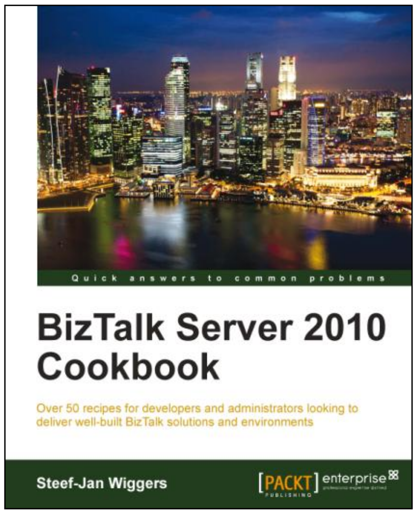
BizTalk Server 2010 Cookbook

Steef-Jan Wiggers



Chapter No. 9
"Testing BizTalk Artifacts"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.9 "Testing BizTalk Artifacts"

A synopsis of the book's content

Information on where to buy this book

About the Author

Steef-Jan Wiggers is an IT architect with over 13 years of experience as a Consultant, Technical Lead Developer, and Application Architect, specializing in custom applications, enterprise application integration (BizTalk), Web services, and Windows Azure. He has experience in architecting, designing, developing, and supporting sophisticated and innovative software using many different Microsoft technologies and products. Steef-Jan is very active in the BizTalk community as a blogger, Wiki author/editor, MSDN forums writer, and public speaker. He has been awarded the Microsoft Most Valuable Professional (MVP) award in 2010 for his contributions to the world-wide BizTalk Server community and has been re-awarded in July 2011.

Steef-Jan lives in the Netherlands, is married to Lian, and has three lovely children, Stan, Ellis, and Cato. Last but not the least, they are accompanied by their English Cocker Spaniel, Barry. Steef-Jan is certified in MCDBA, MCSD, MCSD.NET, MCSA, MCAD, MCTS: BizTalk Server BizTalk Server 2006, BizTalk Server 2006 R2, and BizTalk Server 2010.

Steef-Jan works as a Specialist Knowledge Provider for Ordina, which lays the foundation of its clients' future success by offering a coherent proposition of Consulting, IT, and Outsourcing services. Within Ordina, Steef-Jan is responsible for BizTalk Expertise Group — sharing knowledge, exchanging experience, planning meetings, and facilitate courses. He manages the Line Of Buisness (LOB) BizTalk to create/maintain LOB year plans, coordinates contacts with the BizTalk community (BTUG) in the Netherlands and Sweden, and partners with Microsoft for BizTalk.

Steef-Jan has been a Technical Reviewer of the BizTalk 2010 Patterns book written by Dan Rosanova and is currently involved as a Technical Reviewer for the forthcoming book (MCTS): Microsoft BizTalk Server 2010 (70-595) Certification Guide. He is also a co-author of a series of BizTalk Server Administration books.

BizTalk Server 2010 Cookbook

In your day-to-day job as a BizTalk developer, administrator, or consultant, you will face challenges when it comes to deploying a BizTalk Server environment, advising clients on integration with BizTalk, developing BizTalk solutions, working with its components such as Business Rules, or a feature such as AppFabric Connect. As an administrator, you will be responsible for keeping BizTalk healthy. The practical recipes in this book can strengthen your skills and knowledge. A developer can face challenges during implementation of functionality in an orchestration or while testing the solution. Having recipes in this book on how to deal with these challenges can be of tremendous value. Some of the recipes in this book will show you how to use other BizTalk-related tooling from Microsoft, the community, and third parties which can greatly improve your productivity as a developer or administrator. This book will provide you with guidance on using out of the box BizTalk capabilities combined with the capabilities offered by BizTalk tools found on CodePlex, and the Microsoft Download Center. BizTalk Server 2010 Cookbook is a practical guide for developers and administrators, which they can use as reference guide for their day-to-day job, making their lives easier.

What This Book Covers

Chapter 1, Setting up a BizTalk Server Environment, will provide the reader with guidance on setting up a robust and healthy BizTalk environment, from its inception to its deployment. Recipes in this chapter will show the reader how to use some of the community and Microsoft tools to validate, test, and tune the BizTalk environment and how to set up and configure critical components, such as MSDTC and SSO.

Chapter 2, *BizTalk Server Automation: Patterns,* gives an idea about patterns that can be applied to orchestrations. Recipes discussed in this chapter will show how to implement some of the common integration patterns with BizTalk Server 2010.

Chapter 3, BizTalk Server Instrumentation, Error Handling, and Deployment, introduces the reader to the concepts of instrumentation, error handling, and deployment of BizTalk solutions. Recipes discussed in this chapter will show how to implement instrumentation to BizTalk solutions, using logging and tracing, how to implement error handling, and how to deploy BizTalk solutions, using out of the box BizTalk features or the BizTalk deployment framework.

Chapter 4, Securing your Message Exchange, explains about security in BizTalk messaging context. It will show how to provide message security by using encryption and decryption, or signing and verifying. It will also touch transport security using SSL.

Chapter 5, WCF Services with BizTalk, introduces the concept of communicating with WCF Services through BizTalk Server. Recipes discussed in this chapter will show how to consume a WCF Service, expose schemas as a WCF Service, and other related possibilities.

Chapter 6, BizTalk AppFabric Connect, explains about the new BizTalk Server 2010, which shifts the boundaries of BizTalk to Windows Azure. Recipes discussed in this chapter will demonstrate how to expose orchestrations, and LOB systems through Window Azure Service Bus.

Chapter 7, Monitoring and Maintenance, will provide the reader with practical recipes on keeping BizTalk Server healthy and what monitoring solutions are best suitable in a given scenario. This chapter will demonstrate the capabilities found in community tooling, SCOM, and alternative monitoring product BizTalk360.

Chapter 8, Applying Rules, introduces the concepts of BizTalk Business Rules Engine. It will demonstrate how to use BRE with and without using the BizTalk Server runtime.

Chapter 9, Testing BizTalk Artifacts, will provide the reader with a couple of recipes to enable testing of the different BizTalk artifacts, such as schemas, pipelines, maps, and orchestrations. Testing can be done using the test capabilities of Visual Studio in conjunction with community test tooling.

9
Testing BizTalk Artifacts

In this chapter, we will cover:

- Testing BizTalk artifacts inside Visual Studio
- Unit testing a BizTalk solution with BizUnit
- Applying code coverage to a BizTalk orchestration
- Testing BizTalk solutions with BizMock
- Using the BizTalk Map Test Framework
- Testing pipelines and pipeline components

Introduction

Testing is an important part when you are developing BizTalk solutions. Before deploying your solution into production, you need to be confident that it will perform, and do the job it is intended to do. A developer is responsible for creating robust and solid BizTalk solutions. He/she needs to test the BizTalk solutions with its artifacts before it is deploy to a test environment, where integration and other tests will be performed.

With the BizTalk Server 2009, the unit test feature was introduced, which offered built-in developer support for testing schemas, maps, and pipelines. This was a great enhancement for developers. Before that, developers had to rely on frameworks such as BizUnit. Now, the developer has one IDE for developing and testing the BizTalk solutions. Yet, the frameworks such as BizUnit or BizMock still have their value as they provide a rich API. They also offer great flexibility and control over your tests.

For last couple of years, people in the community have built frameworks to support developers in testing. Frameworks such as BizUnit, BizMock, and the BizTalk Map Test Framework are all available through CodePlex and support the BizTalk Server 2010. Another useful tool for testing is the Pipeline Testing Library made by Tomas Restrepo.

How to use these frameworks and the test library will be discussed in the recipes which you will find in this chapter. The focus in this chapter is on testing artifacts during development and not integration, load, or performance testing. You will find recipes to test the following artifacts:

- Document schemas
- Orchestrations
- Pipelines
- Pipeline components
- Maps

Testing BizTalk artifacts inside Visual Studio

During development of your BizTalk solution, you can do some of the testing before you even deploy your solution. You can leverage the capabilities of testing in Visual Studio using, for instance, the **Test Map** menu item for testing a map. Besides that you can use the Unit Test functionality offered by Visual Studio. The Unit Testing Framework (`http://msdn.microsoft.com/en-us/library/ms243147(v=VS.90).aspx`) supports unit testing in Visual Studio and allows you to create unit tests (`http://msdn.microsoft.com/en-us/library/ms182517.aspx`). In this recipe, you will learn the test capabilities offered by Visual Studio in BizTalk projects. The following tasks will be shown in this recipe:

- Validating an XML document instance
- Testing a map
- Unit test a schema

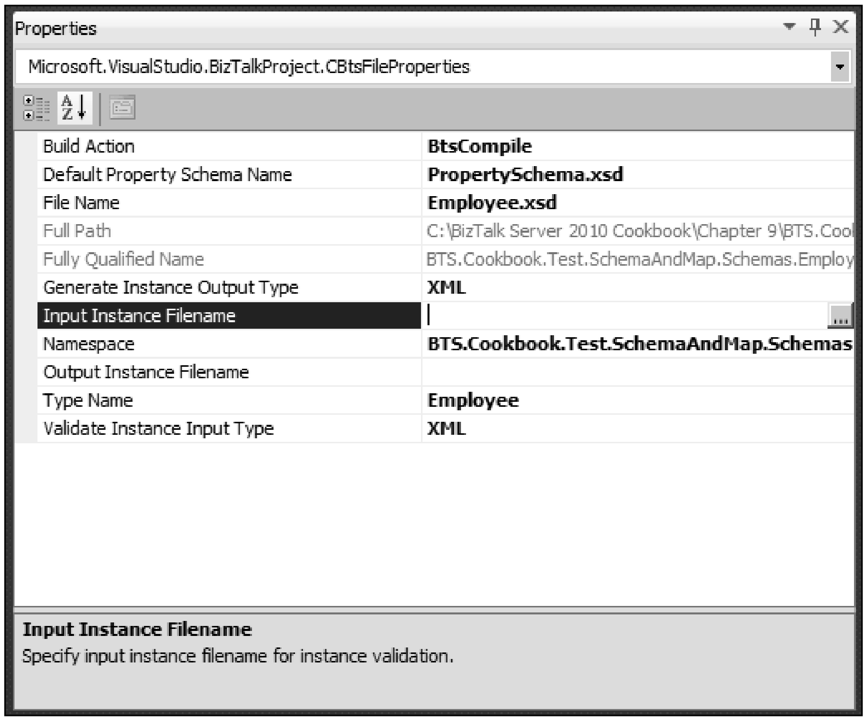
Getting ready

Download the code (`BTS.Cookbook.Test.SchemaAndMap`) belonging to this book from the Packt website or use your own BizTalk project.

How to do it...

