

Building a SAS Based Web Service

Jonathan Williamson, Amadeus Software Limited



ABSTRACT

Many IT observers are heralding Web Services as the next great thing. But what are Web Services? And how can we build SAS based Web Services?

This paper is intended for developers new to Web Services. Some knowledge of Visual Basic will be advantageous.

In this paper, we will introduce Web Services, what they are and why we might use them. Using a simple scenario, we will then run through the process of building a basic web service using the Microsoft® .NET architecture and SAS software™. Finally, we will use this to show how different types of client system can access web services.

WHAT ARE WEB SERVICES?

Web services are a generalised model for building distributed applications. They can be implemented on any operating system that supports communication over the Internet, allowing us to integrate and access the many different incompatible object models that have been created over time. These object models, like Distributed Component Model (DCOM), Remote Method Invocation (RMI) and Internet Inter-Orb Protocol (IIOP) are very useful, but unfortunately, they depend on object model specific communication protocols. Web Services extend these models to communicate using the Simple Object Access Protocol (SOAP) and eXtensible Markup Language (XML).



Web Services use SOAP to encode method calls in XML, enabling methods to be called on remote systems without regard for the underlying technology that is implementing the method. In turn these SOAP messages can be transmitted between the client and server using various standard communications systems, the primary one – and the one we will focus on – being HTTP

Web Services allow us to call remote methods, but how does the client system know what methods a web service provides, what parameters the methods require, or what data they return? The answer lies in using a common standard to describe the methods a web service exposes. Various description languages have been created for this, however the Web Services Description Language (WSDL) is the standard promoted by the World Wide Web Consortium (W3C) and backed by IBM®, SUN Microsystems® and Microsoft®. WSDL defines the XML structure that web services can use to describe the methods they provide and how those methods can be called.

BUILDING A WEB SERVICE

To demonstrate building a simple web service we will use a simple business scenario. This is a trivial example since we just want to show the concepts involved in writing such a system.

THE SCENARIO

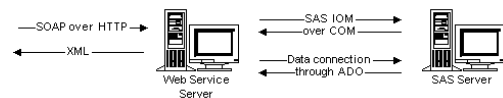
A superstore wants to ensure its suppliers will always be able to meet its demand for their products. To aid them in planning

it wishes to allow each supplier to retrieve up to date stock information.

To solve this problem the superstore wishes to build a web service that allows each manufacturer to enter their manufacturer code and retrieve the latest stock figures directly from the superstores stock control system.

SOLUTION ARCHITECTURE

The Web Service is implemented using the Microsoft .NET architecture. This makes implementing a web service very simple and allows the calling of SAS software™ through the Integrated Object Model (IOM) COM component. IOM allows the Web Service to initiate and control a SAS session, with data being transferred via Active Data Objects (ADO.NET).



The web service controls the overall process in order to satisfy the client request.

It must:

1. Accept the manufacturers code
2. Pass this to a stored SAS procedure that will generate the result data.
3. Retrieve the result data from SAS
4. Return the data to the client as XML.

CREATING THE WEB SERVICE

You create a .NET Web Service using an .asmx file. This file contains the **WebService** directive, which identifies the public class that implements the XML Web service logic. Here is the WebService directive for our example:

```
<%@ WebService Language="vb"
    Codebehind="Inventory.asmx.vb"
    Class="SasStockController.SimpleInventory"
%>
```

This declares that we are writing a web service, we are writing it in Visual Basic®, the class containing the web service is called SasStockController.SimpleInventory and it can be found in the file Inventory.asmx.vb. Since we are using the .NET environment, we could write the class in any language supported by the .NET framework.

In .NET all Web Services must inherit from the WebService class and must contain public methods that correspond to each method to be exposed by the service. Methods that are exposed by the web service are identified using the **WebMethod** attribute.

In our example, we will create a single web method called GetStockLevels. This will take the supplier reference parameter and return an ADO dataset (in XML form) containing the stock information for all of the suppliers products. Here is the framework for the web service.



```
Public Class SimpleInventory
    Inherits System.Web.Services.WebService

    <WebMethod()> _
    Public Function GetStockLevels( _
        ByVal supplierRef As Integer) _
        As DataSet
        ' Function Implementation
    End Function

End Class
```

Now we need to write the code to implement GetStockLevels. This needs to:

- Validate Call / Parameters
- Connect to SAS IOM
- Call the SAS Stored Procedure
- Retrieve the result dataset
- Clear up SAS Objects

In this simple example we don't need to perform any validation on the parameter – .NET will ensure the parameter is numeric, which is enough for our code to work. In a real application we may want to perform range checking, or add security to ensure only authorized people can access the stock information.

