

Web Services Tools and Demo

Typical Tools for Design, Development and Deployment of Web Services
With demos of creating simple WS clients and WS servers

Version 15

WS tools and demos – the “How To’s” of WS

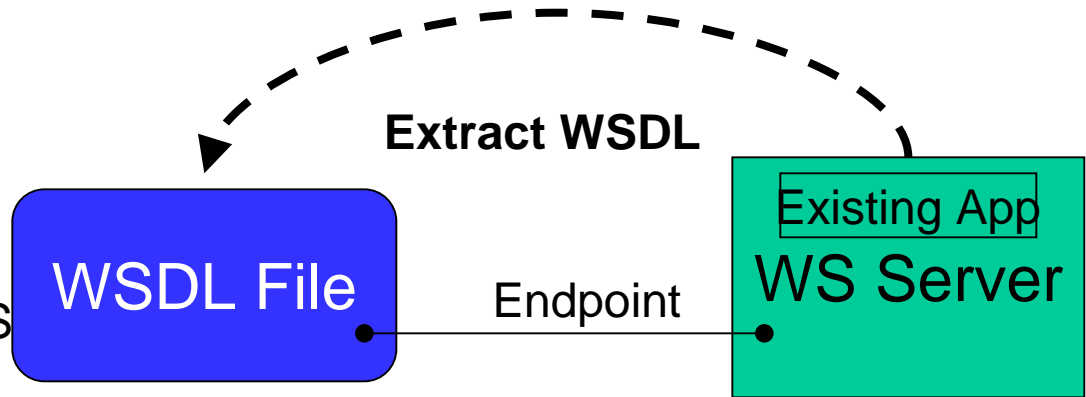
- Tools are needed for:
 - Creating/defining XML documents and schemas
 - Creating/modifying service definitions/WSDL files
 - Mapping WS artifacts to/from programming concepts
 - Both directions may be needed depending on the integration scenario
 - Managing and locating deployed Web Services
- Some typical and available tools:
 - XMLSPY – General Purpose for XML docs and WSDL files
 - Artix – C++/Java-based Web Services
 - Enterprise Web Services & middleware integration toolkit from IONA
 - VS.NET tool set - Microsoft
 - J2EE/App Servers – J2EE in general – e.g. EJB
 - Many vendors and many tools, ... and more every day
 - IONA, IBM, Sun, BEA, Cape Clear, Altova (XMLSpy), etc.

Two Basic Approaches for building a Web Service

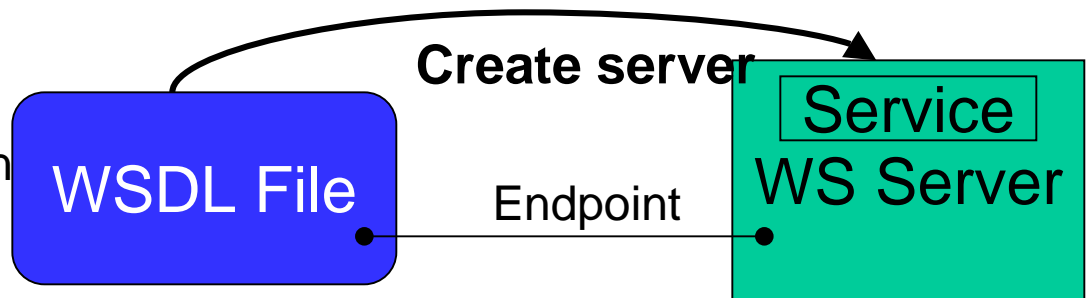
- **From Scratch** – a new application or system
 - Start with WSDL and/or data/message schemas and produce the application using generated application building artifacts:
 - “blank” client and server, client-side stubs & server-side skeletons
 - Write the business logic in the “blank” server
 - Fill out the “blank” client code
- **From Existing Systems** – either
 - Automatically derive the WSDL from the existing interfaces for the application or from data schema files
 - There are automatic WSDL generators for CORBA IDL, Java classes, J2EE EJBs, COBOL, other middleware (MQ, Tuxedo,...), message/XML Schema files, . . .
 - Often you have to enhance the generated WSDL
 - Do a top-down design of the WSDL, and implement the service using new code and the existing systems
 - You may get better WSDL. Remember that one of the main goals of a service is to keep its clients simple.