Performing the following steps will enable you to test an instance of a document schema:

1. Navigate to a schema you want to test in your BizTalk project.
2. In **Properties**, select **Input Instance Filename**, as shown in the following screenshot, and click on ellipsis (**...**):



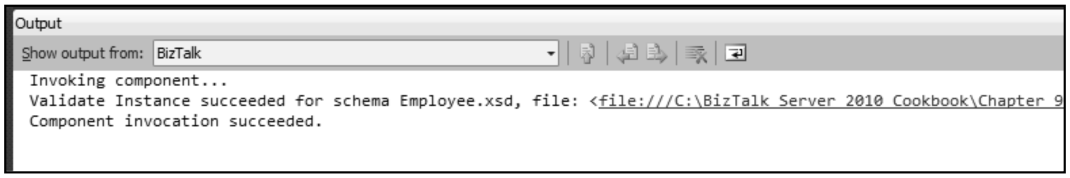
Properties

Microsoft.VisualStudio.BizTalkProject.CBtsFileProperties

Build Action	BtsCompile
Default Property Schema Name	PropertySchema.xsd
File Name	Employee.xsd
Full Path	C:\BizTalk Server 2010 Cookbook\Chapter 9\BTS.Cool
Fully Qualified Name	BTS.Cookbook.Test.SchemaAndMap.Schemas.Employ
Generate Instance Output Type	XML
Input Instance Filename	
Namespace	BTS.Cookbook.Test.SchemaAndMap.Schemas
Output Instance Filename	
Type Name	Employee
Validate Instance Input Type	XML

Input Instance Filename
Specify input instance filename for instance validation.

3. Navigate to the XML file you want to validate (that is, test).
4. Click on **Open**.
5. Right-click on the schema and select **Validate Instance**.
6. In the **Output** window, you will see the result, as depicted in the following screenshot:



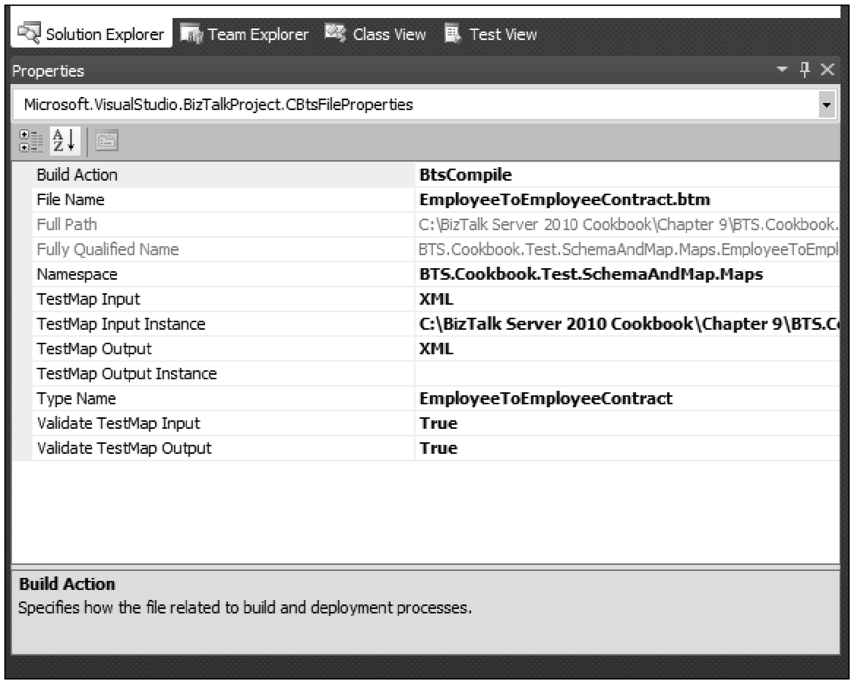
Output

Show output from: BizTalk

```
Invoking component...
Validate Instance succeeded for schema Employee.xsd, file: <file:///C:\BizTalk Server 2010 Cookbook\Chapter 9
Component invocation succeeded.
```

Performing the following steps will enable you to test a map:

1. In **Solution Explorer**, right-click on the map to test and select **Properties**.
2. Set **TestMap Input** and **TestMap Output** to **XML**.
3. Set the **TestMap Input Instance** property to the path of the instance you want to use as input to test the map:

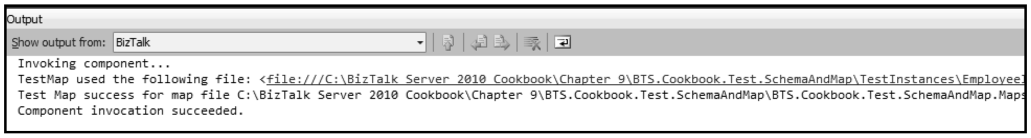


Microsoft.VisualStudio.BizTalkProject.CBtsFileProperties

Build Action	BtsCompile
File Name	EmployeeToEmployeeContract.btm
Full Path	C:\BizTalk Server 2010 Cookbook\Chapter 9\BTS.Cookbook.
Fully Qualified Name	BTS.Cookbook.Test.SchemaAndMap.Maps.EmployeeToEmpl
Namespace	BTS.Cookbook.Test.SchemaAndMap.Maps
TestMap Input	XML
TestMap Input Instance	C:\BizTalk Server 2010 Cookbook\Chapter 9\BTS.C
TestMap Output	XML
TestMap Output Instance	
Type Name	EmployeeToEmployeeContract
Validate TestMap Input	True
Validate TestMap Output	True

Build Action
Specifies how the file related to build and deployment processes.

4. Right-click on the map and select **Test Map**. This will produce an output document that can be validated for accuracy. Check the **Output** window in Visual Studio for details in case of a failure:



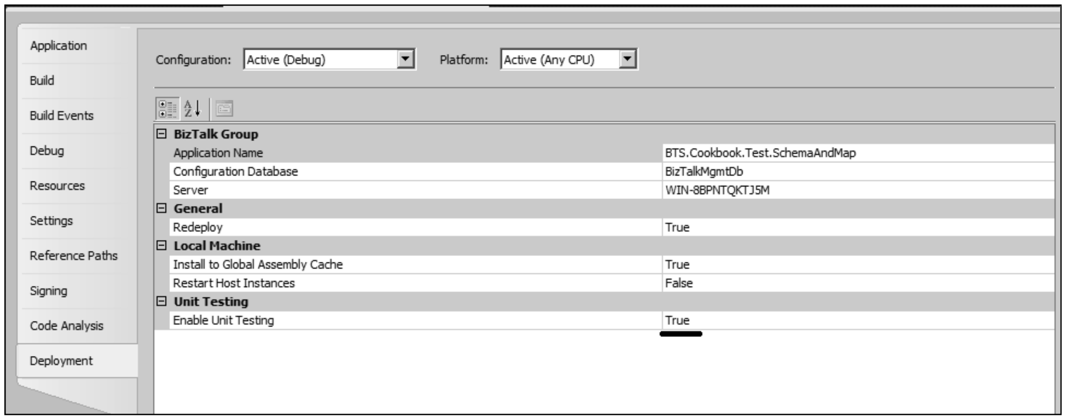
```
Invoking component...
TestMap used the following file: <file:///C:\BizTalk Server 2010 Cookbook\Chapter 9\BTS.Cookbook.Test.SchemaAndMap\TestInstances\Employee
Test Map success for map file C:\BizTalk Server 2010 Cookbook\Chapter 9\BTS.Cookbook.Test.SchemaAndMap\BTS.Cookbook.Test.SchemaAndMap.Maps
Component invocation succeeded.
```

5. Click on the provided link for the output to view the result of the map.

Performing the following steps will enable you to unit test your BizTalk project containing schema(s), map(s), or pipeline(s). Here, the unit test of a map and schema will be outlined:

1. Right-click on the BizTalk project containing the schemas.

2. Select the **Deployment** tab and set the **Enable Unit Testing** property to **True**. This will modify the BizTalk artifacts so that they inherit from `TestableSchemaBase`, `TestableMapBase`, or `TestablePipelineBase`:



Configuration: Active (Debug) Platform: Active (Any CPU)

BizTalk Group	
Application Name	BTS.Cookbook.Test.SchemaAndMap
Configuration Database	BizTalkMgmtDb
Server	WIN-8BPNTQKTJ5M
General	
Redeploy	True
Local Machine	
Install to Global Assembly Cache	True
Restart Host Instances	False
Unit Testing	
Enable Unit Testing	True

3. Build the project and then click on **Rebuild Solution**.
4. You can repeat these steps for other BizTalk projects which you want to unit test.
5. The next step is to build a unit test applicable for BizTalk projects.
6. Add a test project to your solution and give it an appropriate name.
7. Within the test project, set references to your BizTalk projects on which you wish to perform the unit test.
8. Set the reference to `Microsoft.BizTalk.TestTools.dll` and `Microsoft.XLANGs.BaseTypes.dll`.
9. In **Unit Test Class**, you can test methods, as shown in the following code snippet for validating an XML document instance and/or test map:

```
[TestMethod]
  public void ValidateEmployeeInstanceTest()
  {
    BTS.Cookbook.Test.SchemaAndMap.Schemas.Employee emp = new
    Schemas.Employee();
    bool success = false;
    success = emp.ValidateInstance(@"C:\BizTalk Server 2010
    Cookbook\Chapter 9\BTS.Cookbook.Test.SchemaAndMap\
    TestInstances\EmployeeInstance.xml",
    Microsoft.BizTalk.TestTools.Schema.OutputInstanceType.XML);
    Assert.IsTrue(success);
  }
[TestMethod()]
```

```
public void MapOutputEmployeeToEmployeeContractTest()
{
  BTS.Cookbook.Test.SchemaAndMap.Maps.
  EmployeeToEmployeeContract emp2empcon = new
  Maps.EmployeeToEmployeeContract();
  string inputEmployee = @"C:\BizTalk Server 2010
  Cookbook\Chapter 9\BTS.Cookbook.Test.SchemaAndMap\
  TestInstances\EmployeeInstance.xml";
  string outputEmployeeContract = @"C:\BizTalk Server 2010
  Cookbook\Chapter 9\BTS.Cookbook.Test.SchemaAndMap\
  TestInstances\EmployeeContract_Output.xml";
  emp2empcon.TestMap(inputEmployee, Microsoft.BizTalk.
  TestTools.Schema.InputInstanceType.Xml,
  outputEmployeeContract, Microsoft.BizTalk.TestTools.
  Schema.OutputInstanceType.XML);
  Assert.IsTrue(File.Exists(outputEmployeeContract));
}
```

10. You can test the methods by clicking on **Run all Tests in Solution**.

How it works...

Testing of schemas can be done within a BizTalk project by validating an instance of that schema. You provide the location of the instance in the schema file property **Input Instance Filename**. The other schema file property which needs to be set is **Validate Instance Input Type** to configure the format of the instance message identified by the **Input Instance Filename** property. By choosing **Validate Instance**, the specified document instance will be loaded and validated to the selected schema.