We connect to SAS through IOM by creating a workspace manager object and calling one of the CreateWorkspace functions.

```
Dim obWorkspaceManager As New _
    SASWorkspaceManager.WorkspaceManager()
Dim xmlInfo As String

obSAS = _
    obWorkspaceManager.Workspaces.
    CreateWorkspaceByServer("", _
        SASWorkspaceManager.Visibility.
        VisibilityProcess, Nothing, "", _
        "", xmlInfo)
```

In order to call the stored process we have to point the stored process service to our code, create a string of name, value pairs for the parameters and finally call the stored procedure.

```
obSAS.LanguageService.
StoredProcessService.Repository = _
    "file:c:\SASRepository"

Dim params As String
params = "company=" & supplierRef & _
    " outdata=work.output"

obSAS.LanguageService.
StoredProcessService.Execute( _
    "GetStockLevels", params)
```

The stored process will generate the results in the dataset specified by the outdata parameter. So next, we use ADO.NET to pickup the dataset and return it to the client. We do this using the OleDb Data Adapter and linking it to our SAS session to gain access to the work library.

```
Dim obDataset As New System.Data.DataSet()
Dim obAdapter As New
    System.Data.OleDb.OleDbDataAdapter( _
        "select * from work.output", _
        "provider=sas.iomprovider.1; SAS
        Workspace ID=" & _
        obSAS.UniqueIdentifier)
obAdapter.Fill(obDataset, "stocklevels")

GetStockLevels = obDataset
```

Finally, we clear up the objects used. .NET garbage collection should tidy up all objects itself once they go out of scope, however when dealing with data connections any close method should be called to release resources as soon as possible. Since data adapters in Visual Basic do not have a close method, we set it to nothing to show we have finished with it.

We also MUST remove the SAS session from the workspace manager to keep everything tidy. Not doing so will prevent the session being closed. It is important to ensure the SAS session is removed no matter what happens; in the full code in appendix 2 we include a try catch block around the call to SAS so we still remove the session correctly in the event of an exception.

WRITING THE SAS CODE

Now we have our web service that will call SAS we need to write the stored process that will perform the data manipulation.

The stored process service passes the parameters by replacing the comment *ProcessBody; with macro variable definitions for each parameter. Therefore, for our example, all we need is a simple SAS data step that will extract the data for the supplier by using the parameter that is available as a macro variable. We add a keep statement to limit the columns returned.

```
libname stocksys 'c:\SASRepository\Data';

data &outdata;
    set stocksys.levels(
        where=(compcode=&company)
        keep=compcode itemcode itemname
        stocklvl onorder);
run;
```

In the code in Appendix 3, you will notice we set the macro variables prior to the ProcessBody comment. This makes it easier to test the code.

TESTING THE WEB SERVICE

Now we have all our code written we can test the web service. .NET makes it very easy to test web services using a standard browser, we just need to navigate to the asmx file. This generates a simple page describing the web service and all the methods it exposes.

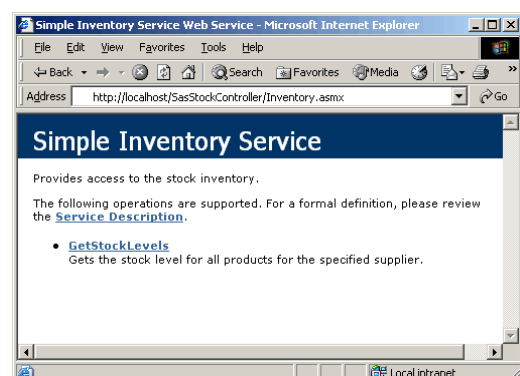


Figure 1 - Web Service HTML Information Page

The method name is a hyperlink to the method call. This gives further information about the method and generates a form that allows us to enter the parameters and call the method.

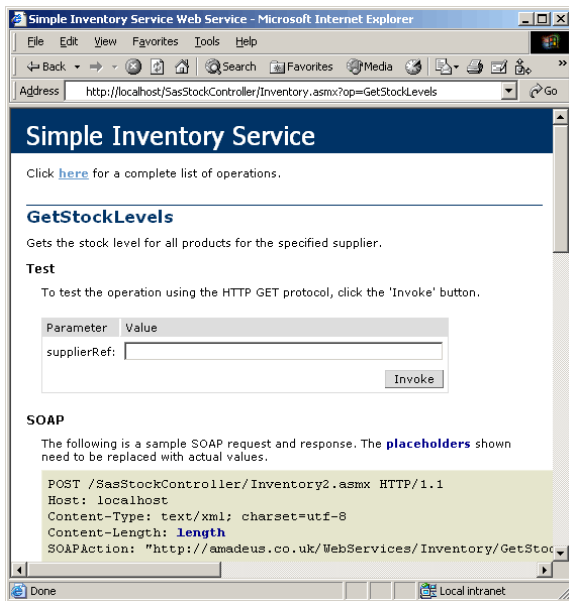


Figure 2 - Web Method HTML Page

Clicking the invoke button calls the method and displays the XML response. Appendix 1 gives an example response. As you can see there is a lot of xml for a relatively small amount of data. This is one of the main drawbacks of using a generic text based mechanism like XML – there can be a lot of overhead in terms of metadata describing what a message contains.

