to enable scope selection for the mock. The default setting of unchecked is appropriate for most test case scenarios.

There are some additional controls on the advanced preference page that controls the Copy/Paste behavior in the test suite editor.

3.6 Samples

The server installation has a package named WmServiceMockSamples which includes services and a test suite with some test cases that can be used to test these services and demonstrate the capabilities of WmTestSuite. Test developers can use this sample package for a quick start.

The WmServiceMockSamples package has the following top level services:

- wmServiceMockSamples.services:getDuplicateLoans
- wmServiceMockSamples.services:getDuplicateLoansXML

These services can be used as sample business services that need to be unit tested. The wmServiceMockSamples.services:getDuplicateLoansXML service is a wrapper over the wmServiceMockSamples.services:getDuplicateLoans service and demonstrates a service that takes XML input and returns XML output.

The WmServiceMockSamples package itself can be imported as a java project into the workspace. The sample test suite is included in the file resources/test/setup/wmTestSuite.xml. This is user interface based version of the test suite that does not require any code.

There are two test classes in the resources/test/source folder. The class DuplicateCheckTest.java demonstrates the use of JUnit java code for creating test cases. The class AllTests.java demonstrates the execution of the test cases in an XML file created by the plugin. It uses the settings in default.properties file to find the XML file and the Integration Server to test against.

There is also a test folder in the package with a set of service that can be used as test cases for testing these services. These are reference test cases. Normally, the test cases is executed from a client then they can be automated easily from JUnit. But these sample services are useful to demonstrate the services in the WmServiceMock package that enable to the tester to setup and configure mocks for services. These test services are:

wmServiceMockSamples.test:testDupCheckCatchBlock

wmServiceMockSamples.test:testDupCheckSucessNoResults

wmServiceMockSamples.test:testDupCheckSucessWithResults

wmServiceMockSamples.test:testDupCheckSucessWithXMLResults

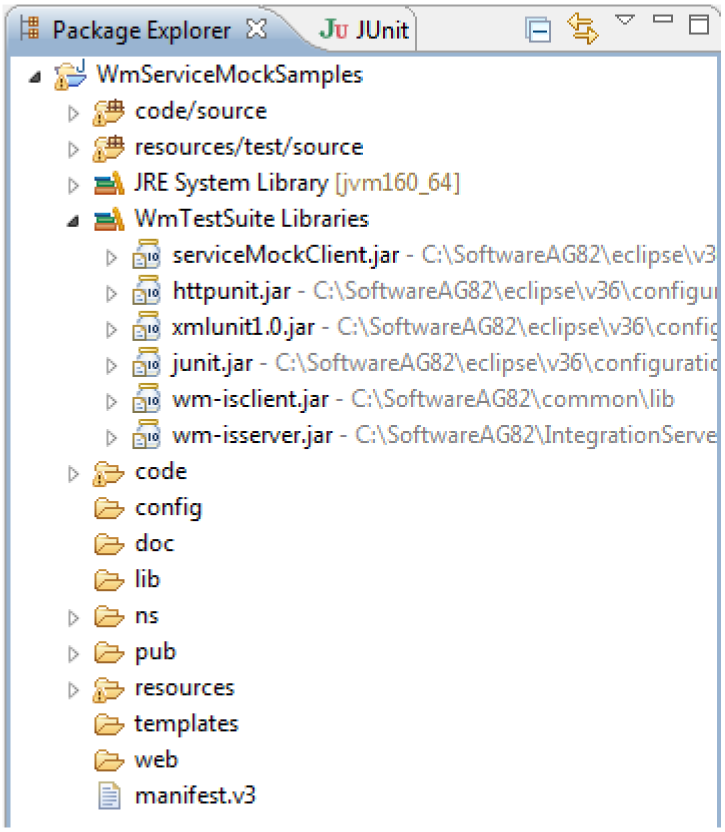
4 Creating a Test Suite

4.1 Before You Begin

Before creating a test suite, ensure that you organize the required test cases and data files in a test folder.

If the test cases are packaged in the Integration Server package, it provides a common source location for all related assets. You can import the package on eclipse workspace.

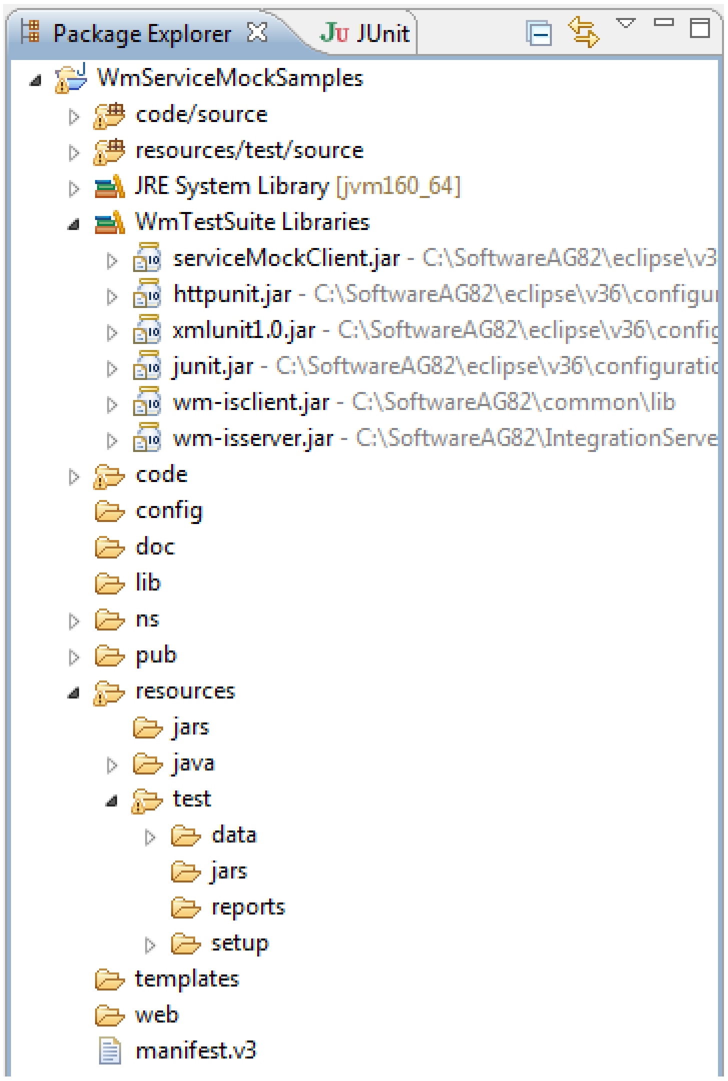
Example: Figure below shows a package hierarchy as exposed in Eclipse. Here the test cases are created in the resources test folder of the Integration Server package.



```
Package Explorer    JUnit
WmServiceMockSamples
    code/source
    resources/test/source
    JRE System Library [jvm160_64]
    WmTestSuite Libraries
        serviceMockClient.jar - C:\SoftwareAG82\eclipse\v3
        httpunit.jar - C:\SoftwareAG82\eclipse\v36\configu
        xmlunit1.0.jar - C:\SoftwareAG82\eclipse\v36\config
        junit.jar - C:\SoftwareAG82\eclipse\v36\configuratio
        wm-isclient.jar - C:\SoftwareAG82\common\lib
        wm-isserver.jar - C:\SoftwareAG82\IntegrationServe
    code
    config
    doc
    lib
    ns
    pub
    resources
    templates
    web
    manifest.v3
```

While any organizational structure that conforms to Integration Server package structure can be used, the following example provides a useful approach for organizing test cases in your environment. For ease of organization, follow the steps below:

1. Right click the **Resources** directory and create a subdirectory under it called **test.**
2. Right-click the test directory and create two additional folders **data** and **setup**.
3. Add the test suites to the **setup** directory and organize the **data** directory using subdirectories for each test suite to contain the input data files for the test cases comprising a test suite.



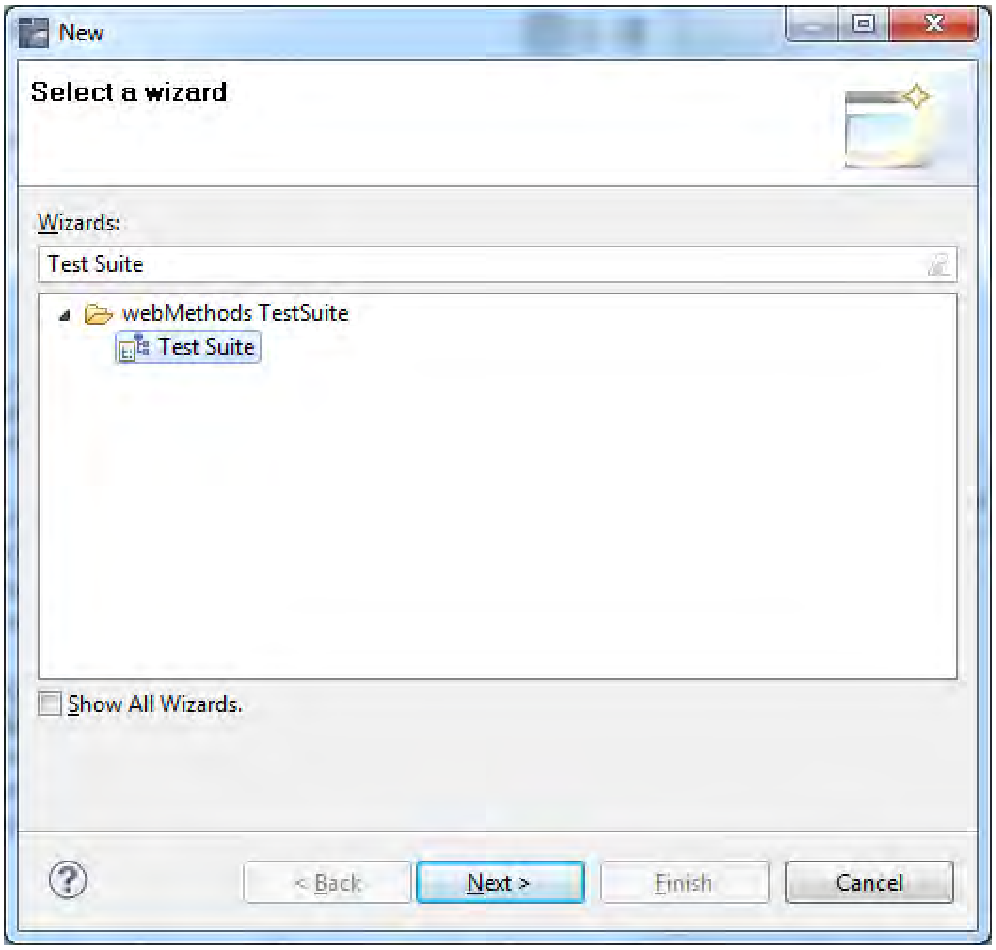
```
WmServiceMockSamples
    code/source
    resources/test/source
    JRE System Library [jvm160_64]
    WmTestSuite Libraries
        serviceMockClient.jar - C:\SoftwareAG82\eclipse\v3
        httpunit.jar - C:\SoftwareAG82\eclipse\v36\configu
        xmlunit1.0.jar - C:\SoftwareAG82\eclipse\v36\config
        junit.jar - C:\SoftwareAG82\eclipse\v36\configurati
        wm-isclient.jar - C:\SoftwareAG82\common\lib
        wm-isserver.jar - C:\SoftwareAG82\IntegrationServe
    code
    config
    doc
    lib
    ns
    pub
    resources
        jars
        java
        test
            data
            jars
            reports
            setup
    templates
    web
    manifest.v3
```