Testing of maps can also be done within a BizTalk project containing them. Again, you have to set some properties. These involve properties of the map you select. You will find all the properties that need to be set when testing a map in the following table:

Property	Description
Validate TestMap Input	It is a Boolean value indicating whether the source document will be validated against the source schema or not.
Validate TestMap Output	It is a Boolean value indicating whether the output document should be validated against the destination schema or not.
TestMap Input Instance	It is a path to the file which contains an instance of the source schema. It is used when the TestMap Input property is not set to the Generate Instance.

Property	Description
TestMap Input	It indicates the origin of the source document. If it is set to the Generate Instance, BizTalk will generate an instance in its memory that contains values for all attributes and elements in the source schema. If it is set to XML or Native, BizTalk will look for an instance of the document in the location specified in the TestMap Input Instance property. Native indicates a non-XML file such as a flat file.
TestMap Output	It indicates the format of the output document. The document will be output to a file stored in a temporary directory on Windows and accessible through Visual Studio.

By choosing **Test Map**, the specified source document will be loaded and the mapping will occur. The output of the mapping will be either stored in a location that has been provided or to a temporary directory provided through a link in the output window of Visual Studio.

Unit testing a schema means the `ValidateInstance` method is used, which is similar to choosing **Validate Instance** on a schema in a BizTalk project. The `ValidateInstance` method is a member of the `TestabaleSchemaBase` class belonging to the namespace `Microsoft.XLANGs.BaseTypes`. While calling this method you have to provide the path to the document and the output instance type. This method will return a Boolean value indicating whether validation was successful (`true`) or not (`false`).

Unit testing a map means that the `TestMap` method is used, which is similar to choosing **Test Map** on a map in a BizTalk project. The `TestMap` method is a member of the `TestabaleMapBase` class belonging also to the namespace `Microsoft.XLANGs.BaseTypes`. While calling this method you have to provide the path to a source document (instance), type of the instance, the path to the location for the output document (instance), and the type of output instance. This method will not return anything (that is, void method). The way to validate that the map has been executed successfully, is to validate that the output was created.

There's more...

On the TechNet Wiki, you can find an article called **Load Testing BizTalk Server Solutions with Visual Studio 2010** (`http://social.technet.microsoft.com/wiki/contents/articles/load-testing-biztalk-server-solutions-with-visual-studio-2010.aspx`). This article shows how to use Visual Studio test projects for the purpose of load testing a BizTalk Server application including how to create unit tests, how to create load tests, and how to configure load tests to capture the performance counter data required to determine the **Maximum Sustainable Throughput** (**MST**) of a BizTalk Server application:

Michael Stephenson has written an extensive series of blog post on testing in the BizTalk Server (targeted on BizTalk 2009). You can find that post in the document called **BizTalk Testing Guidance – Revisited** at `http://geekswithblogs.net/michaelstephenson/archive/2008/12/12/127828.aspx`.

See also

- Refer to the *Unit testing a BizTalk solution with BizUnit*, *Testing BizTalk solutions with BizMock*, and *Using the BizTalk Map Test Framework* recipes discussed later in this chapter

Unit testing a BizTalk solution with BizUnit

BizUnit is a framework for testing BizTalk artifacts. It offers a flexible and extensible declarative test framework. The BizUnit framework has no dependency on Visual Studio unit testing. Yet, you can leverage the framework in the Visual Studio Test Project.

A developer authors the BizUnit test case(s) in XML or code (BizUnit object model), and by doing so he/she configures how the test framework should execute the case(s). A test case is made up of three stages:

- Test setup
- Test execution
- Test cleanup

Each stage in a test case can have zero or more test steps, and these test steps are, in general, autonomous. This means that the state can be flowed between each state if required by using the `context` object which is passed to each test step by the framework.

Currently, BizUnit is at its fourth version supporting BizTalk 2010 (.NET 4.0) and this recipe will show how to use BizUnit 4.0 to unit test an orchestration in BizTalk 2010. The test case demonstrated in this recipe will execute a test case through code.

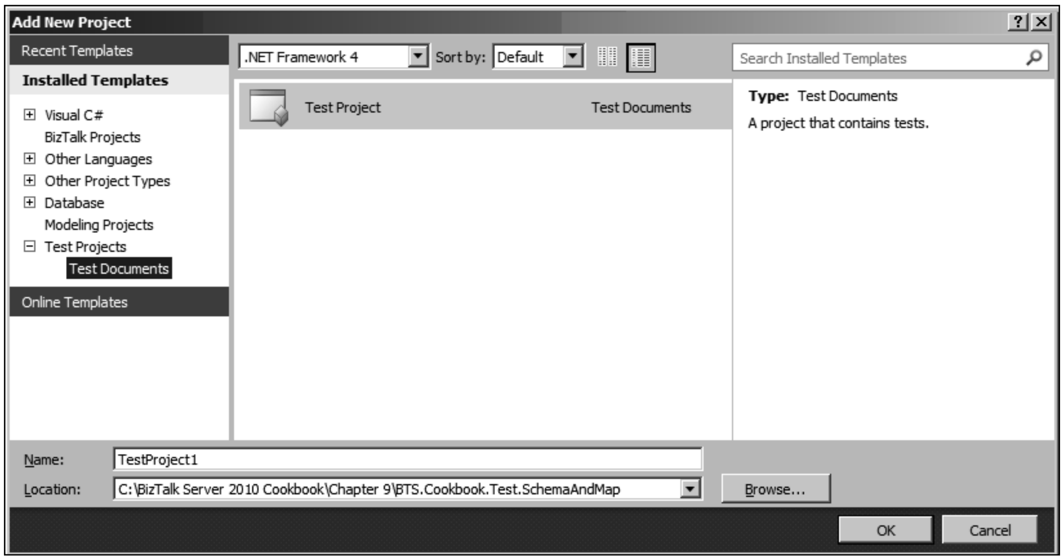
Getting ready

Download BizUnit from CodePlex (`http://bizunit.codeplex.com/releases/view/66850`). Install BizUnit on your machine. For reference, you can download the sample solution belonging to this chapter from the Packt website.

How to do it...

The following steps will show how to perform automated tests with BizUnit:

1. Open the solution you want to test in Visual Studio 2010.
2. Add a test project to the solution:



Type: Test Documents
A project that contains tests.

Name: TestProject1
Location: C:\BizTalk Server 2010 Cookbook\Chapter 9\BTS.Cookbook.Test.SchemaAndMap

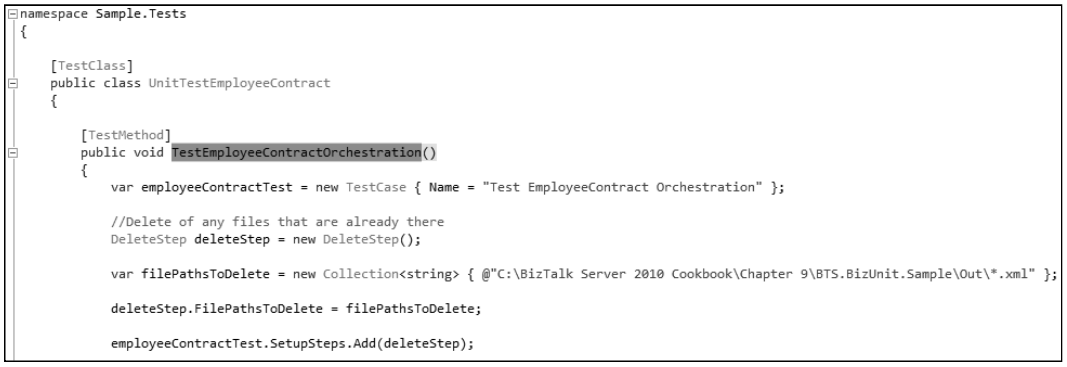
3. Give the project an appropriate name.
4. Click on **OK**.
5. Next, you need to add the BizUnit references to the test project. You will find them at `<install path>\BizUnit\BizUnit 4.0\Bins`. Reference to the following assemblies needs to be added:
   - `BizUnit.dll`
   - `BizUnit.TestSteps.dll`
   - `BizUnit.TestSteps.BizTalk.dll`
6. Now you can create your test methods. Rename `UnitTest1.cs` to an appropriate name if you desire.
7. Add the appropriate `using` statements to the `test` class.

8. Create a test method and add the following code snippet:

```
[TestMethod]
  public void TestEmployeeContractOrchestration()
  {
    var employeeContractTest = new TestCase { Name = "Test
    EmployeeContract Orchestration" };
    //Delete of any files that are already there
    DeleteStep deleteStep = new DeleteStep();
    var filePathsToDelete = new Collection<string>
    { @"C:\BizTalk Server 2010 Cookbook\Chapter
    9\BTS.BizUnit.Sample\Out\*.xml" };
    deleteStep.FilePathsToDelete = filePathsToDelete;
    employeeContractTest.SetupSteps.Add(deleteStep);
    //Create the test step
    var testStep = new CreateStep();
    //Where are we going to create the file
    //Change directory to your own requirement
    testStep.CreationPath = @"C:\BizTalk Server 2010
    Cookbook\Chapter 9\BTS.BizUnit.Sample\
    In\EmployeeInstance.xml";
    var dataLoader = new FileDataLoader();
    //Where are we getting the file from
    //Change directory to your own requirement
    dataLoader.FilePath = @"C:\BizTalk Server 2010
    Cookbook\Chapter 9\BTS.BizUnit.Sample\TestInstances\
    EmployeeInstance.xml";
    testStep.DataSource = dataLoader;
    employeeContractTest.ExecutionSteps.Add(testStep);
    //Create a validating read step
    //We should only have one file in the directory
    var validatingFileReadStep = new FileReadMultipleStep
    {
      DirectoryPath = @"C:\BizTalk Server 2010 Cookbook\Chapter
      9\BTS.BizUnit.Sample\Out\",
      SearchPattern = "*.xml",
      ExpectedNumberOfFiles = 1,
      Timeout = 3000,
      DeleteFiles = true
    };
    //Create an XML Validation step
    //This will check the result against the XSD for the Employee
    document
    var validation = new XmlValidationStep();
    var employeeSchema = new SchemaDefinition
```