Diagrams to show these Development Approaches

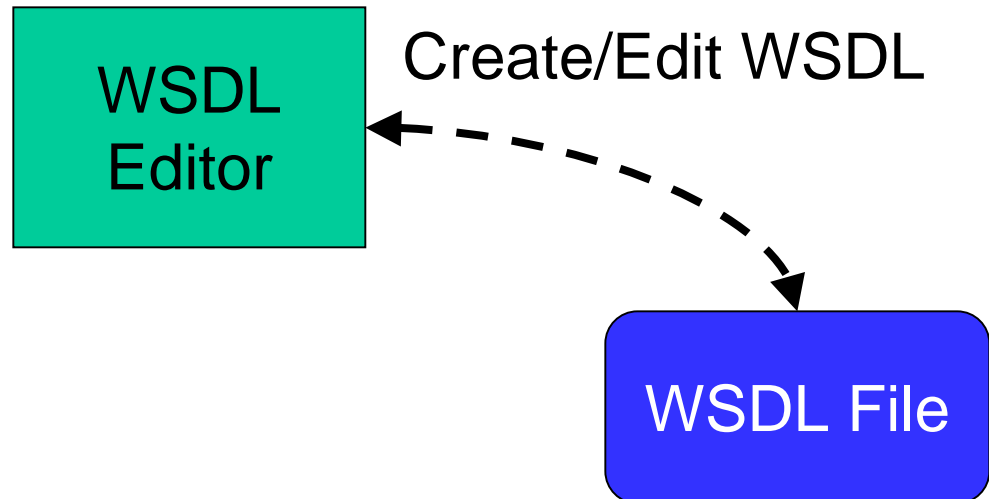
Existing App has an API
 Extract WSDL from API
 Build client and server skeleton
 Client calls App API through WS



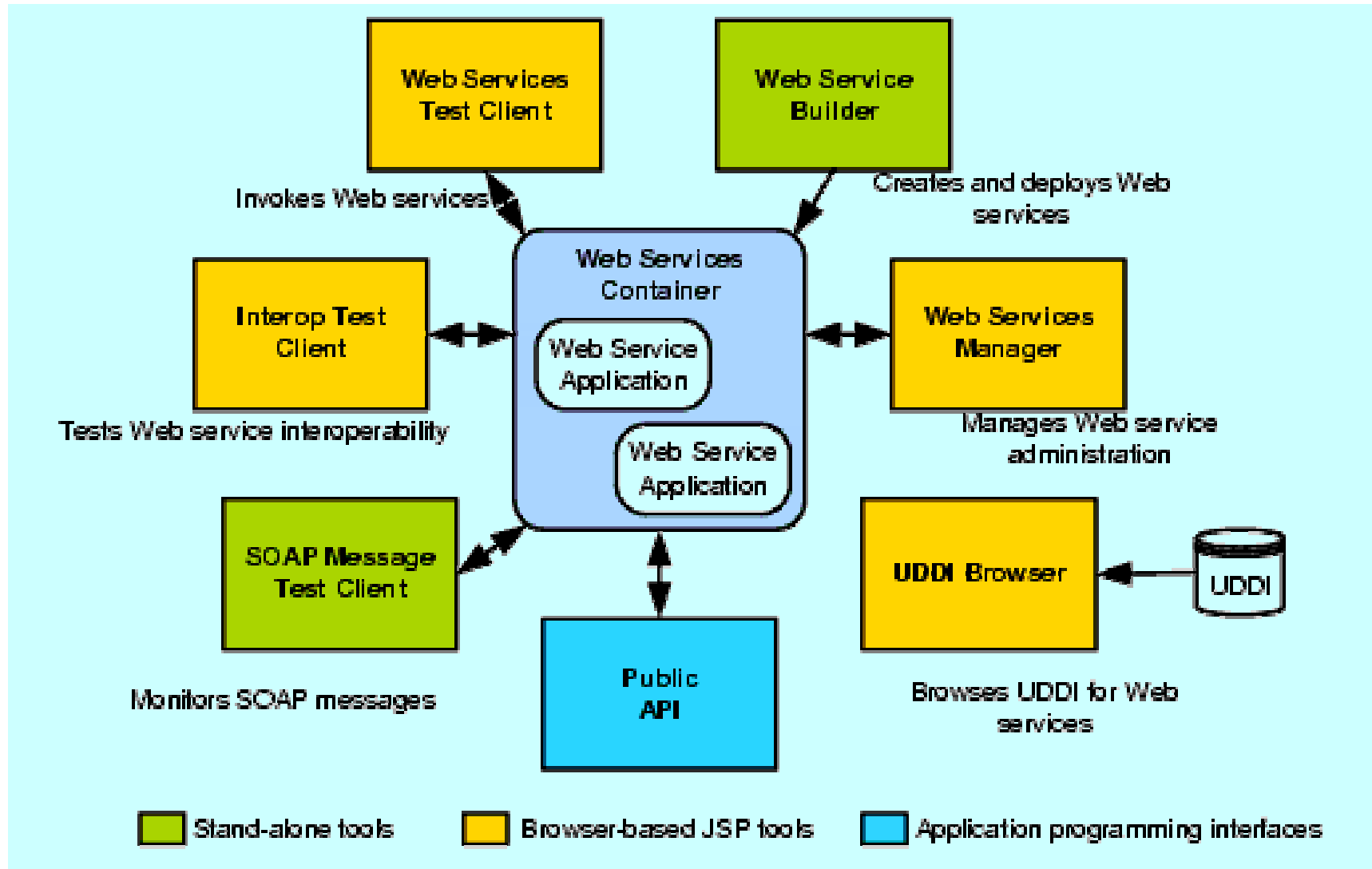
Design a WSDL (top down)
 Create client and server skeleton
 Implement server/service code
 Client calls service through WS