4.2 To Create a Test Suite

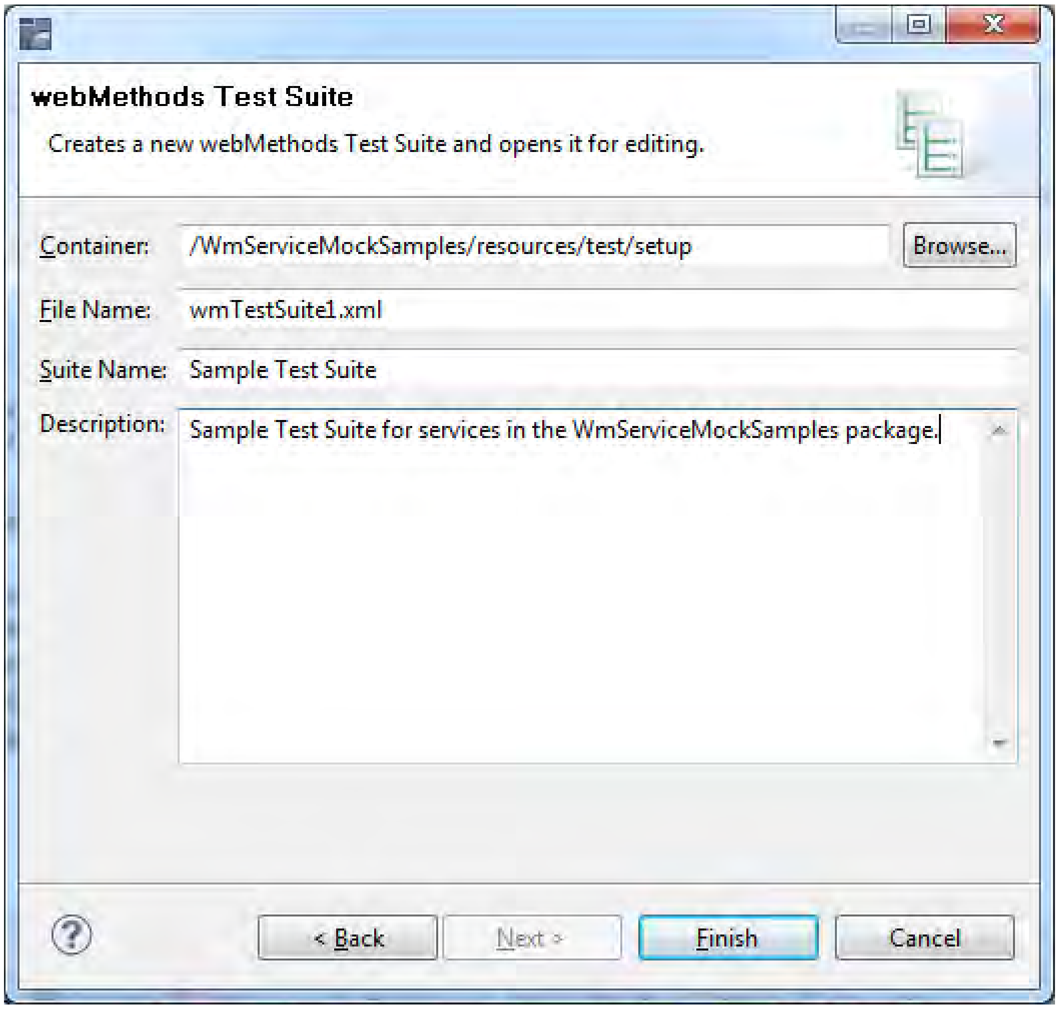
Follow the below steps to create a test suite:

1. Navigate to **resources**> **test**> **setup** folder.
2. Select **File**>**New**>**Other**.

   The **New** screen appears.



3. Select webMethods Test Suite and click **Next**.
4. Complete the information as shown in the image below.



webMethods Test Suite

Creates a new webMethods Test Suite and opens it for editing.

Container: /WmServiceMockSamples/resources/test/setup

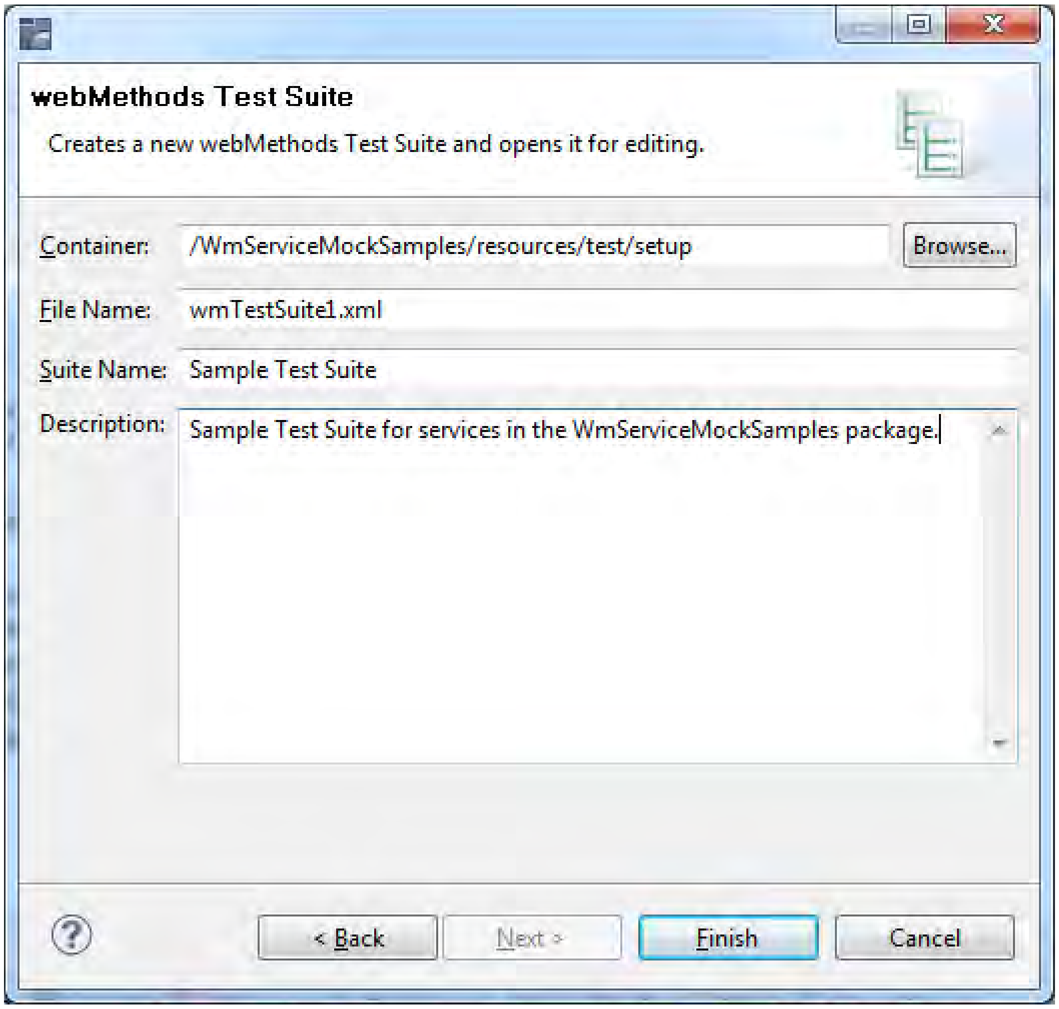
File Name: wmTestSuite1.xml

Suite Name: Sample Test Suite

Description: Sample Test Suite for services in the WmServiceMockSamples package.

5. Click **Finish**.

   You can now start creating the test cases.


webMethods Test Suite
Creates a new webMethods Test Suite and opens it for editing.
Browse...
< Back
Next >
Finish
Cancel

Container: /WmServiceMockSamples/resources/test/setup

File Name: wmTestSuite1.xml

Suite Name: Sample Test Suite

Description: Sample Test Suite for services in the WmServiceMockSamples package.

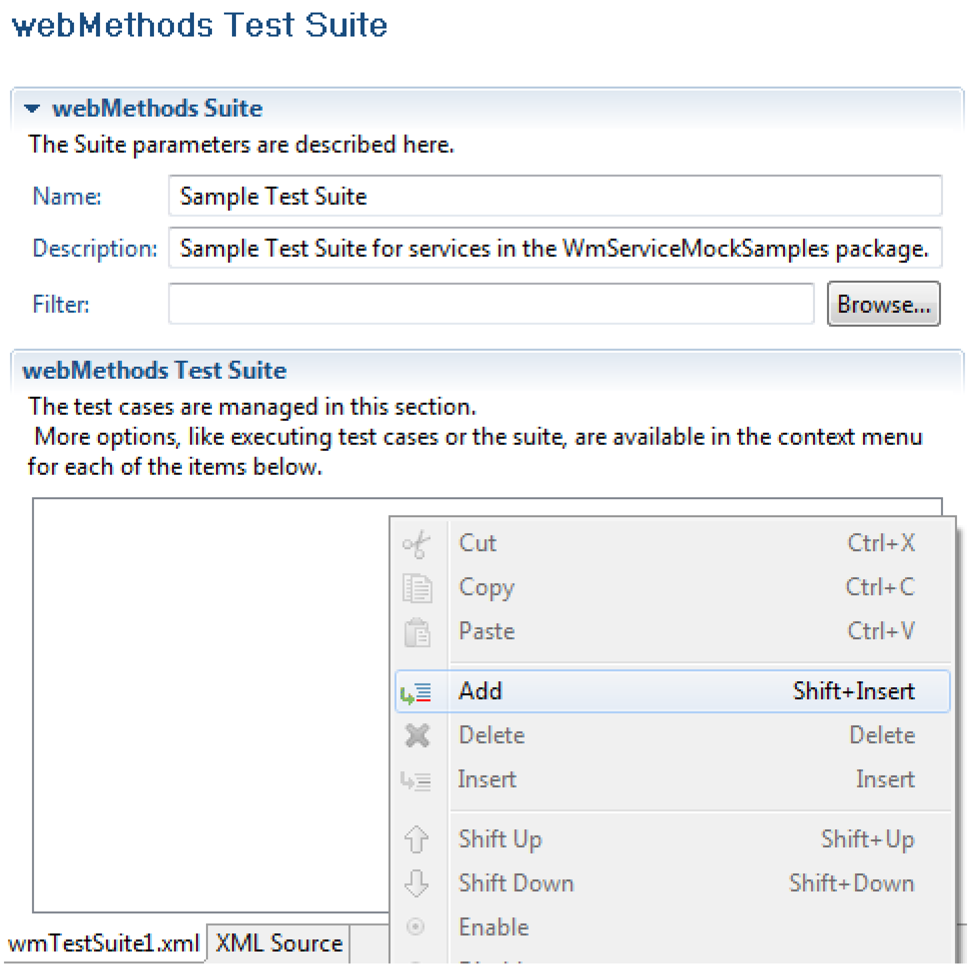
5 Adding Test Cases

Adding a test case involves the following steps:

- Adding required test details (See Test Details)
- Adding the required service details (See Service Details)
- Defining inputs to services (See Inputs)
- Defining the expected output (See Expected Output)

5.1 Test Details

To add test cases, right click in the webMethods Tests section of the display.


webMethods Test Suite

webMethods Suite

The Suite parameters are described here.

Name: Sample Test Suite

Description: Sample Test Suite for services in the WmServiceMockSamples package.