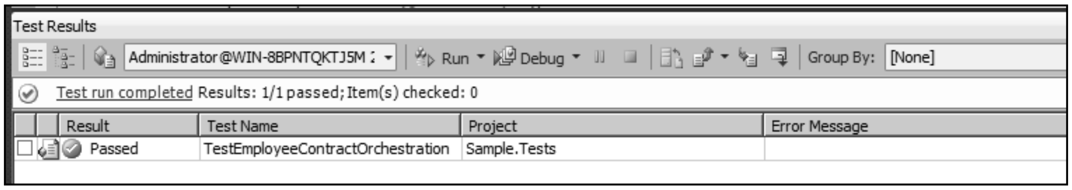
```
        {
          XmlSchemaPath =
          @"C:\BizTalk Server 2010 Cookbook\Chapter
          9\BTS.BizUnit.Sample\BTS.BizUnit.Sample.Schemas\
          EmployeeContract.xsd",
          XmlSchemaNameSpace =
          "http://BTS.BizUnit.Sample.Schemas.EmployeeContract"
        };
        validation.XmlSchemas.Add(employeeSchema);
        validatingFileReadStep.SubSteps.Add(validation);
        employeeContractTest.ExecutionSteps.
        Add(validatingFileReadStep);
        //Execute Tests
        var bizUnit = new BizUnit.BizUnit(employeeContractTest);
        bizUnit.RunTest();
      }
```

9. Build the test project.
10. Test the project by clicking on **Run Tests in Current Context** (refer to the following screenshot) or **Run all Tests in Solution**:



```
namespace Sample.Tests
{
    [TestClass]
    public class UnitTestEmployeeContract
    {
        [TestMethod]
        public void TestEmployeeContractOrchestration()
        {
            var employeeContractTest = new TestCase { Name = "Test EmployeeContract Orchestration" };

            //Delete of any files that are already there
            DeleteStep deleteStep = new DeleteStep();

            var filePathsToDelete = new Collection<string> { @"C:\BizTalk Server 2010 Cookbook\Chapter 9\BTS.BizUnit.Sample\Out\*.xml" };

            deleteStep.FilePathsToDelete = filePathsToDelete;

            employeeContractTest.SetupSteps.Add(deleteStep);
```

11. Check if the test has run successfully:



Test run completed Results: 1/1 passed; Item(s) checked: 0

Result	Test Name	Project	Error Message
Passed	TestEmployeeContractOrchestration	Sample.Tests	

How it works...

A BizUnit test case in this recipe is created through code. First a `TestCase` is instantiated and then the first step is to delete any files in the location, where the bound port will send its messages to. This is necessary to bring the status to its original state, where no processing of messages has taken place. The first stage of the test case is to put it up-front, but you can choose to have this stage at the end.

The next step is to define the test step(s) for the unit test to execute. The `TestStep` object has a `CreationPath` property that defines the path (directory and filename) that BizTalk should be watching with a Receive port. A `FileLoader` object is instantiated to enable to assign the `FilePath` property. This property contains the location to the XML message which is going to be offered to the orchestration. Now, this step is fully configured and added to the `TestCase` object:

```
employeeContractTest.ExecutionSteps.Add(testStep)
```

The following steps involve checking the execution of the orchestration. Does the file exist in the output directory (has the orchestration processed the incoming message and sent it to the `out` folder), and does the message in the output directory adhere to the schema (for example, `EmployeeContract`)? A `FileReadMultiStep` object is created and a few properties are set:

- The `DirectoryPath` property which defines the directory to read
- The `SearchPattern` property for the type of file
- `ExpectedNumberOfFiles` to define the number of files to be expected
- `TimeOut` to define the time (milliseconds) when timeout will occur
- The `DeleteFiles` property to determine whether or not to delete the file

To determine if the output message adheres to the schema a `ValidationStep` object and a `SchemaDefinition` object are instantiated. Then, for the next steps two more properties are set:

- `XmlSchemaPath` to determine the location of the schema
- `XmlSchemaNamespace` to set the namespace (that is, `targetNamespace`)

To the `validation.XmlSchemas.Add` method, the instance of the `SchemaDefinition` object is offered. The instance of the `ValidationStep` object is added to `FileReadMultiStep` as a substep. Finally, the instance of this object is added to an instance of the `TestCase` object (that is, `employeeContractTest`). Now the test setup stage is complete.

To execute the test case a `BizUnit` object is instantiated with reference to the `TestCase` object (that is, `employeeContractTest`). Then the `Run` method is called. This concludes the test execution stage. As the complete test case is coded in a `test` class of a test project in Visual Studio, you can execute the test within Visual Studio and view the result.

There's more...

You will find more background information on BizUnit itself on MSDN and CodePlex:

- **Using BizUnit for Automated Testing**: `http://msdn.microsoft.com/en-us/library/cc594538(v=BTS.10).aspx`
- **BizUnit - Framework for Automated Testing of Distributed Systems**: `http://bizunit.codeplex.com/`

On CodeProject, you will find an article called **BizUnit 4.0 and BizTalk 2010** (`http://www.codeproject.com/KB/biztalk/BizUnit4BTS2010.aspx`). This article explains about BizUnit 4.0 and BizTalk 2010 in more detail.

See also

- Refer to the *Testing BizTalk artifacts inside Visual Studio* recipe discussed earlier in this chapter

Applying code coverage to a BizTalk orchestration

Code coverage (`http://en.wikipedia.org/wiki/Code_coverage`) describes the degree to which a source code of a certain program has been tested. In the BizTalk context this means how much of the orchestration flow has been covered based on the tracked data. The code coverage is measured through a community tool called the **Orchestration Profiler**. This tool queries the BizTalk databases and creates CHM (compiled help) report files illustrating the level of coverage for specified BizTalk orchestrations. In this recipe, we will apply code coverage for an orchestration used in *Chapter 7*, *Monitoring and Maintenance*. You can also apply code coverage to orchestration(s) of your own choice.

Getting ready

Download the Orchestration Profiler v1.2 from CodePlex (`http://biztalkorcprofiler.codeplex.com/releases/view/42777`). Extract the files and run the `.msi` file. Finally, make the following adjustments to the `config` file in folder, where the profiler is installed:

```
<add key="HelpCompilerLocation" value="C:\Program Files (x86)\HTML
Help Workshop\hhc.exe" />
<codeBase version="3.0.1.0" href="C:\Program Files (x86)\Microsoft
BizTalk Server 2010\Tracking\Microsoft.BizTalk.XLangView.dll"/>
```

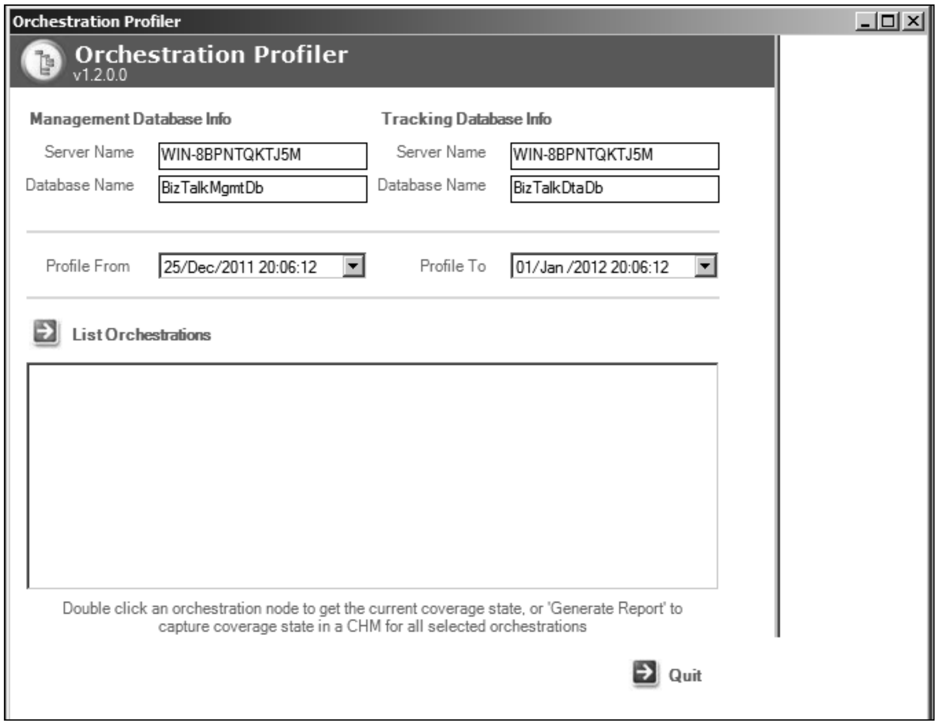
Try adding the following at the end of the `config` file (but before the `</configuration>` tag):

```
<startup useLegacyV2RuntimeActivationPolicy="true"><supportedRuntime
version="v4.0" sku=".NETFramework,Version=v4.0"/></startup>
```

How to do it...

The following steps will show you how to work with the Orchestration Profiler:

1. Open the Orchestration Profiler.
2. Click on **List Orchestrations**:



Orchestration Profiler

Orchestration Profiler
v1.2.0.0

Management Database Info
Server Name: WIN-8BPNTQKTJ5M
Database Name: BizTalkMgmtDb

Tracking Database Info
Server Name: WIN-8BPNTQKTJ5M
Database Name: BizTalkDtaDb

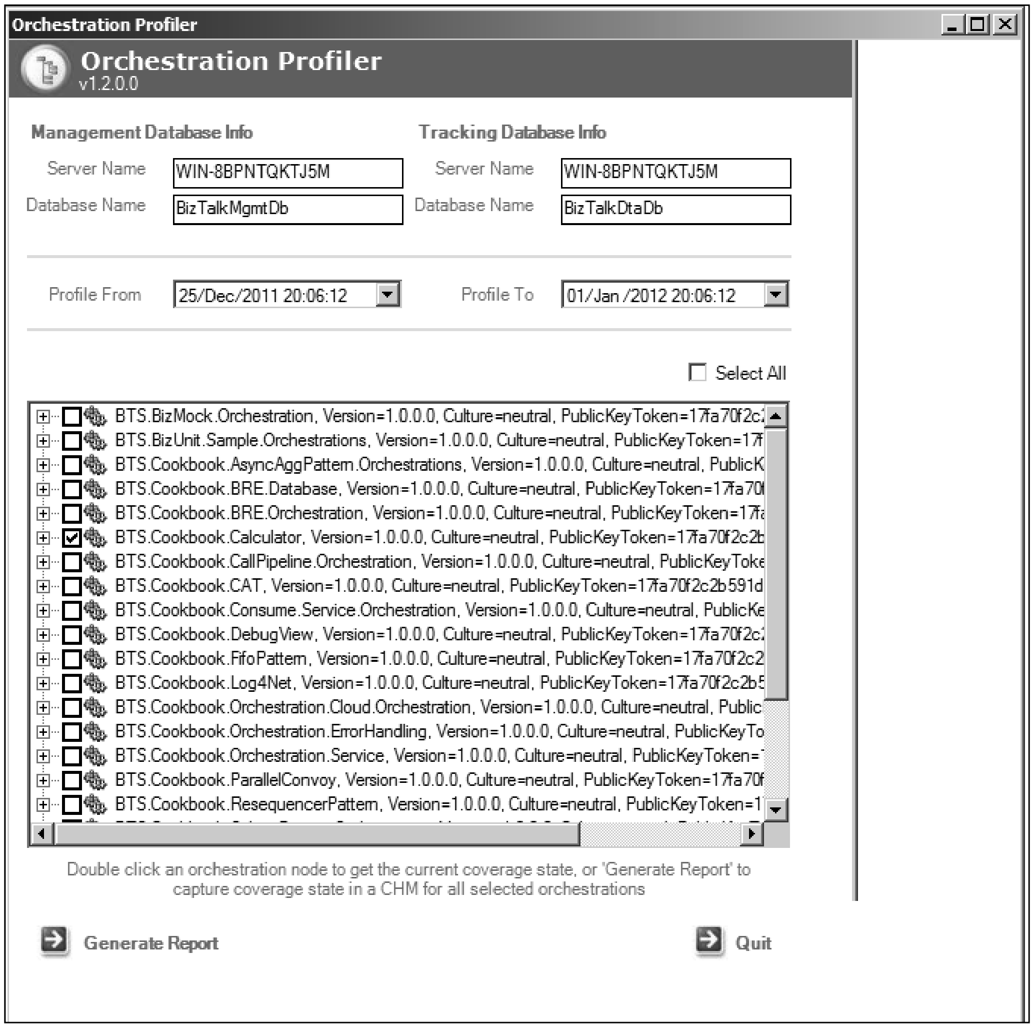
Profile From: 25/Dec/2011 20:06:12
Profile To: 01/Jan /2012 20:06:12