BUILDING WEB SERVICE CLIENTS

Now we have a web service we need to be able to write applications that can call it.

As you have just seen, calling the web service from a browser is extremely simple, however it does not provide a particularly attractive or friendly interface. In the following sections, we will show how simple client interfaces can be built using Visual Studio and simple HTML.

VISUAL STUDIO® CLIENTS

The Microsoft® Visual Studio® development system's support for web services is very good and this is not restricted to the server side. To demonstrate, we will build two simple clients to call the web service and display the data. The first will use Visual Basic forms and the second ASP.NET.

In order to build applications in Visual Studio that call web services we need to build a proxy class for the service. This class is used by the client and handles the generation of the SOAP messages needed to call the methods of our web service. In this way, much of the complexity of web services is hidden from the main code of the client.

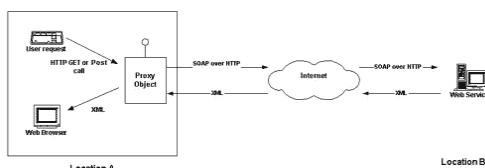


Figure 3 - Web Service Proxy Classes

We could build the proxy class manually using command line programs, however we can get Visual Studio to do it for us by adding in a web reference to our project. This is similar to adding a reference to an external COM component.

To add the reference we right click on the project in the solution explorer and choose 'Add Web Reference'.

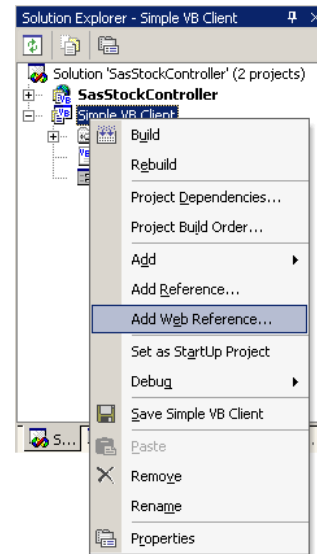


Figure 4 - Adding a Web Reference in VS.NET

This brings up a dialog for us to enter the address of the web service. In this case the web service will be running on our machine so we enter:

<http://localhost/SasStockController/Inventory.aspx>

Visual Studio then queries the web service to discover what methods it provides and displays the information. You will notice that this is the same information that was displayed through Internet Explorer earlier. In the background Visual Studio has also retrieved the WSDL XML document.

When we click the Add Reference button Visual Studio creates a proxy class for us to use in accessing the web service. This class means we can create references to the web service and call its methods just like we would with any other object in Visual Basic.

VISUAL BASIC FORMS (RICH CLIENT)

Our VB client will consist of a single form. The form needs to contain a text box for entry of the company code, a button to retrieve the data and a datagrid to display the results.

Behind the button we need to add code to:

- Check the company code parameter
- create a web service object
- call the web method to retrieve the data
- display the data in the datagrid.
- Clear up the objects

Since Visual Studio has created a proxy class for our web service, declaring and initialising the web service is the same as using any other object in Visual Basic.

```
' Create the web service object
Dim webService As _
    localhost.SimpleInventoryService
```

Building a SAS Based Web Service

```
webService = New _  
    localhost.SimpleInventoryService()
```

By default the proxy class uses the hostname of the web service as its namespace, so to create the web service object we specify the hostname used in creating the web reference. This name can be changed through the web references properties, however all references to the class in code must also be updated.

Now we've got our web service object we can call the web method.

```
' Call the web method  
Dim stockLevels As Data.DataSet  
stockLevels = _  
    webService.GetStockLevels(companyRef)
```

As you can see, this is done in exactly the same way as calling any other class's methods.