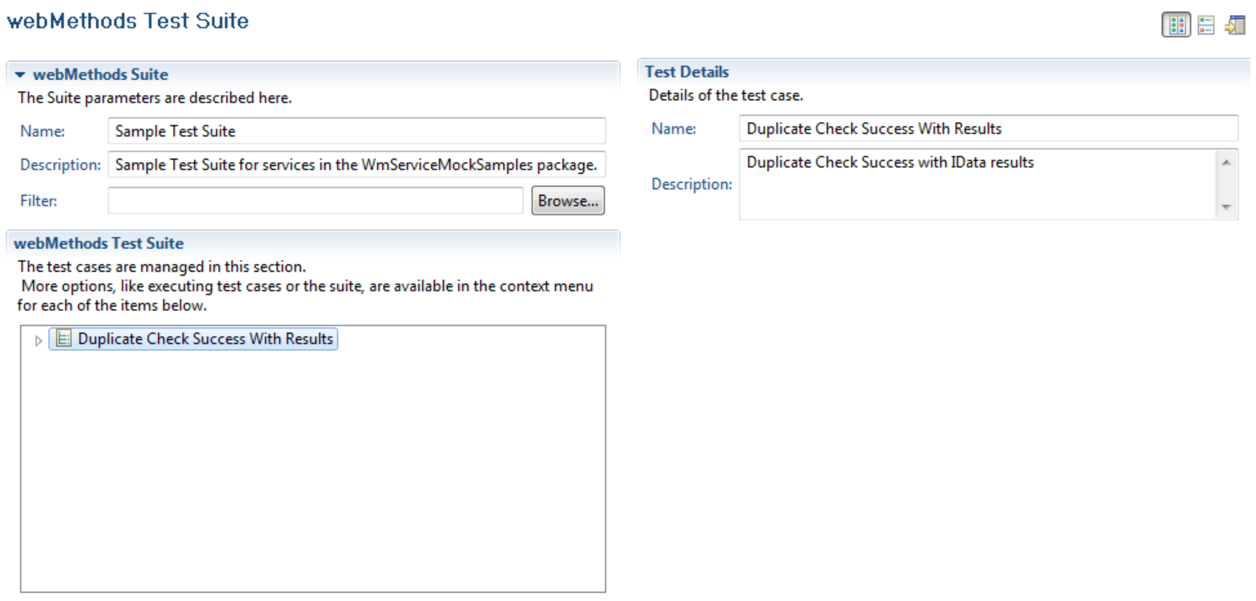
Filter:

webMethods Test Suite

The test cases are managed in this section.

More options, like executing test cases or the suite, are available in the context menu for each of the items below.

The Test Details section appears in the details area to the right side of the display. Fill in the name of the test case and a description of the test case.


webMethods Test Suite

webMethods Suite
The Suite parameters are described here.

Name:	Sample Test Suite
Description:	Sample Test Suite for services in the WmServiceMockSamples package.
Filter:	

webMethods Test Suite
The test cases are managed in this section.
More options, like executing test cases or the suite, are available in the context menu for each of the items below.

Duplicate Check Success With Results

Test Details
Details of the test case.

Name:	Duplicate Check Success With Results
Description:	Duplicate Check Success with IData results

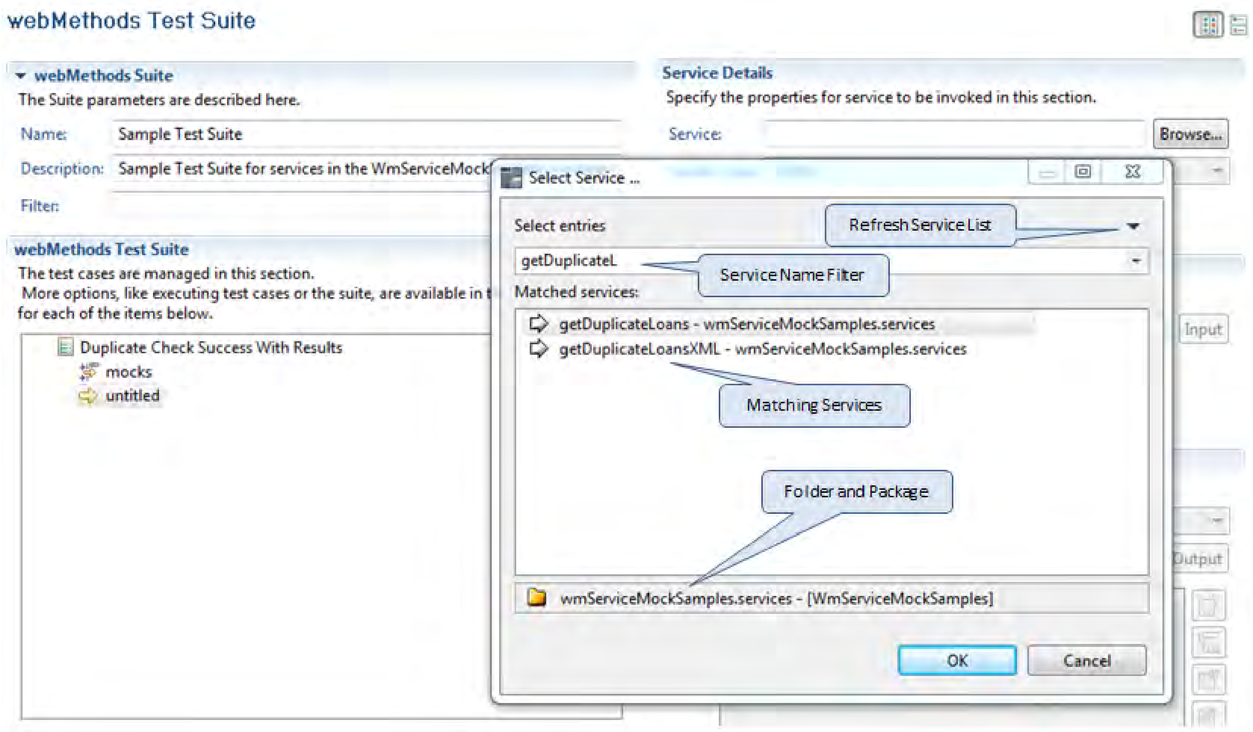
5.2 Service Details

After creating the test details, you can now select the service to be tested.

1. Click **Browse** in the Service Details section to enable searching for the service within the Integration Server (as defined in the WmTestSuite Preferences section).

   Note: You can narrow the search by entering a package name or service name and the service browser will restrict the display to only those packages or services matching the search criteria entered.

2. Click ▼ to refresh or set additional options for the service browser.


webMethods Test Suite

webMethods Suite
The Suite parameters are described here.
Name: Sample Test Suite
Description: Sample Test Suite for services in the WmServiceMock
Filter:

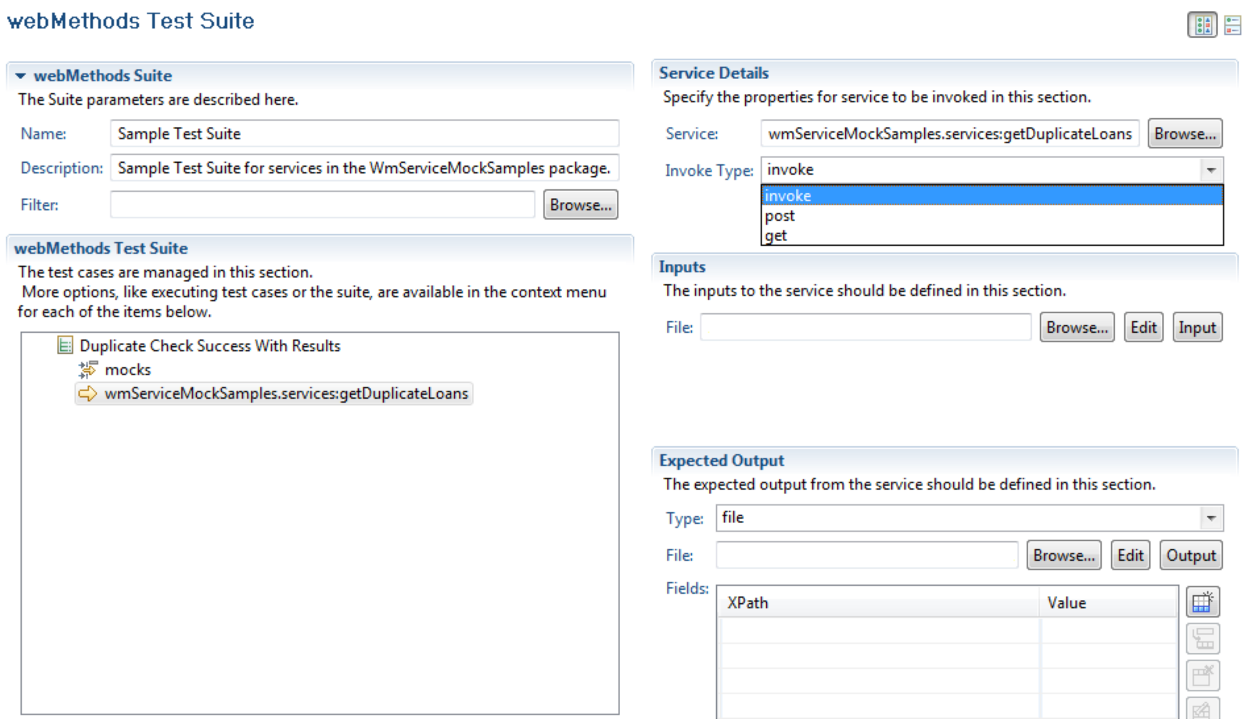
webMethods Test Suite
The test cases are managed in this section.
More options, like executing test cases or the suite, are available in t
for each of the items below.

Duplicate Check Success With Results
mocks
untitled

Service Details
Specify the properties for service to be invoked in this section.
Service:

Select Service ...
Select entries
getDuplicateL
Matched services:
getDuplicateLoans - wmServiceMockSamples.services
getDuplicateLoansXML - wmServiceMockSamples.services
wmServiceMockSamples.services - [WmServiceMockSamples]

3. Select the way in which the service will be called as shown in the image below. Options are invoke, post, and get to indicate the protocol to be used.


webMethods Test Suite

webMethods Suite
The Suite parameters are described here.
Name: Sample Test Suite
Description: Sample Test Suite for services in the WmServiceMockSamples package.
Filter:

webMethods Test Suite
The test cases are managed in this section.
More options, like executing test cases or the suite, are available in the context menu for each of the items below.

Duplicate Check Success With Results
mocks
wmServiceMockSamples.services:getDuplicateLoans

Service Details
Specify the properties for service to be invoked in this section.
Service: wmServiceMockSamples.services:getDuplicateLoans
Invoke Type: invoke

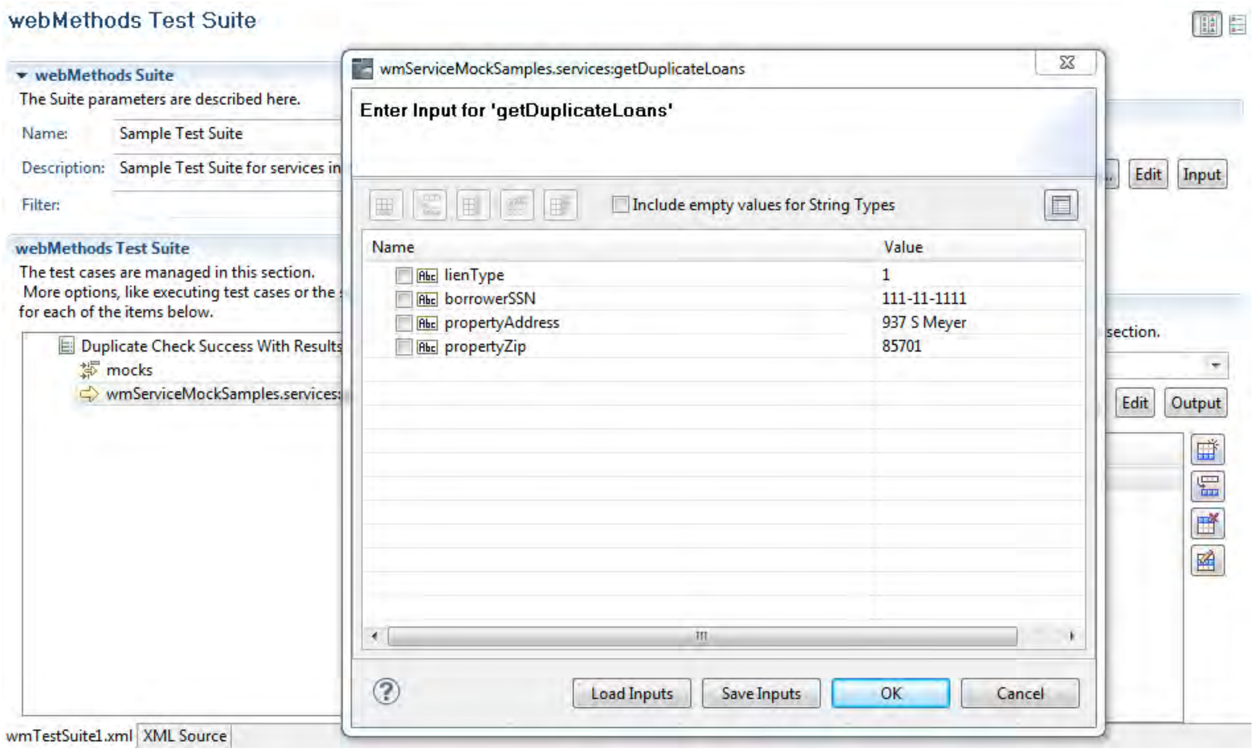
Inputs
The inputs to the service should be defined in this section.
File:

Expected Output
The expected output from the service should be defined in this section.
Type: file
File:
Fields:

XPath	Value

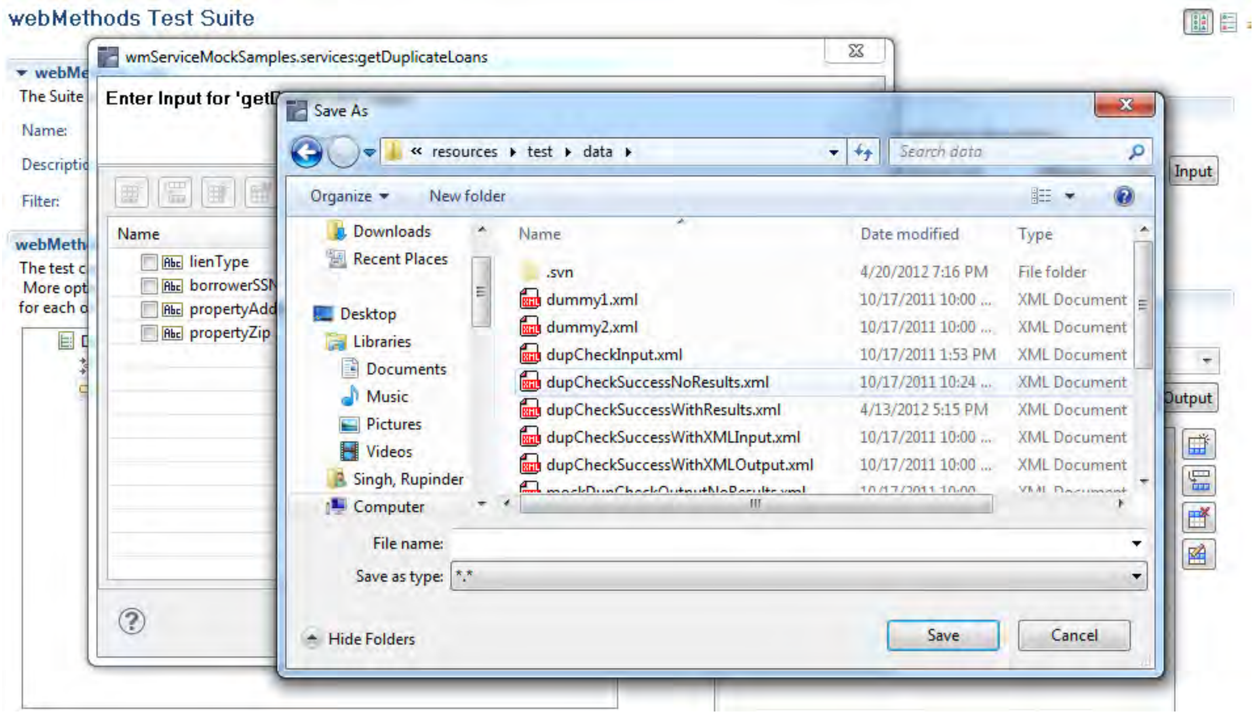
5.3 Inputs

WmTestSuite looks at the *service signature* for services in the Integration Server. When defining the inputs to the service selected in Service Details, click **Input** to display the structure of the inputs defined in the service signature. Provide values for each of the input parameters defined in the service, including complex structures such as, document and nested documents. Click **Edit** to use the same interface for modifying previously saved input data files in the same way.


webMethods Test Suite
wmServiceMockSamples.services:getDuplicateLoans
Enter Input for 'getDuplicateLoans'
Include empty values for String Types
Load Inputs
Save Inputs
OK
Cancel
Edit
Input
Output
wmTestSuite1.xml
XML Source

Name	Value
lienType	1
borrowerSSN	111-11-1111
propertyAddress	937 S Meyer
propertyZip	85701

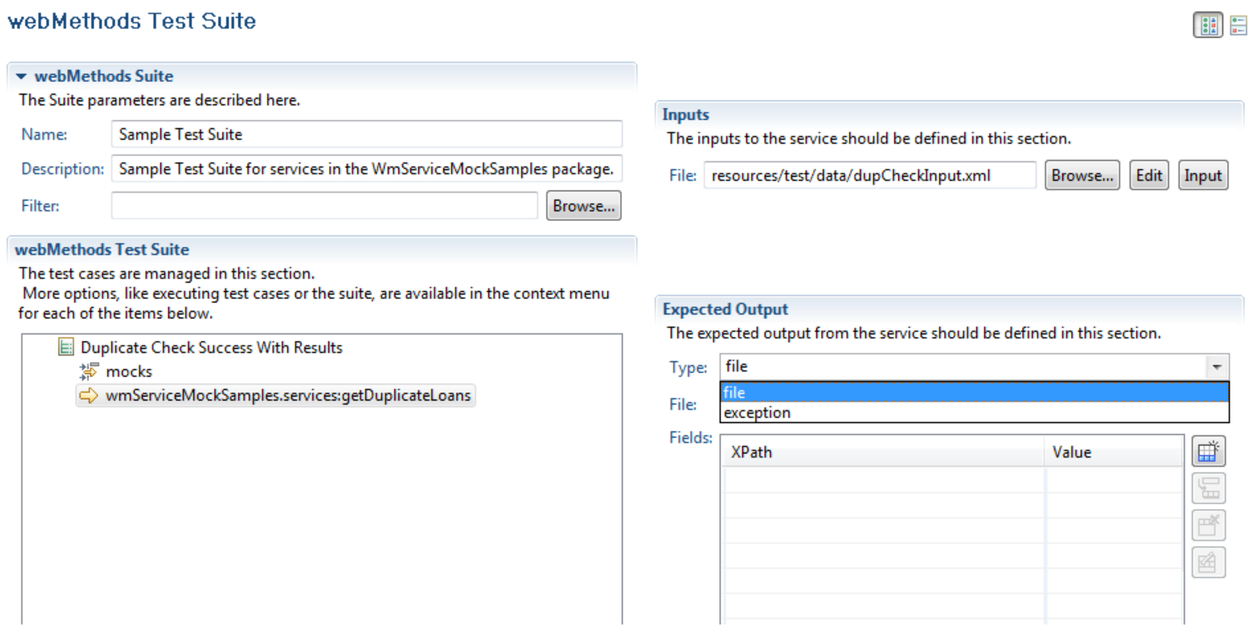
Once the inputs are defined, click **Save** to save the file in the location designated for input data to the test suite. Click **Browse** to select the saved file and add it as the input to the service.



Name	Date modified	Type
.svn	4/20/2012 7:16 PM	File folder
dummy1.xml	10/17/2011 10:00 ...	XML Document
dummy2.xml	10/17/2011 10:00 ...	XML Document
dupCheckInput.xml	10/17/2011 1:53 PM	XML Document
dupCheckSuccessNoResults.xml	10/17/2011 10:24 ...	XML Document
dupCheckSuccessWithResults.xml	4/13/2012 5:15 PM	XML Document
dupCheckSuccessWithXMLInput.xml	10/17/2011 10:00 ...	XML Document
dupCheckSuccessWithXMLOutput.xml	10/17/2011 10:00 ...	XML Document

5.4 Expected Output

The final step in defining a test case is to complete the Expected Output section. Expected Output can be in the form of data returned from the service or an exception returned from the service.



webMethods Test Suite

webMethods Suite
The Suite parameters are described here.
Name: Sample Test Suite
Description: Sample Test Suite for services in the WmServiceMockSamples package.
Filter: Browse...

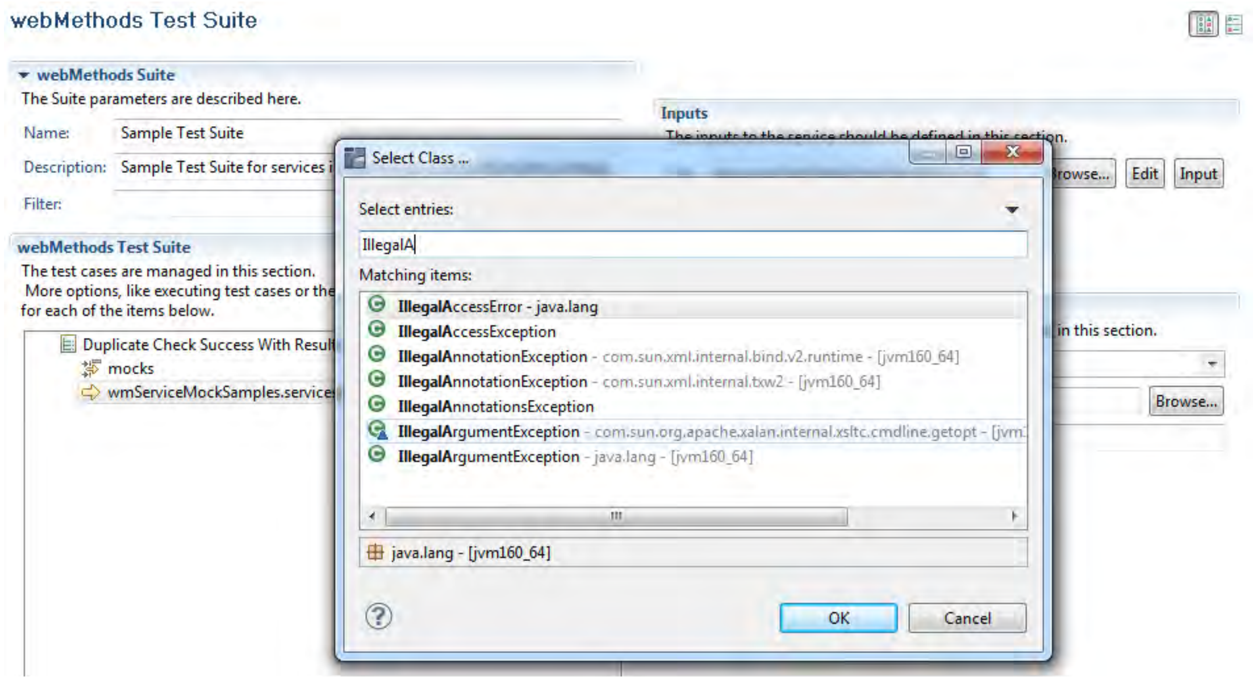
webMethods Test Suite
The test cases are managed in this section.
More options, like executing test cases or the suite, are available in the context menu for each of the items below.
Duplicate Check Success With Results
mocks
wmServiceMockSamples.services:getDuplicateLoans

Inputs
The inputs to the service should be defined in this section.
File: resources/test/data/dupCheckInput.xml Browse... Edit Input

Expected Output
The expected output from the service should be defined in this section.
Type: file
file
exception
File:
Fields:

XPath	Value

When you definee an exception or error output from the service, the class browser can be used. Click **Browse** to display the classes and select the proper exception class to use.



webMethods Test Suite

webMethods Suite
The Suite parameters are described here.
Name: Sample Test Suite
Description: Sample Test Suite for services i
Filter:

webMethods Test Suite
The test cases are managed in this section.
More options, like executing test cases or the
for each of the items below.
Duplicate Check Success With Resul
mocks
wmServiceMockSamples.service

Inputs
rowse... Edit Input
in this section.
Browse...