List Orchestrations

Double click an orchestration node to get the current coverage state, or 'Generate Report' to capture coverage state in a CHM for all selected orchestrations

Quit

3. Select the orchestration(s) for which you want to generate a report to obtain a view of the coverage:



Orchestration Profiler

Orchestration Profiler
v1.2.0.0

Management Database Info

Server Name	WIN-8BPNTQKTJ5M
Database Name	BizTalkMgmtDb

Tracking Database Info

Server Name	WIN-8BPNTQKTJ5M
Database Name	BizTalkDtaDb

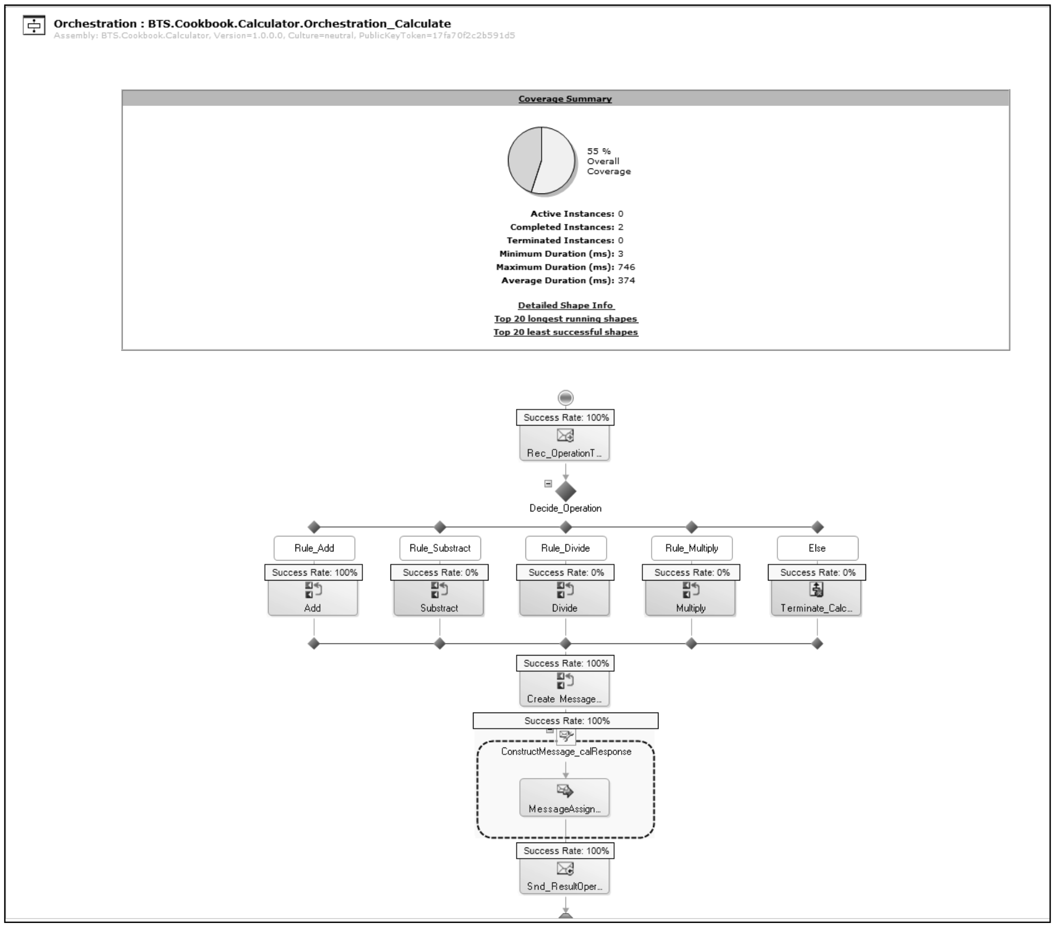
Profile From 25/Dec/2011 20:06:12 Profile To 01/Jan/2012 20:06:12

☐ Select All

- ☐ BTS.BizMock.Orchestration, Version=1.0.0.0, Culture=neutral, PublicKeyToken=17fa70f2c
- ☐ BTS.BizUnit.Sample.Orchestrations, Version=1.0.0.0, Culture=neutral, PublicKeyToken=17f
- ☐ BTS.Cookbook.AsyncAggPattern.Orchestrations, Version=1.0.0.0, Culture=neutral, PublicK
- ☐ BTS.Cookbook.BRE.Database, Version=1.0.0.0, Culture=neutral, PublicKeyToken=17fa70
- ☐ BTS.Cookbook.BRE.Orchestration, Version=1.0.0.0, Culture=neutral, PublicKeyToken=17f
- ☑ BTS.Cookbook.Calculator, Version=1.0.0.0, Culture=neutral, PublicKeyToken=17fa70f2c2b
- ☐ BTS.Cookbook.CallPipeline.Orchestration, Version=1.0.0.0, Culture=neutral, PublicKeyToke
- ☐ BTS.Cookbook.CAT, Version=1.0.0.0, Culture=neutral, PublicKeyToken=17fa70f2c2b591d
- ☐ BTS.Cookbook.Consume.Service.Orchestration, Version=1.0.0.0, Culture=neutral, PublicKe
- ☐ BTS.Cookbook.DebugView, Version=1.0.0.0, Culture=neutral, PublicKeyToken=17fa70f2c
- ☐ BTS.Cookbook.FifoPattern, Version=1.0.0.0, Culture=neutral, PublicKeyToken=17fa70f2c2
- ☐ BTS.Cookbook.Log4Net, Version=1.0.0.0, Culture=neutral, PublicKeyToken=17fa70f2c2b5
- ☐ BTS.Cookbook.Orchestration.Cloud.Orchestration, Version=1.0.0.0, Culture=neutral, Public
- ☐ BTS.Cookbook.Orchestration.ErrorHandling, Version=1.0.0.0, Culture=neutral, PublicKeyTo
- ☐ BTS.Cookbook.Orchestration.Service, Version=1.0.0.0, Culture=neutral, PublicKeyToken=
- ☐ BTS.Cookbook.ParallelConvoy, Version=1.0.0.0, Culture=neutral, PublicKeyToken=17fa70f
- ☐ BTS.Cookbook.ResequencerPattern, Version=1.0.0.0, Culture=neutral, PublicKeyToken=1

Double click an orchestration node to get the current coverage state, or 'Generate Report' to capture coverage state in a CHM for all selected orchestrations

4. When you clicked on **Generate Report**, you have to choose a location for the report and click on **OK**:



Orchestration : BTS.Cookbook.Calculator.Orchestration_Calculate
Assembly: BTS.Cookbook.Calculator, Version=1.0.0.0, Culture=neutral, PublicKeyToken=17fa70f2c2b591d5

Coverage Summary

55 % Overall Coverage

Active Instances: 0
Completed Instances: 2
Terminated Instances: 0
Minimum Duration (ms): 3
Maximum Duration (ms): 746
Average Duration (ms): 374

Detailed Shape Info
Top 20 longest running shapes
Top 20 least successful shapes

5. You can click on one of the following links in the report:

- **Detailed Shape Info**
- **Top 20 longest running shapes**
- **Top 20 least successful shapes**

How it works...

This tool generates a CHM report file illustrating the level of coverage for the specified BizTalk orchestrations. It is important that in the configuration the `HelperCompilerLocation` property is set to the folder containing the `hhc.exe` file.

While running the Orchestration Profiler, you can select one or more orchestrations and then choose **Generate Report**. The profiler will then query the BizTalk databases and execute `hhc.exe`. Subsequently, a consolidated view of the orchestration tracking data for a specified period of time will be produced. A developer will then see how the orchestrations are running and how much coverage they get. In addition to the simple coverage information, the data presented helps to identify latency and code path exceptions by highlighting long running and error prone orchestration shapes, which are key to effective performance testing.

There's more...

The Orchestration Profiler is a pretty straightforward tool you can use to measure code coverage. There is, unfortunately, not much documentation to be found on the Orchestration Profiler. The following are the resources related to this tool:

- **BizTalk Orchestration Profiler v1.2**: `http://santoshbenjamin.wordpress.com/2010/03/29/biztalk-orchestration-profiler-v1-2/`
- **Performance Tools**: `http://msdn.microsoft.com/en-us/library/ee377051%28v=bts.70%29.aspx`
- **BizTalk Server 2006 Orchestration Profiler**: `http://biztalkorcprofiler.codeplex.com/`

See also

- Refer to the *Testing BizTalk artifacts inside Visual Studio* recipe discussed earlier in this chapter

Testing BizTalk solutions with BizMock

BizMock is a framework for testing BizTalk solutions. It has been developed by Pierre Milet Llobet. It is available through CodePlex. BizMock uses a **Domain Driven Design** (**DDD**) approach, fluent interface API, and has mocking capabilities. The latter means you do not need to depend on an infrastructure, such as a database, as you isolate the tests on the developer machine or build server.

The tests you write are executed with Visual Studio using its regular tests and `C#` code. With BizMock, you can quickly test your orchestrations, schemas, maps, and pipelines by writing unit tests that can emulate the messages received at the Receive ports and validate messages sent from the Send ports.

In this recipe, we will follow the typical steps for the BizMock test project:

1. Deploy your BizTalk orchestration, pipeline, or map.
2. Create a test project and add the BizMock assembly references.
3. Add the `artifacts.tt` and `artifacts.xml` file to the test project.
4. Modify `artifacts.xml` declaring your artifacts definitions.
5. Write the test methods that use the generated artifacts and run it.