We display the data in the grid by setting the data grid's DataSource property to our resulting data table:

```
' Display data in data grid  
Me.DataGrid.DataSource = _  
    stockLevels.Tables(0)
```

Finally, we clear the web service object

```
' clear the web service object  
webService = Nothing
```

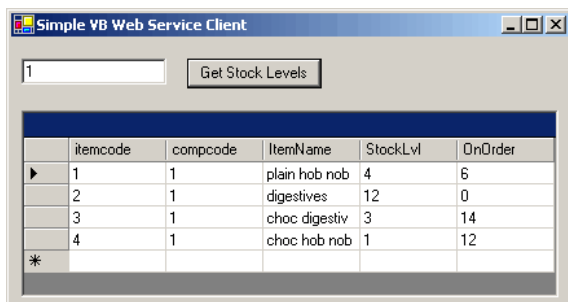


Figure 5 - Simple VB Client

ASP.NET CLIENT

The ASP.NET System allows us to write server pages that generate HTML dynamically. The ASP code executes on the server and results in HTML that is returned to the browser for display to the user. This allows us to generate highly dynamic web pages and web applications, however it is important to remember that the code is executed on the server, not in the browser. Therefore, any interaction with our web service will be between the web server and the web service. This is illustrated in Figure 6 below.

We can write ASP.NET applications in any language supported by the .NET Framework. In order to show how similar ASP.NET can be to VB rich client applications we will continue to use VB.

One of the new features of ASP.NET is Web Forms. These allow us to program using the event model familiar from rich client development with ASP.NET taking care of the intricacies of maintaining the state of the various objects on the page.

As before our first step is to create a web reference to our service. Since we are still using the Visual Studio development environment this is done in the same way.

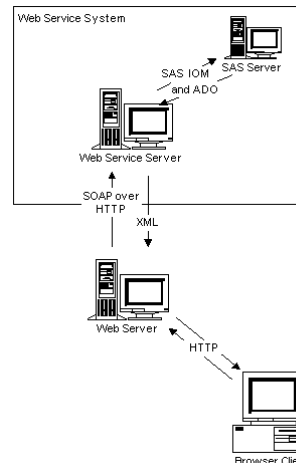


Figure 6 - Accessing a Web Service through ASP.NET

To mimic the VB client above we add the ASP.NET equivalent controls onto the web form. These controls are designed to behave like their rich client cousins. In fact they do it so well that we can take the exact code from the VB client and place it behind the ASP.NET button. We only need to make two changes in order to allow the code to work.

1. The code will execute on a web server, therefore we cannot display a normal message box if the parameter is non-numeric. So we change the line to set the input box's value to indicate an incorrect value; `Me.companyCode.Text = "Invalid Company Code"`
2. The data grid behaves slightly differently. Once we have set its DataSource property, we must call the `DataBind()` method to produce the table; `Me.DataGrid.DataBind()`

The full code for the ASP.NET client is given in Appendix 4.

Running the client, we can see that it responds in much the same way.

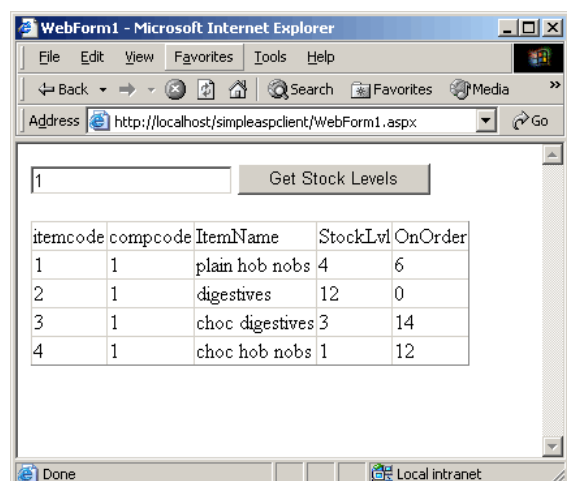


Figure 7 - Simple ASP Client

BROWSER CLIENT

Accessing web services using server pages as shown above maintains the web services advantage of separating client presentation from the data manipulation layer, however the web server is performing the web service communication (As



illustrated in Figure 6). It would be far nicer to allow the browser to be the client in the web service communication. This scenario is illustrated in Figure 8 below. Here we can see that the Browser client is accessing the web server to retrieve normal web pages, but accessing the web service directly to retrieve the data it requires.