Select Class ...
Select entries:
IllegalA
Matching items:
IllegalAccessError - java.lang
IllegalAccessException
IllegalAnnotationException - com.sun.xml.internal.bind.v2.runtime - [jvm160_64]
IllegalAnnotationException - com.sun.xml.internal.txw2 - [jvm160_64]
IllegalAnnotationsException
IllegalArgumentException - com.sun.org.apache.xalan.internal.xsltc.cmdline.getopt - [jvm
IllegalArgumentException - java.lang - [jvm160_64]
java.lang - [jvm160_64]
OK Cancel

To define the data output from the service, use the following methods:

- Use XPath expressions to define which data elements in the output data should be evaluated.

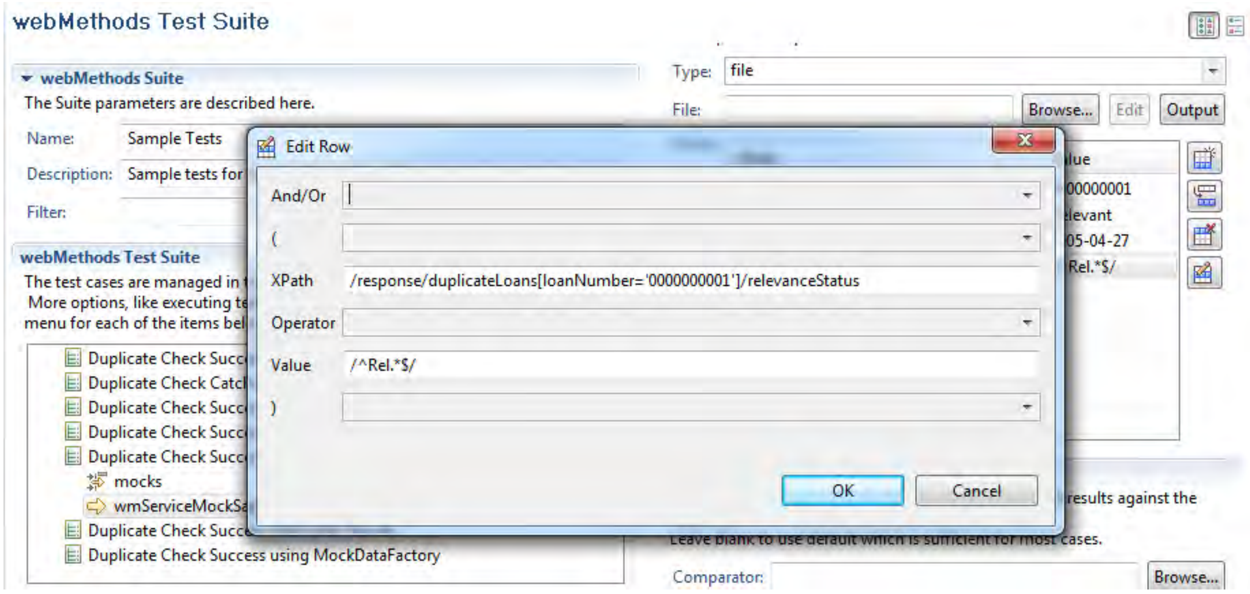
a. Copy the XPath expressions from Software AG Designer. To copy the XPath expression, right click on the variable in the **Results** area when executing the service within Software AG Designer and select **Copy**.

b. Click ▦, ▤, or ✎ and paste the copied value into the XPath field.

An entry for the variable selected is created. However, it in most cases it is not required to edit them as the webMethods paths are 0 index based whereas XPath expressions are 1 index based.

c. Enter the expected output for the field in the **Value** field.

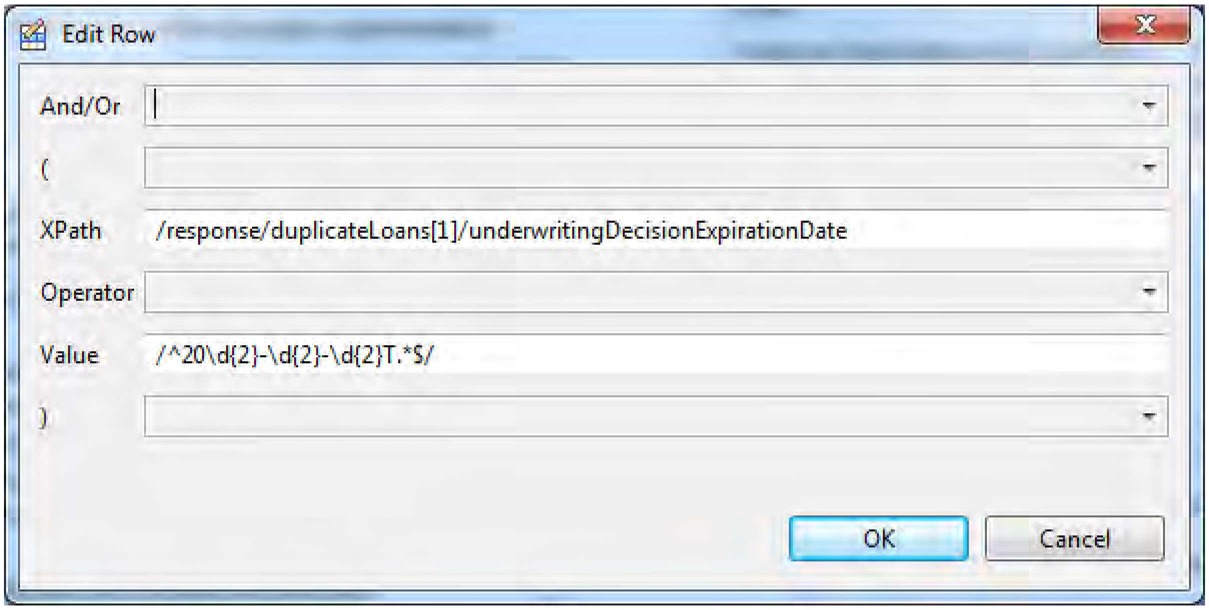
You can select AND, OR, parentheses, and operators from the drop-down lists to create complex evaluations of the output.



Edit Row

And/Or	
(	
XPath	/response/duplicateLoans[loanNumber='0000000001']/relevanceStatus
Operator	
Value	/^Rel.*$/
)	

- Use regular expressions to evaluate the output returned from the service. The regular expression should be placed in the Value field preceded and followed by "/" character.



Edit Row

And/Or	
(	
XPath	/response/duplicateLoans[1]/underwritingDecisionExpirationDate
Operator	
Value	/^20\d{2}-\d{2}-\d{2}T.*$/
)	

6 Mocks

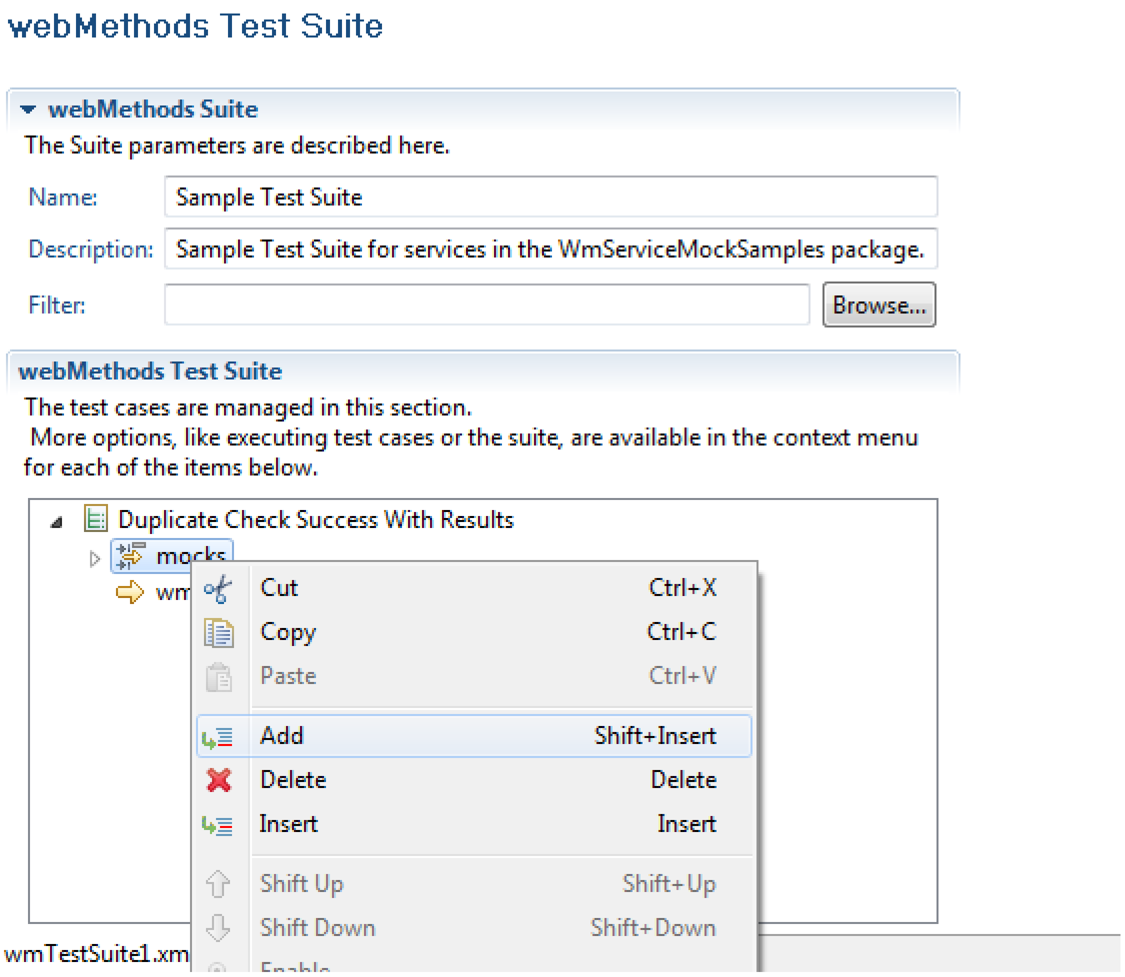
This section provides information on using mocks in test cases and settings to use when adding a mock. A sample Java mock data factory is also provided in this section as a reference.

6.1 Using Mocks

Mocks are used when resources that a service may require to properly execute may not be available when a test case or test suite is developed or executed. Mocks provide a means of simulating interaction with resources that are unavailable.

Follow the steps to create a mock:

1. Expand the test case for which the mock is to be defined, right click on mocks and choose **Add**.



webMethods Test Suite

webMethods Suite

The Suite parameters are described here.

Name: Sample Test Suite

Description: Sample Test Suite for services in the WmServiceMockSamples package.