Getting ready

Download BizMock2010 from CodePlex (`http://bizmock.codeplex.com/`). Extract the files, and then go to `.\src\BizMockSetup\Debug` and execute `BizMockSetup.msi`. Finally, register the BizMock adapter from the BizTalk Administration Console. For a reference project (`BTS.BizMock.Orchestration`), you can download the source code belonging to this chapter.

How to do it...

The following steps describe how to test an orchestration using BizMock:

1. Build and deploy the solution containing the orchestration, pipeline, map, and/or other artifacts (with the sample belonging to the book you can deploy `BTS.BizMock.Orchestration`).
2. Open Visual Studio 2010.
3. In the **New Project** dialog box, from the list of installed templates, click on **Visual C#**. In the right pane, select **Test** and choose **Test Project**. For the project name, enter an appropriate name and then click on **OK**.
4. Add the following references to the project:
   - `Microsoft.Biztalk.Adapter.Framework.dll`
   - `Microsoft.BizTalk.ApplicationDeployment.Engine.dll`
   - `Microsoft.BizTalk.ExplorerOM.dll`
   - `Microsoft.BizTalk.GlobalPropertySchemas.dll`
   - `Microsoft.BizTalk.Interop.TransportProxy.dll`
   - `Microsoft.BizTalk.Operations.dll`

- Microsoft.BizTalk.Pipeline.dll
- Microsoft.BizTalk.Streaming.dll
- Microsoft.Samples.BizTalk.Adapter.Common.dll
- Microsoft.XLANGs.BaseTypes.dll
- BizMockAdapterManagement.dll
- BizMockArtifactsSchema.dll
- BizMockery.dll
- BizMockMessaging.dll
- BizMockReceiveAdapter.dll
- BizMockTransmitAdapter.dll
- System.Xml.dll
- Reference to your orchestration (that is, `BTS.BizMock.Orchestration`)

5. Rename `UnitTest1.cs` to `ArtifactsTests.cs`.
6. This class needs to contain the following assemblies:

```
using BizMock;
using BizMock.Tools;
using System.Xml.Serialization;
using BTS.BizMock.Orchestration;
```

7. Copy the `Artifacts.tt` and `Artifacts.xml` files from `<install directory>\Install older \BizMock 2010\BizMock 2010\src\BizMockTest\Artifacts.tt` to your project.
8. Add the `Artifacts.tt` and `Artifacts.xml` files to your project.
9. Modify both files to your requirements (for example, artifacts), see the sample part of the artefact definition in the following code snippet:

```
<MessageInstanceArtifacts>
  <MessageInstance>
    <Name>Msg_Simple</Name>
    <Type>SimpleMessageType</Type>
    <Files>
      <File>C:\\BizTalk Server 2010 Cookbook\\Chapter
      9\\BTS.BizMock.Test\\BTS.BizMock.Test\\MessageInstances
      \\simpleschema_input.xml</File>
    </Files>
  </MessageInstance>
  <MessageInstance>
    <Name>Msg_Simple2</Name>
    <Type>SimpleMessageType2</Type>
```

```
          <Files>
            <File>C:\\BizTalk Server 2010 Cookbook\Chapter
            9\BTS.BizMock.Test\\BTS.BizMock.Test\\MessageInstances\\
            simpleschema2_input.xml</File>
          </Files>
        </MessageInstance>
        <MessageInstance>
          <Name>Msg_MultiPart</Name>
          <Type>MultiPartMessageType</Type>
          <Files>
            <File>C:\\BizTalk Server 2010 Cookbook\\Chapter
            9\BTS.BizMock.Test\\BTS.BizMock.Test\MessageInstances\\
            simpleschema_input.xml</File>
            <File>C:\\BizTalk Server 2010 Cookbook\\Chapter
            9\BTS.BizMock.Test\\BTS.BizMock.Test\MessageInstances\\
            simpleschema2_input.xml</File>
          </Files>
        </MessageInstance>
      </MessageInstanceArtifacts>
      <MessageVerifierArtifacts>
```

10. In `ArtifactsTests.cs` you can create the test methods with artifacts and use the code shown, as follows (that is, you will find the complete code with the source code):

```
[TestMethod]
  [DeploymentItem("MessageInstances")]
  //testing an orchestration that receives a multipart message and
  return 2 multipart messages
  public void MultipartWithLoopDirectTest()
    {
      Expect.TIMEOUT = 6000;
      Expect.DELAY = 5000;
      artifacts.MultipartWithLoopOrchestration.Start();
        artifacts.Msg_MultiPart.InterchangeID = "X";
        artifacts.Vrf_Multipart.MultiPart1.Field = "Y";
        artifacts.Vrf_Multipart.MultiPart2.Field = "Y";
      Submit.Request(artifacts.Msg_MultiPart).
      To(artifacts.OneWayDirect);
      Expect.AtLeast(2).Request.At(artifacts.OWPort_2).
      Verify(artifacts.Vrf_Multipart);
    }
```

11. Create a new `test` class and name it as `BizMockeryTests.cs`.
12. This class can contain one or more test methods to all kinds of test regarding the artifacts you are using for your mock test.

13. With the samples provided with BizTalk, you can pick any tests you want, or create them by yourself as the following code:

```
[TestMethod]
  [DeploymentItem("MessageInstances")]
  public void VerifyFileInstance()
  {
    //Change directory to your requirement
    BizMockMessage msg = new BizMockMessage(@"C:\BizTalk Server
    2010 Cookbook\Chapter 9\BTS.BizMock.Test\BTS.BizMock.Test\
    MessageInstances\simpleschema_input.xml");
    XmlMessageVerifier msgVrf = new XmlMessageVerifier();
    msgVrf.ExpectedValue("/*[local-name()='Root' and namespace-
    uri()='http://BTS.BizMock.Orchestration.SimpleSchema']/
    *[local-name()='Field' and namespace-uri()='']", "X");
    Expect.Message(msg).Verify(msgVrf);
  }
```

14. Once you are done with building your test methods, you can build and run the tests.

How it works...

As a developer you do not have to configure the BizTalk solution you build. You can deploy the solution and then setup a test project. To the test project, a `T4-template` (`http://msdn.microsoft.com/en-us/library/bb126445.aspx`) file (`artifcats.tt`) is added which is used to convert the XML file to a class that can be used in the tests. The XML file contains references to all artifacts used in the test, such as messages, ports, maps, and so on. The artifacts are required to be able to run a test that is defined in a class module within a `test` method. In this recipe, a test is run for an orchestration using the artifacts orchestration object start method:

```
artifacts.MultipartWithLoopOrchestration.Start();
```

A multipart message is constructed with the artifacts `MsgMultipart` object assigning values to the `InterchangeID`, and `Multipart Field` properties. Then, this is submitted to BizTalk. The `Expect` object method `RequestAtInfo` will be called to verify the result.

Another class is created in this recipe that contains the test methods for the messages themselves, such as `verify` if the message exists and if the accepted results are in the message itself. An instance of the `BizMockMessage` object is created with a reference to the location of the input message and then an instance of the `XMLMessageVerifier` object is created. The `ExpectedValue` method of the `XMLMessageVerifier` object is called with an XPath query to the field element in the message with the expected value as the parameters.

When running the test, the artifacts such as ports are created, enlisted, and started. The same counts for the orchestration. A message is picked up through a Receive port configured with the BizMock adapter. Furthermore, the location of the input message and its expected values are verified.

There's more...

There is little documentation on BizMock itself, other than what is offered through CodePlex. For more details refer to the **BizMock** documentation at `http://bizmock.codeplex.com/documentation`.

There are a few blog posts on using the BizMock, where people share their experiences:

- **BizMock using it for orchestration unit testing**: `http://snefs.blogspot.com/2011/08/bizmock-using-it-for-orchestration-unit.html`
- **Introduction to BizMock**: `http://talentedmonkeys.wordpress.com/2011/08/29/introduction-to-bizmock/ (with reference to PowerPoint presentation)`

See also

- Refer to the *Testing BizTalk artifacts inside Visual Studio* and *Unit Testing a BizTalk solution with BizUnit* recipes discussed earlier in this chapter

Using the BizTalk Map Test Framework

The BizTalk Map Test Framework has been developed by Maurice den Heijer. It enables developers to perform tests on their maps using template files and XPath queries. The framework offers a great deal of extensibility when using it within a Visual Studio Test Project.

A developer can define a large number of test cases to be applied upon the map and these test cases are run within a single method call. Hence, the productivity of testing a map is enhanced significantly. Also, the developer only needs to maintain two XML files (one source XML (that is, the input file) and a result XML (that is, the output file)). Both files are basically instances of a source schema and a destination schema of the map to be tested.

In this recipe, a BizTalk map will be tested through using the BizTalk Map Test Framework in the Visual Studio Test Project (that is, unit testing).

Getting ready

Download the BizTalk Map Test Framework 1.0 from CodePlex (`http://mtf.codeplex.com/releases/view/58330`). Unzip it and place the content in a folder. In the folder, you will have three assemblies—`MapTestFramework.Common.dll`, `nunit.framework.dll`, and `xmlunit.dll`. To use this recipe, you can use your own mapping or one provided through the code accompanying this book.

How to do it...