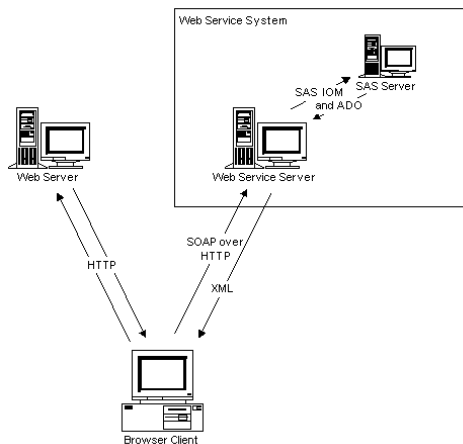


Figure 8 - Accessing a Web Service through IE

It is possible to write web pages that can call web services directly with both Netscape 7 and Internet Explorer 5. Unfortunately, this area is still very immature and so we have the age-old problem of Microsoft and Sun's implementations differing considerably. Sun's implementation in Netscape is integrated into the Mozilla browser and appears reasonably mature. In comparison Microsoft's still has an 'unsupported' status.

That said, Internet Explorer is by far the dominant browser at this time, so we shall produce a simple client using the Microsoft method.

Internet Explorer (IE) uses its extensible behaviours to add web service functionality to a web page. The web service behaviour comes in the form of an HTML component – webservice.htc – that can be downloaded from the Microsoft IE Components web site.

To use the behaviour it has to be attached to an element. The extra methods it provides are then added to that component.

```
<div id="webServiceDiv"
    style="behavior:url(webService.htc)"
    onresult="handleResult()">
```

In our example, we use the name webServiceDiv to identify the element, and therefore that is what we will use in order to call the web service methods.

There are three parts to the code we need to write to communicate with our web service.

- Initialisation
- Calling the web service
- Processing the response

Unlike in the previous examples the default calling mode in IE is asynchronous, therefore the calling of the service and the processing the response are very much separated.

The first thing we need to do is initialise the behaviour by telling it the location of our web service. This allows it to download the WSDL contract that specifies what methods our service provides.

```
svcElm
document.getElementById("webServiceDiv");

var sWSDL = "http://localhost/SasStockController/Inventory.aspx?WSDL"

svcElm.useService(sWSDL,"WebService");
```

Notice that we have to specify the parameter **WSDL** so the behaviour retrieves the XML definition, rather than the user-friendly HTML page generated by our service.

The web services methods are called using the web behaviours callService method. We pass the method name and the method parameters.

```
var parmValue = document.
getElementById("parameter").value;

svcElm.WebService.callService(
    "GetStockLevels", parmValue);
```

When the web service sends the result it is passed to the handling function. In our example we have linked the handleResult() function by specifying it handles the onresult event of the div element. Alternatively, we could have passed a callback function to callService when invoking the web service.

When the web service sends the result, it is passed by the behaviour to handleResult(). This checks that the call did not produce an error and then processes the result data. In order to translate the xml into an HTML table we use an XSL transform.

RUNNING THE BROWSER CLIENT

Figure 9 shows the output of the web browser client. This is aesthetically different from the server page client, however we could easily change the HTML to display the output in an identical manner.

When you run the browser client you will notice that clicking the 'Retrieve Stock Levels' button does not submit the page back to the server. This is one of the advantages of using web services directly within the browser – requests for data can be made without having to rebuild the entire page.

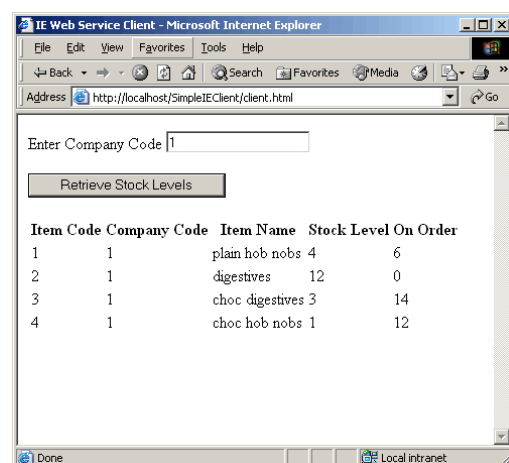


Figure 9 - Web Browser Client

Combining this ability to retrieve data with the flexibility of server pages to generate the initial page view provides us with a very powerful mechanism for generating web browser based applications.



CONCLUSION

Today's businesses demand a more integrated, flexible architecture to provide the information and analytics they require. The web services architecture provides us with a method of meeting that demand.

Coupling Web Services flexibility of delivery with SAS's data manipulation abilities provides us a powerful system for meeting the businesses needs.

Web services are still very much an evolving technology, however, even at this early stage in their development, we have seen how easily a web service can be created and used to manipulate and retrieve data through SAS.

REFERENCES

How to Implement a Web Service with SAS - A Business Scenario by Anton Fuchs, Daniel Jahn, Tony Dean and Bryan Allen (SAS Institute).