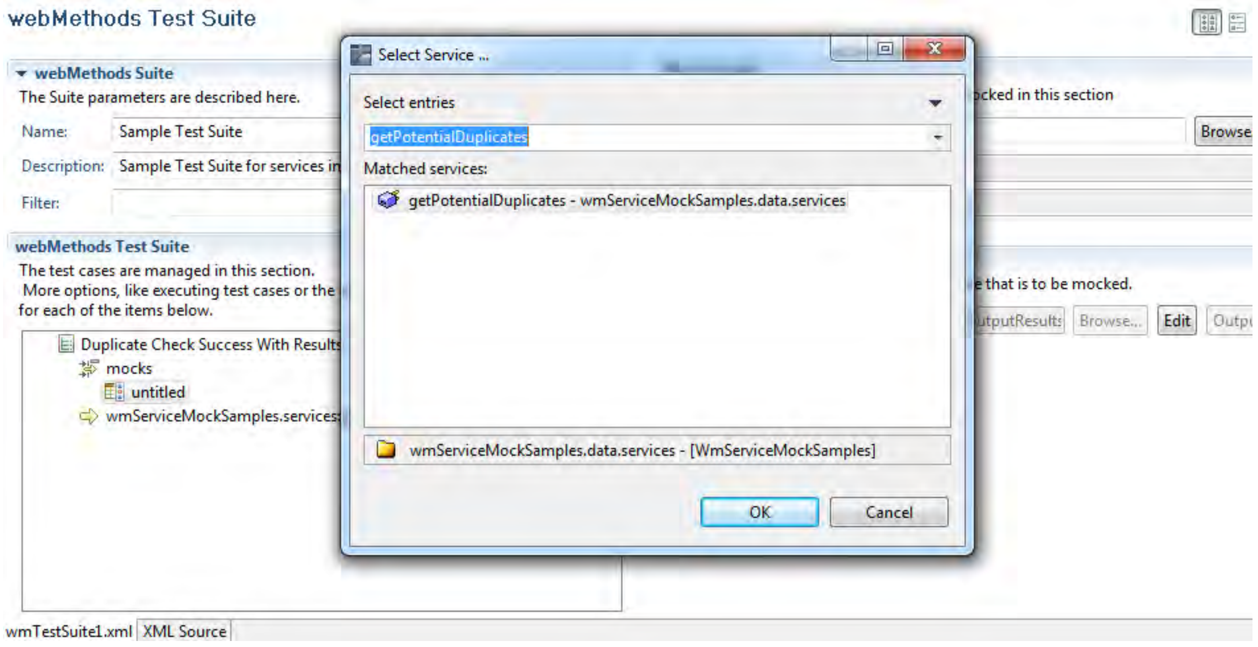
Filter:

webMethods Test Suite

The test cases are managed in this section.
More options, like executing test cases or the suite, are available in the context menu for each of the items below.

2. To select the service to be mocked, click **Browse** and use the Service Browser to select the service.

3. Enter the first few characters of the service name to reduce the list. Click ▾ to refresh the list and access preferences for the Service Browser.


webMethods Test Suite

webMethods Suite

The Suite parameters are described here.

Name: Sample Test Suite

Description: Sample Test Suite for services in

Filter:

webMethods Test Suite

The test cases are managed in this section.
More options, like executing test cases or the
for each of the items below.

Duplicate Check Success With Results
mocks
untitled
wmServiceMockSamples.services

Select Service ...

Select entries

getPotentialDuplicates

Matched services:

getPotentialDuplicates - wmServiceMockSamples.data.services

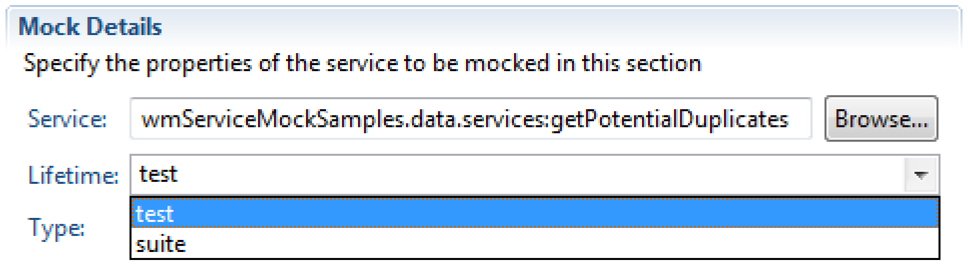
wmServiceMockSamples.data.services - [WmServiceMockSamples]

cked in this section

e that is to be mocked.

4. Select the required **Lifetime**.

   Valid selections are ***test*** (mock is effective only for the selected test case) and ***suite*** (mock will be effective for all of the test cases that follow in the test suite).


Mock Details

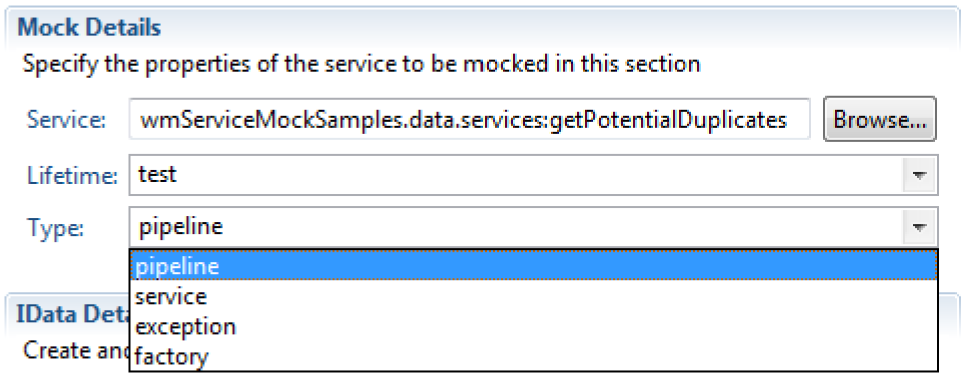
Specify the properties of the service to be mocked in this section

Service: wmServiceMockSamples.data.services:getPotentialDuplicates

Lifetime: test

Type:

5. Select the required **Type**.


Mock Details

Specify the properties of the service to be mocked in this section

Service: wmServiceMockSamples.data.services:getPotentialDuplicates

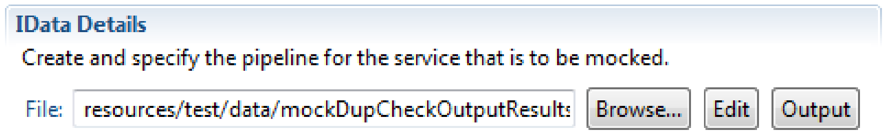
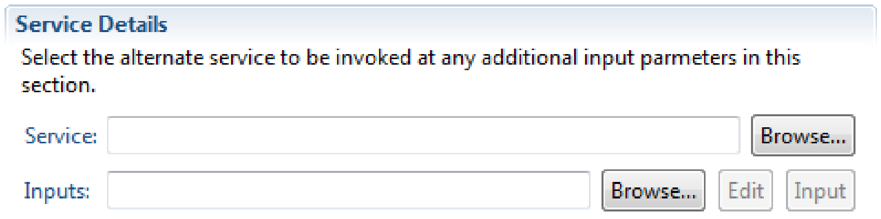
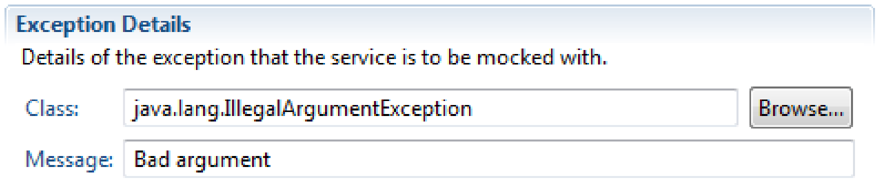
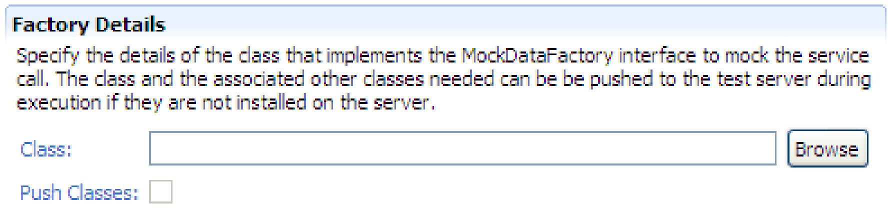
Lifetime: test

Type: pipeline

IData Det

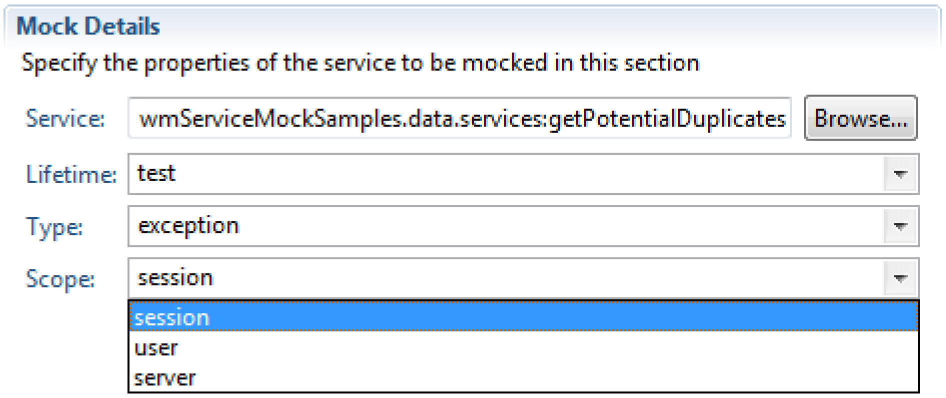
Create an

Following are the valid selections:

Type	Function	Details
pipeline	Service is intercepted and specified pipeline is returned (for details on creating or editing pipeline data see Inputs).	 IData Details Create and specify the pipeline for the service that is to be mocked. File: resources/test/data/mockDupCheckOutputResults
service	Service is intercepted and substituted with a call to the selected service (for details on working with the Service Browser, see Service Details)	 Service Details Select the alternate service to be invoked at any additional input parmeters in this section. Service: Inputs:
exceptio n	Service is intercepted and an exception is returned (for details on selecting exception classes, see Expected Output)	 Exception Details Details of the exception that the service is to be mocked with. Class: java.lang.IllegalArgumentException Message: Bad argument
factory	A call to the mocked service is intercepted and data is returned based on the input (for details on creating a mock factory, see Sample Mock Factory)	 Factory Details Specify the details of the class that implements the MockDataFactory interface to mock the service call. The class and the associated other classes needed can be be pushed to the test server during execution if they are not installed on the server. Class: Push Classes: ☐

6. Select **Scope**.

   It is recommended to use the ***session*** scope for most purposes. Scope selection is only allowed if the corresponding preference in enabled.



Mock Details

Specify the properties of the service to be mocked in this section

Service: wmServiceMockSamples.data.services:getPotentialDuplicates

Lifetime: test

Type: exception

Scope: session

6.2 Sample Mock Factory

The following code snippet illustrates the minimum requirements for creating a mock factory. The factory class and any other classes should be designed to evaluate the input data to the factory and return data relevant to that input in an IData format. The example below returns static data.

```
package com.wm.ps.serviceMock.samples;

import com.wm.app.b2b.server.BaseService;
import com.wm.app.b2b.server.invoke.ServiceStatus;
import com.wm.data.IData;
import com.wm.data.IDataFactory;
import com.wm.ps.serviceMock.MockDataFactory;

public class SampleMockDataFactory implements MockDataFactory
{
  private static final long serialVersionUID = 2L;

  public IData createData(BaseService baseService, IData pipeline, ServiceStatus
serviceStatus)
  {
    IData[] results = new IData[]{IDataFactory.create(new Object[][]{
        {"originationSource","W"},
          {"bizType","RT"},
          {"lockExpirationDate","20050427"},
          {"floatLoanIndicator","Y"},
          {"uwFinalDecisionCode","0"},
          {"uwDecisionExpiryDate","20050427"},
          {"canDate","20050427"},
          {"loanCloseStatusType","T"},
          {"fileReceivedAtRocDate","20050221"},
          {"loanReadyToFundIndicator","P"},
          {"regisDate","20051221"},
          {"loanSubmitToUwDate","20050427"},
          {"loanNumber","0000000001"},
          {"branch","TOTAL ADVANTEDGE LLC          "},
          {"underwritingDecisionCode","0"},
          {"underwritingDecisionExpirationDate","20050427"},
          {"lockDate","20051220"},
          {"lockIndicator","Y"},
          {"tmoLoanStageCode","3"},
          {"tmoLoanStageDate","20050427"},
          {"product","C30       "},
          {"borrowerFirstName",".          "},
          {"borrowerLastName","XX             "},
          {"propertyAddress","937 S MEYER                   "},
          {"propertyCity","TUCSON                    "},
          {"propertyState","AZ"},
          {"propertyZip","85701"}
      })};

    IData output = IDataFactory.create(new Object[][]{{"results", results}});
    return IDataFactory.create(new Object[][]{{"getPotentialDuplicatesOutput", output}});
  }
}
```

6.3 Mocks Beyond Unit Testing

Although WmTestSuite added the ability to mock service calls in Integration Server for unit testing, the feature is so powerful that its use cannot be limited to unit testing alone. One common case is to use the mocking capability to provide flow service instrumentation.