The following steps describe the process to test a BizTalk map using the BizTalk Map Test Framework:

1. Open your BizTalk solution that contains the map(s) you want to test.
2. Right-click on the BizTalk project containing the maps and click on **Properties**.
3. Select the **Deployment** tab and set the **Enable Unit Testing** property to **True**.
4. Build/rebuild your project.
5. Right-click on your solution, select **Add**, and choose **New Project...**.
6. Select the **Test Projects** templates.
7. Expand the node and choose **Test Document**. Click on the **Test Project** template.
8. Give it an appropriate name and click on **OK**.
9. Add a reference to `MapTestFramework.Common.dll`, `Microsoft.BizTalk.TestTools.dll`, `Microsoft.XLANGs.BaseTypes.dll`, and the assembly that contains the map(s) you want to test.
10. Inherit from the `MappingFixture` class:

    ```
    public class UnitTests : MappingFixture
    ```

11. Implement the `CreateMap` method and return an instance of the `TestableMapBaseMapExecuter` class. Pass an instance of the map as a parameter to the constructor of the class:

    ```
    protected override IMapExecuter CreateMap()
    {
      return new TestableMapBaseMapExecuter(new
      EmployeeToEmployeeContract());
    }
    ```

12. The next step is to add the `SourceBasePath` and the `ExpectedBasePath` properties and return your template files:

    ```
    protected override string ExpectedPathBase
    {
      get { return @"C:\BizTalk Server 2010 Cookbook\Chapter
      9\BTS.Cookbook.Map.Test.Framework\TestInstances\Output.xml";
      }
    }
    protected override string SourcePathBase
    {
      get { return @"C:\BizTalk Server 2010 Cookbook\Chapter
      9\BTS.Cookbook.Map.Test.Framework\TestInstances\
      TestInstance.xml"; }
    }
    ```

13. Now implement the `base` test:

    ```
    [TestMethod]
      public void TestBase()
      {
        base.ExecuteBaseTest();
        return;
      }
    ```

14. Determine which elements participate in your test and instantiate your `MapTestCases` (this depends, of course, on your schemas, the following code illustrates the XPath queries belonging to the code sample):

    ```
      MapTestCases collection = new MapTestCases(
        new string[] {
          "/*[local-name()='Employee' and namespace-
          uri()='http://BTS.Cookbook.Map.Test.Framework.
          Schemas.Employee']/*[local-name()='Salary' and namespace-
          uri()='']",
          "/*[local-name()='Employee' and namespace-
          uri()='http://BTS.Cookbook.Map.Test.Framework.Schemas.
          Employee']/*[local-name()='Bonus' and namespace-uri()='']"
        },
        new string[] {
          "/*[local-name()='Employee' and namespace-
          uri()='http://BTS.Cookbook.Map.Test.Framework.Schemas.
          EmployeeContract']/*[local-name()='Contract' and namespace-
          uri()='']/*[local-name()='Salary' and namespace-uri()='']"
        }
      );
    ```

15. Add the test cases by specifying values for the source and the expected files based on the XPath queries of the previous step:

    ```
      collection.AddTestCase(
        new string[] { "10000","1000" },
        new string[] { "$11000" }
      );
      base.ExecuteMapTest(collection);
    return;
    ```

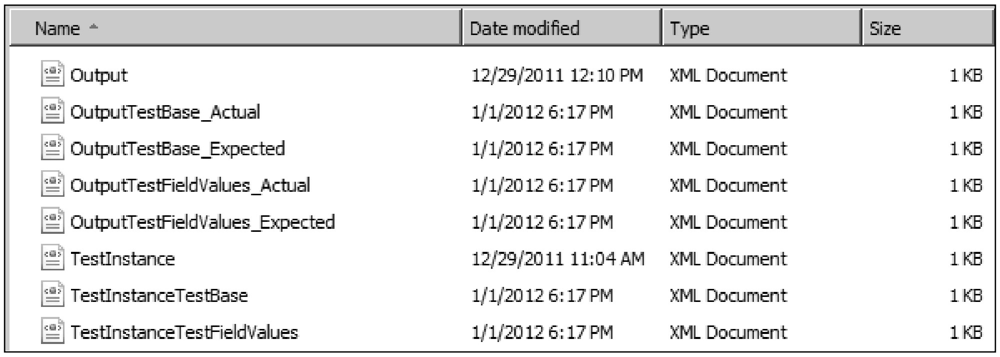
16. You can test the methods by clicking on **Run all Tests in Solution**.

How it works...

To test a map leveraging the BizTalk Test Map Framework, you will need to enable unit testing in the BizTalk project that contains the map(s). Similar to BizUnit, you setup a test case(s) and then execute the test case(s).

Setup involves building the project with the map(s). The created assembly is referenced in the test project, together with references to `Microsoft.BizTalk.TestTools.dll`, `MapTestFramework.Common.dll`, and `Microsoft.XLANGs.BaseTypes.dll`. In the test project you set up the test by implementing a class that inherits from the `MappingFixture` class. In this class, a `CreateMap` method is implemented that returns a `TestableMapBaseMapExecuter` instance with reference to a map (that is, a map called `EmployeeToEmployeeContract`). The next two properties, `SourceBasePath` and `ExpectedBasePath` are set with locations to the path where the input message (that is, the instance of the source schema) and output message (that is, instance of destination schema) reside. Then the `TestBase` method is created that will test whether the source schema instance correctly maps into the expected destination schema instance. Finally, you determine which elements in the source schema instance participate in the test cases. A `MapTestCases` object is instantiated with XPath queries to elements in source schema which will be involved in the test cases. Subsequently, a collection of test cases is built and executed. Each test case uses separate files which are generated by the `ExecuteMapTest` method.

When the test is run, you will see the files, as depicted in the following screenshot, and you can compare the actual and expected files, and investigate the map to determine if the test has failed or succeeded:



Name	Date modified	Type	Size
Output	12/29/2011 12:10 PM	XML Document	1 KB
OutputTestBase_Actual	1/1/2012 6:17 PM	XML Document	1 KB
OutputTestBase_Expected	1/1/2012 6:17 PM	XML Document	1 KB
OutputTestFieldValues_Actual	1/1/2012 6:17 PM	XML Document	1 KB
OutputTestFieldValues_Expected	1/1/2012 6:17 PM	XML Document	1 KB
TestInstance	12/29/2011 11:04 AM	XML Document	1 KB
TestInstanceTestBase	1/1/2012 6:17 PM	XML Document	1 KB
TestInstanceTestFieldValues	1/1/2012 6:17 PM	XML Document	1 KB

There's more...

You can find more information on the BizTalk Map Test Framework at CodePlex in the document called the **BizTalk Map** Test Framework (`http://mtf.codeplex.com/`).

See also

- Refer to the *Testing BizTalk artifacts inside Visual Studio* and *Unit testing a BizTalk solution with BizUnit* recipes discussed earlier in this chapter

Testing pipelines and pipeline components

The testing of pipelines and pipeline components can be done through the Visual Studio unit tests. Similar to the unit testing of the schemas and maps described in the first recipe. The unit testing of the pipelines within Visual Studio is similar to the `Pipeline.exe` tool (for more information on the `Pipeline.exe` tool, see `http://msdn.microsoft.com/en-us/library/aa547988.aspx`.

Besides unit testing of pipelines through Visual Studio, you can also opt for testing a pipeline and its components using the Pipeline Testing Library created by Tomas Restrepo. This library allows developers to programmatically create pipelines, create instances of components, and assign them to their respective stages (including their own custom components and the familiar out of the box components), or load an existing BizTalk pipeline without the need to deploy the supporting BizTalk project.

In this recipe, you will see two possible ways to test pipeline components. One is using Visual Studio's capability of running tests and the other is using a test library in a Visual Studio unit test.

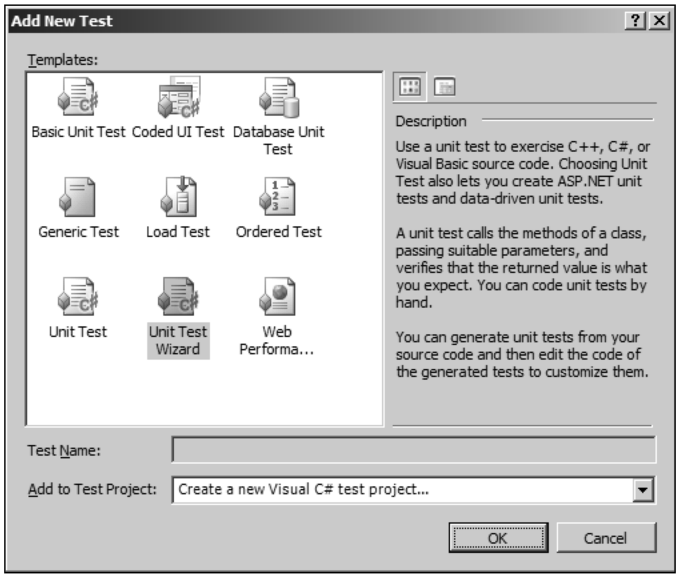
Getting ready

To use this recipe, you can use your pipeline component(s) or use the one provided through the code accompanied by this book. The code belonging to `BTS.Cookbook.Pipeline.Test` solution also has a reference to the Pipeline Testing Library, which you need to download (the library can be downloaded from this location: `http://winterdom.com/2007/08/pipelinetesting11released`) and build on your own machine.

How to do it...

The following steps involve setting up a test, to test a custom pipeline through Visual Studio 2010:

1. In Visual Studio 2010, open the solution containing the custom pipeline(s).
2. In **Solution Explorer**, right-click on the project and then click on **Properties**.
3. In **Project Designer**, click on the **Deployment property page** tab and set **Enable Unit Testing** to **True**.
4. Close the **Project Properties** page after saving the changes.
5. On the main menu, click on **Build** and then click on **Rebuild Solution**.
6. On the main menu, click on **Test** and then click on **New Test**.
7. In the **Add New Test** dialog box, select **Create a new Visual C# test project** for the **Add to Test Project** field. Select **Unit Test Wizard** in the **Templates** list, and then click on **OK**:


Add New Test

Templates:

Basic Unit Test, Coded UI Test, Database Unit Test, Generic Test, Load Test, Ordered Test, Unit Test, Unit Test Wizard, Web Performa...

Description

Use a unit test to exercise C++, C#, or Visual Basic source code. Choosing Unit Test also lets you create ASP.NET unit tests and data-driven unit tests.

A unit test calls the methods of a class, passing suitable parameters, and verifies that the returned value is what you expect. You can code unit tests by hand.

You can generate unit tests from your source code and then edit the code of the generated tests to customize them.

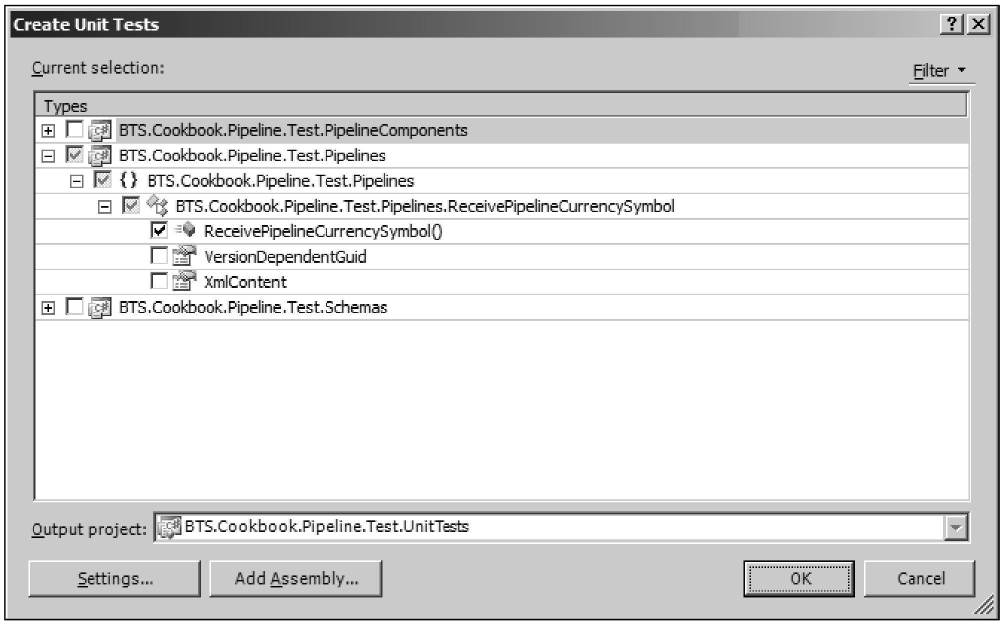
Test Name:

Add to Test Project: Create a new Visual C# test project...