Web Services with SAS® by Anton Fuchs (SAS Institute)

Microsoft IE Components Website –
<http://ie.components.microsoft.com>

MSDN Web Services Website –
<http://msdn.microsoft.com/webservices/>

Netscape Dev Edge Website – <http://devedge.netscape.com/>

CONTACT INFORMATION

Your comments and questions are valued and encouraged.
Contact the author at:

Author Name: Jonathan Williamson
Company: Amadeus Software Limited
Address: Orchard Farm, Witney Lane,
Leaffield, Oxfordshire, OX29 9PG
Phone: 01993 878287
Fax: 01993 878042
Email: jonathan.williamson@amadeus.co.uk
Web: www.amadeus.co.uk

Visual Studio and Visual Basic are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries

SAS and SAS Integration Technologies are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries.

Netscape is a registered trademark of Netscape Communications Corporation in the U.S. and other countries.

Other brand and product names are trademarks or registered trademarks of their respective companies.

APPENDIX 1 EXAMPLE WEB SERVICE RESPONSE

```
<?xml version="1.0" encoding="utf-8"?>
<DataSet xmlns="http://amadeus.co.uk/WebServices/Inventory">
  <xs:schema id="NewDataSet" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xs:element name="NewDataSet" msdata:IsDataSet="true" msdata:Locale="en-GB">
      <xs:complexType>
        <xs:choice maxOccurs="unbounded">
          <xs:element name="Table">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="itemcode" type="xs:double" minOccurs="0" />
                <xs:element name="compcode" type="xs:double" minOccurs="0" />
                <xs:element name="ItemName" type="xs:string" minOccurs="0" />
                <xs:element name="StockLvl" type="xs:double" minOccurs="0" />
                <xs:element name="OnOrder" type="xs:double" minOccurs="0" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>
  <diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata" xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
    <NewDataSet xmlns="">
      <Table diffgr:id="Table1" msdata:rowOrder="0">
        <itemcode>1</itemcode>
        <compcode>1</compcode>
        <ItemName>plain hob nobs</ItemName>
        <StockLvl>4</StockLvl>
        <OnOrder>6</OnOrder>
      </Table>
      <Table diffgr:id="Table2" msdata:rowOrder="1">
        <itemcode>2</itemcode>
        <compcode>1</compcode>
        <ItemName>digestives</ItemName>
        <StockLvl>12</StockLvl>
        <OnOrder>0</OnOrder>
      </Table>
      <Table diffgr:id="Table3" msdata:rowOrder="2">
        <itemcode>3</itemcode>
        <compcode>1</compcode>
        <ItemName>choc digestives</ItemName>
        <StockLvl>3</StockLvl>
        <OnOrder>14</OnOrder>
      </Table>
      <Table diffgr:id="Table4" msdata:rowOrder="3">
        <itemcode>4</itemcode>
        <compcode>1</compcode>
        <ItemName>choc hob nobs</ItemName>
        <StockLvl>1</StockLvl>
        <OnOrder>12</OnOrder>
      </Table>
    </NewDataSet>
  </diffgr:diffgram>
</DataSet>
```



APPENDIX 2 VB.NET WEB SERVICE

The following two files define the VB.NET Web Service.

INVENTORY.ASMX

```
<%@ WebService Language="vb" Codebehind="Inventory.asmx.vb"
Class="SasStockController.SimpleInventory" %>
```

INVENTORY.ASMX.VB

```
Imports System.Web.Services
Imports System.Data
Imports System.Threading

<WebService(Namespace:="http://amadeus.co.uk/WebServices/Inventory", _
    Name:="Simple Inventory Service", _
    Description:="Provides access to the stock inventory.")> _
Public Class SimpleInventory
    Inherits System.Web.Services.WebService

    #Region " Web Services Designer Generated Code "
    ' VS.NET generated code is placed here.
    #End Region

    <WebMethod(Description:="Gets the stock level for all products for the specified supplier.")> _
    Public Function GetStockLevels(ByVal supplierRef As Integer) As DataSet

        Dim obWorkspaceManager As New SASWorkspaceManager.WorkspaceManager()
        Dim xmlInfo As String
        Dim obSAS As SAS.Workspace

        obSAS = obWorkspaceManager.Workspaces.CreateWorkspaceByServer("", _
            SASWorkspaceManager.Visibility.VisibilityProcess, Nothing, "", "", xmlInfo)

        obSAS.LanguageService.StoredProcessService.Repository = "file:c:\SASRepository"

        Try
            Dim params As String
            params = "company=" & supplierRef & " outdata=work.output"
            obSAS.LanguageService.StoredProcessService.Execute("GetStockLevels", params)
        Catch ex As Exception
            obWorkspaceManager.Workspaces.RemoveWorkspace(obSAS)
            Exit Function
        End Try

        Dim obDataset As New System.Data.DataSet()
        Dim obAdapter As New System.Data.OleDb.OleDbDataAdapter("select * from work.output", _
            "provider=sas.iomprovider.1; SAS Workspace ID=" & obSAS.UniqueIdentifier)

        obAdapter.Fill(obDataset)

        GetStockLevels = obDataset

        obAdapter = Nothing
        obWorkspaceManager.Workspaces.RemoveWorkspace(obSAS)

    End Function
End Class
```