Using the wm.ps.serviceMock:loadMock service in the WmServiceMock package, any service can be mocked with an alternate service or class. The new service or Java class code can invoke any operations and then invoke the original mocked service. The mocking framework is intelligent enough to detect recursion and, as such, provides an instrumentation capability.

Mocks can be used to design test cases. A service being tested can also itself be mocked with other code. In such a scenario, the mocked test service can be replaced with other code that can execute pre and post-test operations. This can provide some basic functional testing capabilities for WmTest-Suite.

7 Advanced Options

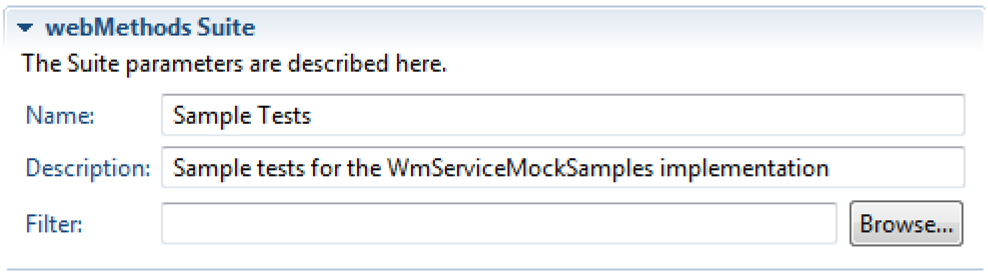
WMTestsuite provides advanced options that can enable the developer to extend the capabilities to create more powerful tests.

7.1 Pipeline Filter

The execution of test cases is initiated as a client to the Integration Server hosting the services to be tested. For this reason, the inputs supplied to the service during execution and the expected outputs need to be serialized over the client-server interaction. If the input or output pipeline contains non-serializable objects, these objects are either lost or seen incorrectly during test execution. In addition, the service input needs to be more dynamic in nature than the static pipeline setup in the test case. Pipeline Filter helps to resolve these issues.

The Pipeline Filter is set once for the entire suite and provides a callback mechanism for the test developer to inject code that can modify various pipeline objects during execution. The Pipeline Filter is a class that implements the com.wm.ps.test.PipelineFilter interface and enables a user to add, remove, or change variables in the pipeline that is created from files as pipelines created from files may not be able to persist custom java objects. The output pipeline from a service can also be filtered using the appropriate method. Only one such instance of the implementing class is created for the test suite and the name of test case is passed as a parameter.

The pipeline filter can be setup for the test suite in the main panel for the suite parameters as shown below:



The Suite parameters are described here.

Name: Sample Tests

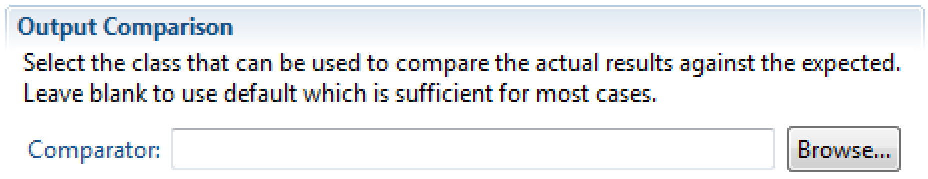
Description: Sample tests for the WmServiceMockSamples implementation

Filter:

7.2 Custom Comparator

Custom comparator provides an extension that can be used to extend, enhance, or replace the standard comparison of expected output.

Each test suite can have its comparator that can be specified from the user interface as shown below:



Select the class that can be used to compare the actual results against the expected. Leave blank to use default which is sufficient for most cases.

Comparator:

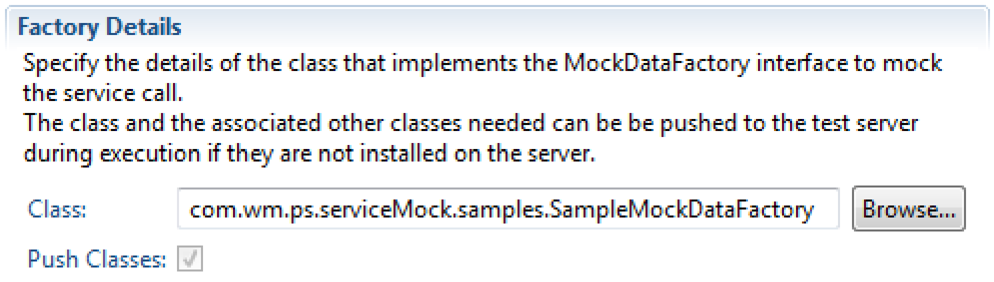
Custom comparators are Java classes that implement the com.wm.ps.test.ResultsValidator interface.

Custom comparators also provide a mechanism to execute operations pre and post service execution. Using custom comparators you can build some basic functional testing capability. For example, if a service writes to a database table, a custom comparator can compare the results from the service and check the destination table to confirm the operation executed successfully.

7.3 Client Mock Factory

The benefit of using a mock factory object as opposed to a service is that it provides a light-weight alternative that does not require the creation of a new service on the server.

WmTestSuite provides an option to dynamically push the classes needed for supporting the mock factory on the server during test execution. This option can be used to avoid the need for frequent restart when java objects are changing. Once stabilized, it would be helpful to deploy the code to the server as this feature is experimental in nature and will only work if the dependency tree is not too complex. The option to push the objects dynamically to the server can be set from the user interface as shown below:


Factory Details

Specify the details of the class that implements the MockDataFactory interface to mock the service call.
The class and the associated other classes needed can be be pushed to the test server during execution if they are not installed on the server.

Class: com.wm.ps.serviceMock.samples.SampleMockDataFactory

Push Classes: ✓

If the option is disabled as in the figure above, enable the corresponding preference as discussed in the Advanced section.

webMethods expression	JXPath equivalent
PosRequest/ns:Log/ns:Transaction[0]/@Flag	PosRequest[@name='ns:Log'][@name='ns:Transaction'][1][@name='@Flag']
PosRequest/ns:Log/ns:Transaction[0]/ns:BUnit[0]/ns:ID/*body	PosRequest[@name='ns:Log'][@name='ns:Transaction'][1][@name='ns:BUnit'][1][@name='ns:ID'][@name='*body']

7.4 XPath Expressions

The XPath expressions used in the expected output panel are different from the usual webMethods path expressions. As mentioned in the Expected Output section, the indices start at 1 instead of the 0 based webMethods indices.

WmTestSuite uses JXPath for evaluating XPath expressions. For details on JXPath expressions, visit http://commons.apache.org/jxpath/. One unique challenge is with variables containing special characters like @ and: in the name. As these have special meaning in JXPath expressions using these have to be written using a special syntax variant.

8 Test Suite Internals

In Software AG Designer, test suite editor provides a user interface to graphically and quickly develop test cases for Integration Server services, the test suite and the test cases are saved in an XML file. The Software AG Designer editor allows editing the XML source directly as long as the user is aware of the format and the associated schema.

It is not recommended to edit the XML file as it can be error-prone. It provides the option to automate the creation of test cases automatically by using code to generate the XML file directly. One such use case is the scenario where service inputs and outputs have been captured in an environment and test cases have to be generated to use these files for regression testing.

A sample XML test suite file in its simplified form can be as shown in the figure below:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<webMethodsTestSuite description="Sample tests for the WmServiceMockSamples implementation"
name="Sample Tests">
    <webMethodsTestCase description="Duplicate Check Success with IData results"
name="Duplicate Check Success With Results">
        <mock folder="wmServiceMockSamples.data.services" name="getPotentialDuplicates">
            <pipeline filename="resources/test/data/mockDupCheckOutputResults.xml"/>
        </mock>
        <service folder="wmServiceMockSamples.services" name="getDuplicateLoans">
            <input>
                <file filename="resources/test/data/dupCheckInput.xml"/>
            </input>
            <expected>
                <file filename="resources/test/data/dupCheckSuccessWithResults.xml"/>
            </expected>
        </service>
    </webMethodsTestCase>
    <webMethodsTestCase description="Duplicate Check Failure handled by the catch block"
name="Duplicate Check Catch Block">
        <mock folder="wmServiceMockSamples.data.services" name="getPotentialDuplicates">
            <exception class="java.lang.IllegalArgumentException" message="Bad argument"/>
        </mock>
        <service folder="wmServiceMockSamples.services" name="getDuplicateLoans">
            <input>
                <file filename="resources/test/data/dupCheckInput.xml"/>
            </input>
            <expected>
                <exception class="java.lang.IllegalArgumentException" message="Bad
argument"/>
            </expected>
        </service>
    </webMethodsTestCase>
</webMethodsTestSuite>
```

9 Java Unit Tests

WmTestSuite java API allows you to create pure JUnit test cases that can provide the same features that a user interface driven codeless test cases do.

The change when creating a WmTestSuite JUnit test case from the traditional test case is that the implementing class extends com.wm.ps.test.WmTestCase instead of junit.framework.TestCase. com.wm.ps.test.WmTestCase does extend the junit.framework.TestCase. The two important methods that are needed for creating test cases using the java API are:

- invokeService – The method to invoke a service on the server
- mockService – There are various variants of this method that allow the user to setup a mock for a service on the server.

A sample JUnit test case is provided here:

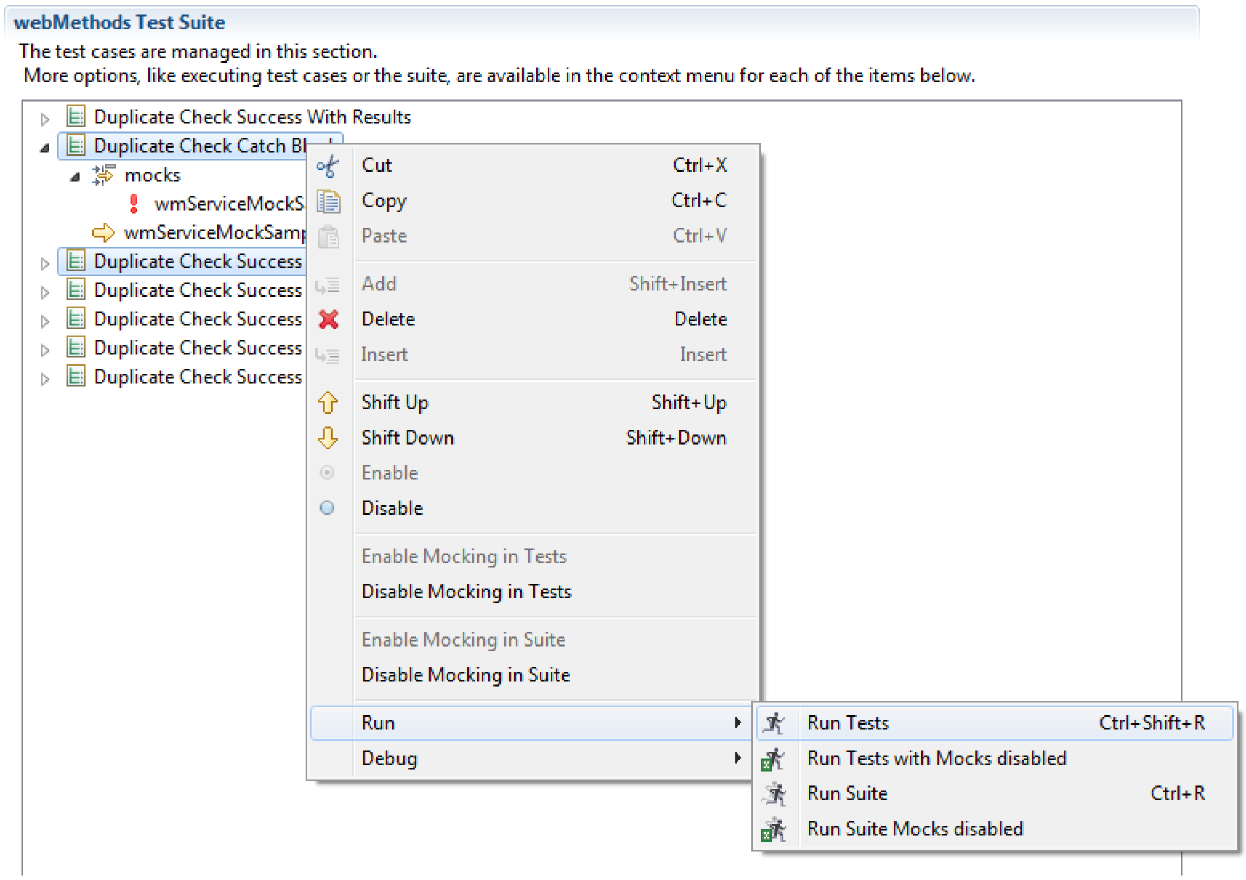
```java
package com.wm.ps.serviceMock.samples;