OK Cancel

8. In the **New Test Project** dialog box, give the project an appropriate name and click on **Create**.

9. In the **Create Unit Tests** dialog box, expand **Types** and select the pipelines and/or components' node(s) and select the constructor(s). For the sample provided by this recipe, the `ReceivePipelineCurrencySymbol` constructor is selected:


Create Unit Tests

Current selection:

Filter

Types

- [ ] BTS.Cookbook.Pipeline.Test.PipelineComponents
- [✓] BTS.Cookbook.Pipeline.Test.Pipelines
  - [✓] { } BTS.Cookbook.Pipeline.Test.Pipelines
    - [✓] BTS.Cookbook.Pipeline.Test.Pipelines.ReceivePipelineCurrencySymbol
      - [✓] ReceivePipelineCurrencySymbol()
      - [ ] VersionDependentGuid
      - [ ] XmlContent
- [ ] BTS.Cookbook.Pipeline.Test.Schemas

Output project: BTS.Cookbook.Pipeline.Test.UnitTests

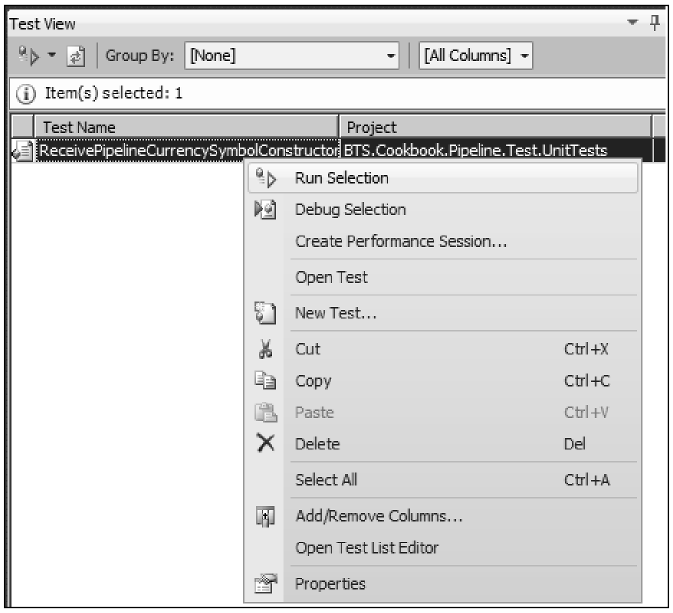
Settings... Add Assembly... OK Cancel

10. Click on **OK**.
11. A class file will open and you can now add the following `using` statements:
    - `using System.IO`
    - `using System.Collection.Specialized`
    - `using System.Collection.Generic`
12. Scroll to the bottom of the file and replace your `ConstructorTest` method with the following code which verifies that the pipeline inputs exist before testing the pipeline. This code also verifies that a message conforming to the **Employee** schema is generated (the code reflects the sample belonging to this recipe):

```
ReceivePipelineCurrencySymbol target = new
ReceivePipelineCurrencySymbol();
  //Collection of messages to test the pipeline
  StringCollection documents = new StringCollection();
    string strSourceEmployee_XML = @"C:\BizTalk Server 2010
    Cookbook\Chapter 9\BTS.Cookbook.Pipeline.Test\TestInstance\
    EmployeeInstance.xml";
      Assert.IsTrue(File.Exists(strSourceEmployee_XML));
      documents.Add(strSourceEmployee_XML);
      // Only a body part for this test message so an empty
      // collection will be passed.
      StringCollection parts = new StringCollection();
      // Dictionary mapping the schema to the namespace and type
      // as displayed in the properties window for the *.xsd
      Dictionary<string, string> schemas = new Dictionary<string,
      string>();
      string SchemaFile = @"C:\BizTalk Server 2010 Cookbook\
      Chapter 9\BTS.Cookbook.Pipeline.Test\BTS.Cookbook.Pipeline.
      Test.Schemas\Employee.xsd";
      Assert.IsTrue(File.Exists(SchemaFile));
      schemas.Add("BTS.Cookbook.Pipeline.Test.Schemas.Employee",
      SchemaFile);
      // Test the execution of the pipeline using the inputs
      target.TestPipeline(documents, parts, schemas);
      // Validate that the pipeline test produced the message
      // which conforms to the schema.
      string[] strMessages = Directory.
      GetFiles(testContextInstance.TestDir +
      "\\out", "Message*.out");
      Assert.IsTrue(strMessages.Length > 0);
      Schemas.Employee employee = new Schemas.Employee();
      foreach (string outFile in strMessages)
```

```
            {
              Assert.IsTrue(employee.ValidateInstance(outFile,
              Microsoft.BizTalk.TestTools.Schema.
              OutputInstanceType.XML));
            }
```

13. In **Solution Explorer**, right-click on **Test Project** and then click on **Build**.
14. On the main menu, click on **Test** and then in the **Windows** list, click on **Test View**.
15. In the **Test View** window, right-click on your pipeline unit test, and then click on **Run Selection**. Verify that you see **Passed** in the **Test Results** window:



Item(s) selected: 1

Test Name	Project
ReceivePipelineCurrencySymbolConstructor	BTS.Cookbook.Pipeline.Test.UnitTests

16. If any test fails, you can double-click on the test in the **Test Results** window to see the assert or exception that caused the test failure.

The following steps involve setting up a test for testing a custom pipeline using the Test Library:

1. Right-click on your solution and select **Add**, choose **New Project...**.
2. Select the **Test Projects** templates.
3. Expand **Node** and choose **Test Document**. Click on the **Test Project** template.
4. Give it an appropriate name and click on **OK**.

5. Add a reference to the following assemblies:
    - `Winterdom.BizTalk.PipelineTesting.dll`
    - `Microsoft.BizTalk.Pipeline.dll`
    - `Microsoft.BizTalk.TestTools.dll`
    - `Microsoft.XLANGs.BaseTypes.dll`
    - `Assembly containing your pipeline component(s)`
    - `Assembly containing your pipeline(s)`
6. Create a `test` method in the **Unit Test Class**.
7. Add the following code in the method:

```
ReceivePipelineWrapper pipeline = PipelineFactory.CreateReceivePip
eline(typeof(ReceivePipelineCurrencySymbol));
  //// Create the input message to pass through the pipeline
  XmlDocument xmlDoc = new XmlDocument();
  ////Change to directory containing the file you use as resource
  xmlDoc.Load(@"C:\BizTalk Server 2010 Cookbook\Chapter
  9\BTS.Cookbook.Pipeline.Test\BTS.Cookbook.Pipeline.Test.
  TestLibrary\EmployeeInstance.xml");
  // Encode the XML string in a UTF-8 byte array
  byte[] encodedString = Encoding.UTF8.GetBytes(xmlDoc.InnerXml);
  // Put the byte array into a stream and rewind it to the
  beginning
  MemoryStream ms = new MemoryStream(encodedString);
    ms.Flush();
    ms.Position = 0;
    Stream stream = ms;
  IBaseMessage inputMessage =
  MessageHelper.CreateFromStream(stream);
  //// Add the necessary schemas to the pipeline, so that
  //// disassembling works
  pipeline.AddDocSpec(typeof(Schemas.Employee));
  //// Execute the pipeline, and check the output
  MessageCollection outputMessages =
  pipeline.Execute(inputMessage);
  //// Assert if outputMessage exists
  Assert.IsNotNull(outputMessages);
  Assert.IsTrue(outputMessages.Count > 0);
  //// Create output message
  if (outputMessages.Count > 0)
  {
    Stream outStream =
    outputMessages[0].BodyPart.GetOriginalDataStream();
```

```
        xmlDoc.Load(outStream);
        xmlDoc.Save(@"C:\BizTalk Server 2010 Cookbook\Chapter
        9\BTS.Cookbook.Pipeline.Test\BTS.Cookbook.Pipeline.
        Test.TestLibrary\OutEmployeeInstance.xml");
    }
```

8. In **Solution Explorer**, right-click on **Test Project** and then click on **Build**.
9. On the main menu, click on **Test**, and then in the **Windows** list, click on **Test View**.
10. In the **Test View** window, right-click on your pipeline unit test, and then click on **Run Selection**. Verify that you see **Passed** in the **Test Results** window.

How it works...

This recipe showed two possible ways to test pipelines and pipeline components by either unit testing them with or without using the Pipeline Testing Library. In case you do not use the Test Library you can enable unit testing in the BizTalk project containing the pipeline(s). The `pipeline` class in your project is derived from the `Microsoft.BizTalk.TestTools.Pipeline.TestableReceivePipeline` class, which models some of the same functionality exposed by the `Pipeline.exe` tool. With the Unit Test Wizard, you can implement the code to the constructor of this class. Within the constructor you can implement the code to set up a test with the pipeline by assigning one or more test documents (path with a file name) to a string collection and a dictionary mapping to the schema and its namespace and type. Next, the `TestPipeline` method with parameters, string collection of test documents, and the dictionary is created. Finally, within the constructor you can put asserts to the test to check if, for instance, there are any output files created.

With the test library you can basically do the same thing as with the Visual Studio unit test. First, an instance to a pipeline that has been referenced in the test project is created. Next, you configure the pipeline by providing a message by loading a test document and a document specification (that is, the schema that the message needs to adhere to). Finally, you assign the result of the execution of the pipeline to a `MessageCollection` object. You can, again, put asserts to the test to check if, for instance, there is any output and stream output of the file.

There's more...

On MSDN, you can find more on unit testing:

- **Unit Testing with BizTalk Server Projects**: `http://msdn.microsoft.com/en-us/library/dd257907.aspx`
- **Using the Unit Testing Feature with Pipelines**: `http://msdn.microsoft.com/en-us/library/dd792682.aspx`

Besides the MSDN resource, BizTalk MVP Michael Stephenson has written an excellent series of blog posts on BizTalk Testing. One of his posts, **BizTalk Testing Series - Testing Pipeline Components** (`http://geekswithblogs.net/michaelstephenson/archive/2008/03/30/120852.aspx`), details the possibilities with testing pipeline components:

You can find information about the Pipeline Testing Library through the following posts:

- **PipelineTesting 1.1 Released**: `http://winterdom.com/2007/08/pipelinetesting11released`
- **Testing Pipeline Components**: `http://winterdom.com/2006/04/testingpipelinecomponents`

See also

- Refer to the *Testing BizTalk artifacts inside Visual Studio* and *Unit testing a BizTalk solution with BizUnit* recipes discussed earlier in this chapter