APPENDIX 3 SAS CODE

The following SAS code is held in GetStocklevels.sas in the SAS repository directory.

```
* Expected parameters ;
* outdata - output dataset ;
* itemcode - items stock code ;
%let outdata=stocksys.out;
%let company=1;

*ProcessBody;

libname stocksys 'c:\SASRepository\Data';

data &outdata;
  set stocksys.levels (where=(compcode=&company)
                        keep=compcode itemcode itemname stocklvl onorder);
run;
```

APPENDIX 4 VB.NET WEB SERVICE CLIENT

```

Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub

    'Form overrides dispose to clean up the component list.
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    Friend WithEvents GetStockLevels As System.Windows.Forms.Button
    Friend WithEvents CompanyCode As System.Windows.Forms.TextBox
    Friend WithEvents DataGrid As System.Windows.Forms.DataGrid
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
        Me.GetStockLevels = New System.Windows.Forms.Button()
        Me.CompanyCode = New System.Windows.Forms.TextBox()
        Me.DataGrid = New System.Windows.Forms.DataGrid()
        CType(Me.DataGrid, System.ComponentModel.ISupportInitialize).BeginInit()
        Me.SuspendLayout()
        '
        'GetStockLevels
        '
        Me.GetStockLevels.Location = New System.Drawing.Point(136, 16)
        Me.GetStockLevels.Name = "GetStockLevels"
        Me.GetStockLevels.Size = New System.Drawing.Size(104, 23)
        Me.GetStockLevels.TabIndex = 0
        Me.GetStockLevels.Text = "Get Stock Levels"
        '
        'CompanyCode
        '
        Me.CompanyCode.Location = New System.Drawing.Point(8, 16)
        Me.CompanyCode.Name = "CompanyCode"
        Me.CompanyCode.Size = New System.Drawing.Size(112, 20)
        Me.CompanyCode.TabIndex = 1
        Me.CompanyCode.Text = "enter company code"
        '
        'DataGrid
        '
        Me.DataGrid.Anchor = (((System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Bottom) _
Or System.Windows.Forms.AnchorStyles.Left) _
Or System.Windows.Forms.AnchorStyles.Right) _
        Me.DataGrid.DataMember = ""
        Me.DataGrid.HeaderForeColor = System.Drawing.SystemColors.ControlText
        Me.DataGrid.Location = New System.Drawing.Point(8, 56)
        Me.DataGrid.Name = "DataGrid"
        Me.DataGrid.Size = New System.Drawing.Size(496, 256)
        Me.DataGrid.TabIndex = 2
        '
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(512, 317)
        Me.Controls.AddRange(New System.Windows.Forms.Control() {Me.DataGrid, Me.CompanyCode,
Me.GetStockLevels})
        Me.Name = "Form1"
        Me.Text = "Simple VB Web Service Client"
        CType(Me.DataGrid, System.ComponentModel.ISupportInitialize).EndInit()
        Me.ResumeLayout(False)
    End Sub

```



```

End Sub

#End Region

Private Sub GetStockLevels_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles GetStockLevels.Click
    Dim companyRef As Integer

    ' Check the user has entered a number
    If Not IsNumeric(Me.CompanyCode.Text) Then
        MsgBox("You must enter a numeric company code.", MsgBoxStyle.Critical, "Invalid Company
Code")
        Exit Sub
    End If
    companyRef = Integer.Parse(Me.CompanyCode.Text)

    ' Create the web service object
    Dim webService As localhost.SimpleInventoryService
    webService = New localhost.SimpleInventoryService()

    ' Call the web method
    Dim stockLevels As Data.DataSet
    stockLevels = webService.GetStockLevels(companyRef)

    ' Display data in data grid
    Me.DataGrid.DataSource = stockLevels.Tables(0)

    ' clear the web service object
    webService = Nothing

End Sub
End Class

```



APPENDIX 5 ASP.NET WEB SERVICE CLIENT

Below is the complete source code for the ASP.NET Web Service Client. The designer generated code has been left out, this code is not required unless you are using VS.Net designers to edit the code.