import com.wm.data.*;
import com.wm.ps.test.*;

public class DuplicateCheckTest extends WmTestCase
{
  public void testDupCheckCatchBlock() throws Exception
  {
    IData input = IDataFactory.create(new Object[][]{
        {"lienType", "1"},
        {"borrowerSSN", "111-11-1111"},
        {"propertyAddress", "937 S Meyer"},
        {"propertyZip", "85701"}
        });

    String exceptionText = "Bad argument";
    mockService("wmServiceMockSamples.data.services", "getPotentialDuplicates", new
               IllegalArgumentException(exceptionText));
    try
    {
      invokeService("wmServiceMockSamples.services", "getDuplicateLoans", input);
      assertFalse(true); //Control getting here means failure
    }
    catch (Exception e)
    {
      assertTrue(e.getMessage().endsWith(exceptionText));
    }
  }

   public void testDupCheckSucessWithResults() throws Exception
  {
    IData input = IDataFactory.create(new Object[][]{
    {"lienType", "1"},
    {"borrowerSSN", "111-11-1111"},
    {"propertyAddress", "937 S Meyer"},
    {"propertyZip", "85701"}
    });

    IData mockOutput =
       WmTestSuiteUtils.getIDataFromFile("resources/test/data/mockDupCheckOutputResults.xml");
    mockService("wmServiceMockSamples.data.services", "getPotentialDuplicates", mockOutput);
    IData output = invokeService("wmServiceMockSamples.services", "getDuplicateLoans", input);

    IDataCursor outCursor = output.getCursor();
    IData response = IDataUtil.getIData(outCursor, "response");
    IDataCursor responseCursor = response.getCursor();
    String creationTime = IDataUtil.getString(responseCursor, "@creationTime");
    assertNotNull(creationTime);
    assertEquals(28, creationTime.length());
    IData[] duplicateLoans = IDataUtil.getIDataArray(responseCursor, "duplicateLoans");
    assertEquals(duplicateLoans.length, 1);
  }
}
```

10 Executing Tests

This section provides details about executing test cases and test suites using the WmTestSuite.

10.1 Executing Test Cases

To execute test cases created with WmTestSuite, right click in the webMethods Tests section.



webMethods Test Suite

The test cases are managed in this section.
More options, like executing test cases or the suite, are available in the context menu for each of the items below.

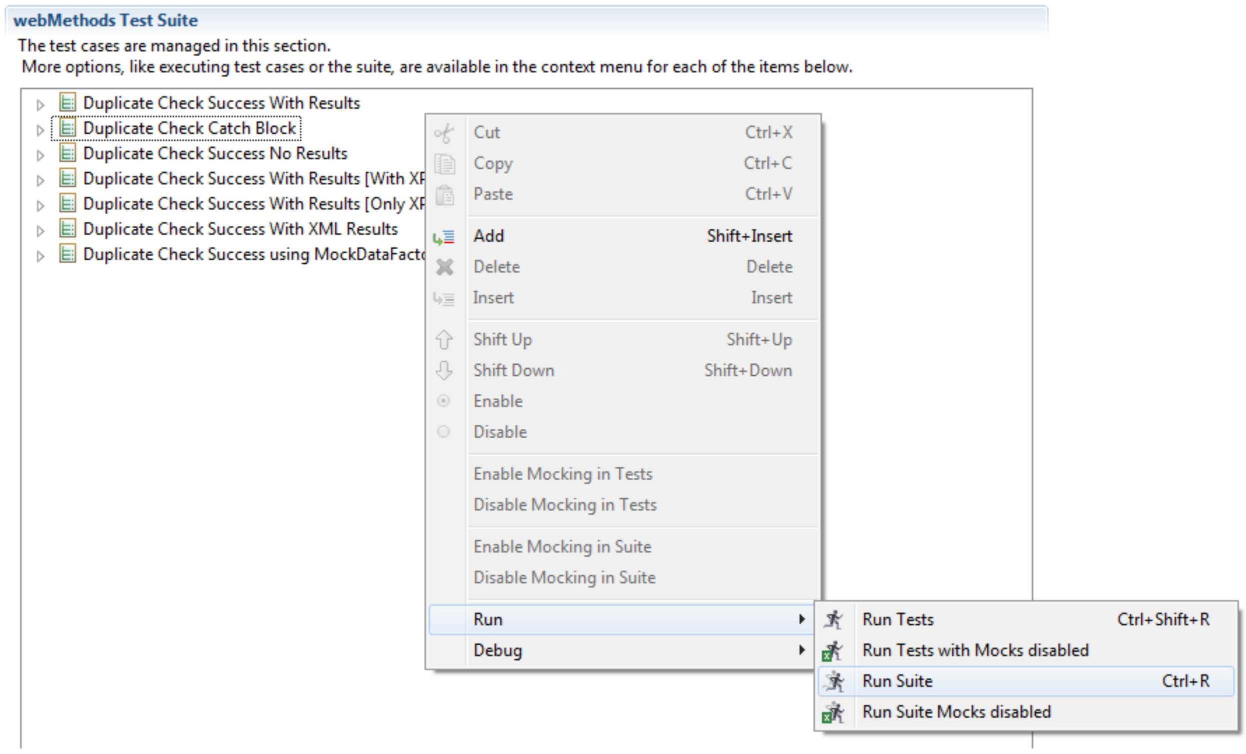
Following options appear (as shown in the figure above):

- **Run Tests** - executes the selected test case(s), using any mocks that are defined for the test case.
- **Run Test with Mocks Disabled** - executes the selected test case, disabling the mocks defined for the test case
- **Disable Test** - marks test case as disabled. Test will not be executed until it is enabled again
- **Disable Mocking in Tests** - marks mocks defined for selected test(s) as disabled. Mocks will not be executed until they are enabled again
- **Add** - adds another test case to the suite
- **Insert** - inserts another test case to the suite after the selected test case

- **Remove** - removes the selected test case from the suite

10.2 Executing Test Suites

To execute test suites created using WmTestSuite, right click in the webMethods Tests section.



webMethods Test Suite

The test cases are managed in this section.
More options, like executing test cases or the suite, are available in the context menu for each of the items below.

Duplicate Check Success With Results
Duplicate Check Catch Block
Duplicate Check Success No Results
Duplicate Check Success With Results [With XF
Duplicate Check Success With Results [Only XF
Duplicate Check Success With XML Results
Duplicate Check Success using MockDataFacto
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Add Shift+Insert
Delete Delete
Insert Insert
Shift Up Shift+Up
Shift Down Shift+Down
Enable
Disable
Enable Mocking in Tests
Disable Mocking in Tests
Enable Mocking in Suite
Disable Mocking in Suite
Run
Debug
Run Tests Ctrl+Shift+R
Run Tests with Mocks disabled
Run Suite Ctrl+R
Run Suite Mocks disabled

Following options appear (as shown in the figure above):

- **Run Suite** - executes the selected test suite, using any mocks that are defined for the test suite
- **Run Suite with Mocks Disabled** - executes the selected test suite, disabling the mocks defined for the test suite
- **Disable Suite** - marks test suite as disabled. The test suite will not be executed until it is enabled again
- **Disable Mocking in Suite** - marks mocks defined for this suite as disabled. Mocks will not be executed until they are enabled again
- **Shift Up** or **Shift Down** - changes the order of test cases in the test suite by shifting the selected test case up or down in the test suite

10.3 Debugging Java Code

Various components in WmTestSuite rely on java code.

Example: Java mock factory and pipeline filter classes.

Use the Debug menu to debug a java code. The support library jar files also have source code associated with them. Debugging into the source can be helpful to understand the internals of the test execution or to enhance capabilities of build new features like custom comparators.

11 Service Usage

Services on the server can be useful to accomplish tasks that may not be possible with the existing tools or utilities. Services in the WmServiceMock package is described below.

11.1 WmServiceMock Services

The services in the WmServiceMock package deal with the mocking aspects of testing. These services provide a way to enable or disable mocks for individual services or server-wide.

wm.ps.serviceMock:loadMock			
Sets up mocking for a service.			
Inputs	scope	session, user, or server. The default is session.	
	service	The name of the service to be mocked. No validation is performed on the name of the server or its existence.	
	mockObject	The mockObject is an object type. The behavior of the mock is controlled by the type of the actual object.	
		java.lang.String	The name of the alternate service to mock the mocked service with.
		java.lang.Exception	The exception to be thrown for the mocked service.
		com.wm.data.IData	The fixed pipeline to return for the mocked service
		com.wm.ps.serviceMock.MockDataFactory	The implementation of the factory objects that creates the dynamic pipeline for the mocked service.
	parms	Optional document containing all extra parameters for the alternate service. This parameter is only needed when mocking a service with an alternate service and the alternate service needs additional inputs.	
Outputs	None		

wm.ps.serviceMock:clearMock		
Removes mocking for a service.		
Inputs	scope	session, user or server. The default is session.

	service	The name of the service to be mocked. No validation is performed on the name of the server or its existence.
Outputs	None	

wm.ps.serviceMock:clearAllMocks	
Removes mocking for all services.	
Inputs	None
Outputs	None

wm.ps.serviceMock:suspendMocks	
Suspends mocking for all services.	
Inputs	None
Outputs	None

wm.ps.serviceMock:resumeMocks	
Resumes mocking for all services, for which it was suspended.	
Inputs	None
Outputs	None

wm.ps.serviceMock:getMockedServices		
Retrieves the list of services that are currently mocked in all the scopes.		
Inputs	None	
Outputs	mockedServices	The list of services that are currently mocked in the session, user, or server scope.

11.2 References

- Javadoc API Reference - The javadoc reference for the WmTestSuite java API. This is useful for advanced WmTestSuite features as well as creating pure java unit tests for Integration Server services.
- JXPath– Documentation for JXPath API