WEBFORM1.ASPX

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="WebForm1.aspx.vb"
Inherits="SimpleASPCClient.WebForm1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<title>WebForm1</title>
<meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
<meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
<meta name="vs_defaultClientScript" content="JavaScript">
<meta name="vs_targetSchema" content="http://schemas.microsoft.com/intellisense/ie5">
</HEAD>
<body>
<form id="Form1" method="post" runat="server">
<P>
<asp:TextBox id="companyCode" runat="server"></asp:TextBox>
<asp:Button id="GetStockLevels" runat="server" Text="Get Stock Levels"></asp:Button>
</P>
<P>
<asp:DataGrid id="DataGrid" runat="server">
</asp:DataGrid>
</P>
</form>
</body>
</HTML>
```

WEBFORM1.ASPX.VB

```
Public Class WebForm1
Inherits System.Web.UI.Page
Protected WithEvents GetStockLevels As System.Web.UI.WebControls.Button
Protected WithEvents DataGrid As System.Web.UI.WebControls.DataGrid
Protected WithEvents companyCode As System.Web.UI.WebControls.TextBox

#Region " Web Form Designer Generated Code "
' VS.NET generated code is placed here.
#End Region

Private Sub GetStockLevels_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles GetStockLevels.Click
Dim companyRef As Integer

' Check the user has entered a number
If Not IsNumeric(Me.companyCode.Text) Then
Me.companyCode.Text = "Invalid Company Code"
Exit Sub
End If
companyRef = Integer.Parse(Me.companyCode.Text)

' Create the web service object
Dim webService As localhost.SimpleInventoryService
webService = New localhost.SimpleInventoryService()

' Call the web method
Dim stockLevels As Data.DataSet
stockLevels = webService.GetStockLevels(companyRef)

' Display data in data grid
Me.DataGrid.DataSource = stockLevels.Tables(0)
Me.DataGrid.DataBind()

' clear the web service object
webService = Nothing

End Sub
End Class
```

APPENDIX 6 IE WEB SERVICE CLIENT

Below is the complete source code for the Internet Explorer Web Service Client. These two files must be copied to the same directory on the web server along with the Web Service behaviour (webservice.htc). The behaviour can be downloaded from the Microsoft IE Components site - <http://ie.components.microsoft.com/behaviors/behaviors/webService/webService.asp>

CLIENT.HTML

```
<html>
<head>
<title>IE Web Service Client</title>

<script type="text/javascript">
    var svcElm;

    function init(){
        svcElm = document.getElementById("webServiceDiv");
        var sWSDL = "http://localhost/SasStockController/Inventory.asmx?WSDL";
        svcElm.useService(sWSDL,"WebService");
    }

    function doWebService(){
        svcElm.innerHTML = "Invoking web service..."
        var parmValue = document.getElementById("parameter").value;
        svcElm.WebService.callService("GetStockLevels", parmValue);
    }

    function handleResult(){
        var html = "";
        var r = event.result;

        if (r.error){
            html += "A service error occurred:\n\n";
            html += r.errorDetail.string;
        }
        else {
            // transform xml into html table and place in div element;
            var xmldoc=new ActiveXObject("MSXML2.DOMDocument");
            xmldoc.async=false;
            xmldoc.loadXML(r.value.xml);

            var xsl doc=new ActiveXObject("MSXML2.DOMDocument");
            xsl doc.async=false;
            xsl doc.load("ADOTable2.xslt");
            html=xmldoc.transformNode(xsl doc);
        }

        webServiceDiv.innerHTML=html;
    }
</script>

</head>

<body onload="init();">
    <form action="#">

        <p>Enter Company Code <input type="text" name="parameter"></p>
        <p><input type="button" name="getWebService" value="Retrieve Stock Levels"
onclick="doWebService();"></p>
    </form>

    <div id="webServiceDiv" style="behavior:url(webService.htc)" onresult="handleResult()">

    </div>

</body>
</html>
```

ADOTABLE2.XSLT

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="html" />

<xsl:template match="//NewDataSet">
```



```

    <table>
      <tr><th>Item      Code</th><th>Company      Code</th><th>Item      Name</th><th>Stock
Level</th><th>On Order</th></tr>
      <xsl:apply-templates />
    </table>
</xsl:template>

<xsl:template match="Table">
  <tr>
    <td><xsl:value-of select="itemcode"/></td>
    <td><xsl:value-of select="compcode"/></td>
    <td><xsl:value-of select="ItemName"/></td>
    <td><xsl:value-of select="StockLvl"/></td>
    <td><xsl:value-of select="OnOrder"/></td>
  </tr>
</xsl:template>

</xsl:stylesheet>

```