Design/update the WSDL
 Utilize in above scenarios during
 development/evolution of system



Typical WS Development/ Deployment Components



Development Demo Scenarios

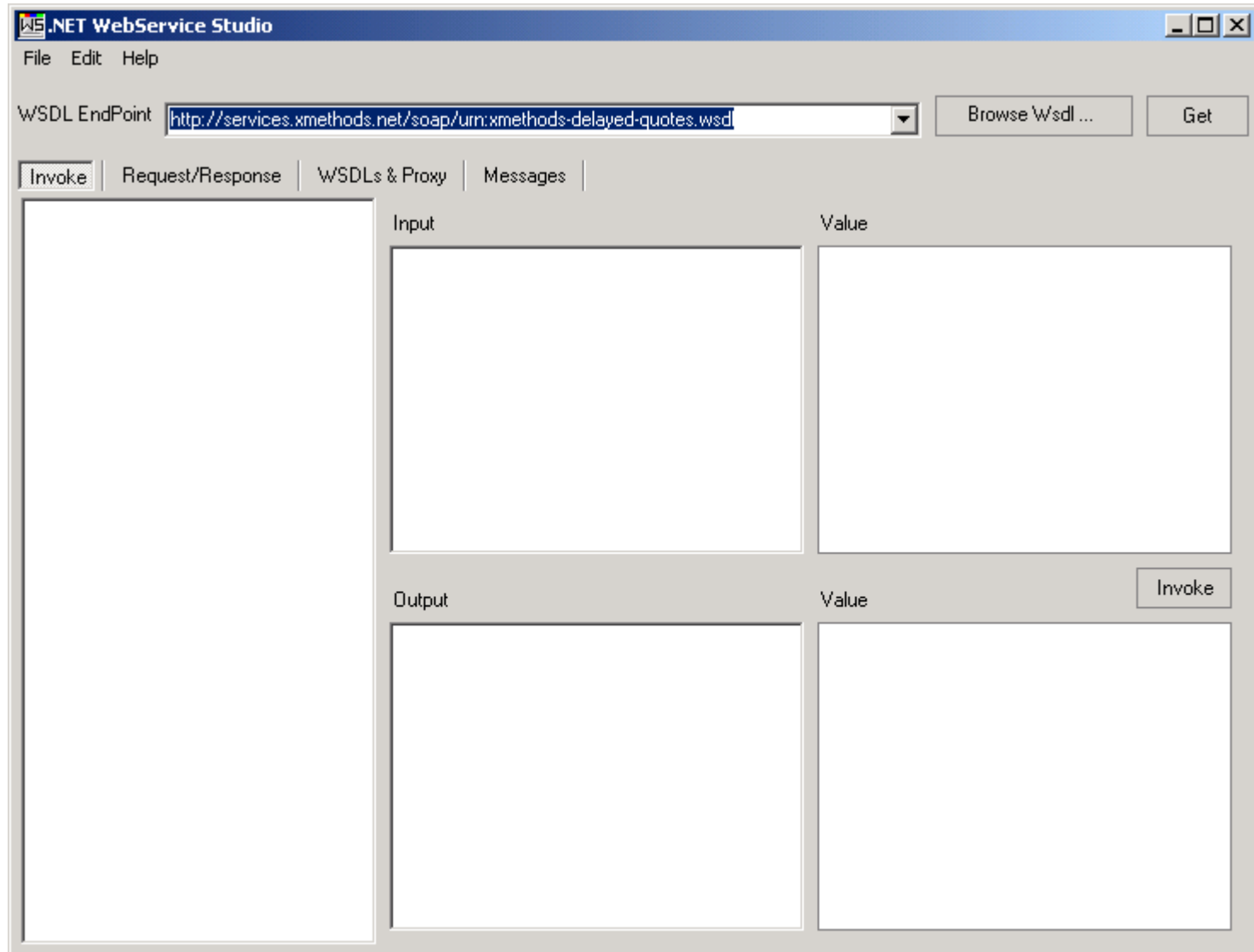
- Automatic client generation
 - Seeing WS clients in action ...
- Hello World – VB.NET client creation
 - Local Artix server
- Stock Quote – VB.NET client creation
 - Our original example - again

- Hello World – VB.NET Web Service Creation

Automated WS Client Generation

- A number of tools available for this task
- Generate Java, VB, C#, Javascript, ASP, JSP and other
- We will use **Web Services Studio 2.0**
 - Freeware product – 114KB – simple install
 - Reads/analyzes a WSDL file, located anywhere on the net
 - Creates/displays automatically generated client/proxy code
 - Simplifies the invocation of WS methods with arbitrary inputs, and then viewing the response messages
 - Can view the actual SOAP messages involved
- Available from:
 - <http://www.gotdotnet.com/> - locate WebServiceStudio 2.0 page
 - Requires .NET SDK environment
- Let's see it in action ...

Screen Shot – WS Studio 2.0



Web Service Client Created and Executed

The screenshot shows the .NET WebService Studio interface. The WSDL EndPoint is set to `http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl`. The 'Invoke' tab is active, showing the 'getQuote' method selected in the left pane. The 'Input' pane shows the method call with a body containing `String symbol = IONA`. The 'Value' pane shows the input parameters: Type: **System.String**, Value: **IONA**, and IsNull: **False**. The 'Output' pane shows the result: `Single result = 7.25`. An 'Invoke' button is visible in the bottom right corner of the output area.

Type	System.String
Value	IONA
IsNull	False

Value
Single result = 7.25

SOAP Messages to/from Service

The screenshot displays the .NET WebService Studio interface. The WSDL EndPoint is set to `http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl`. The **Request/Response** tab is active, showing the following details:

Property	Value
AllowAutoRedire	False
AllowWriteStrea	False
BasicAuthPassw	
BasicAuthUserN	
ContentType	text/xml; charset=...
HttpProxy	
KeepAlive	False
Method	POST
Pipelined	False
PreAuthenticate	False
SendChunked	False
SOAPAction	"urn:xmethods-dela
Timeout	100000
Url	http://66.28.98.12
UseCookieCont:	True
UseDefaultCred:	False

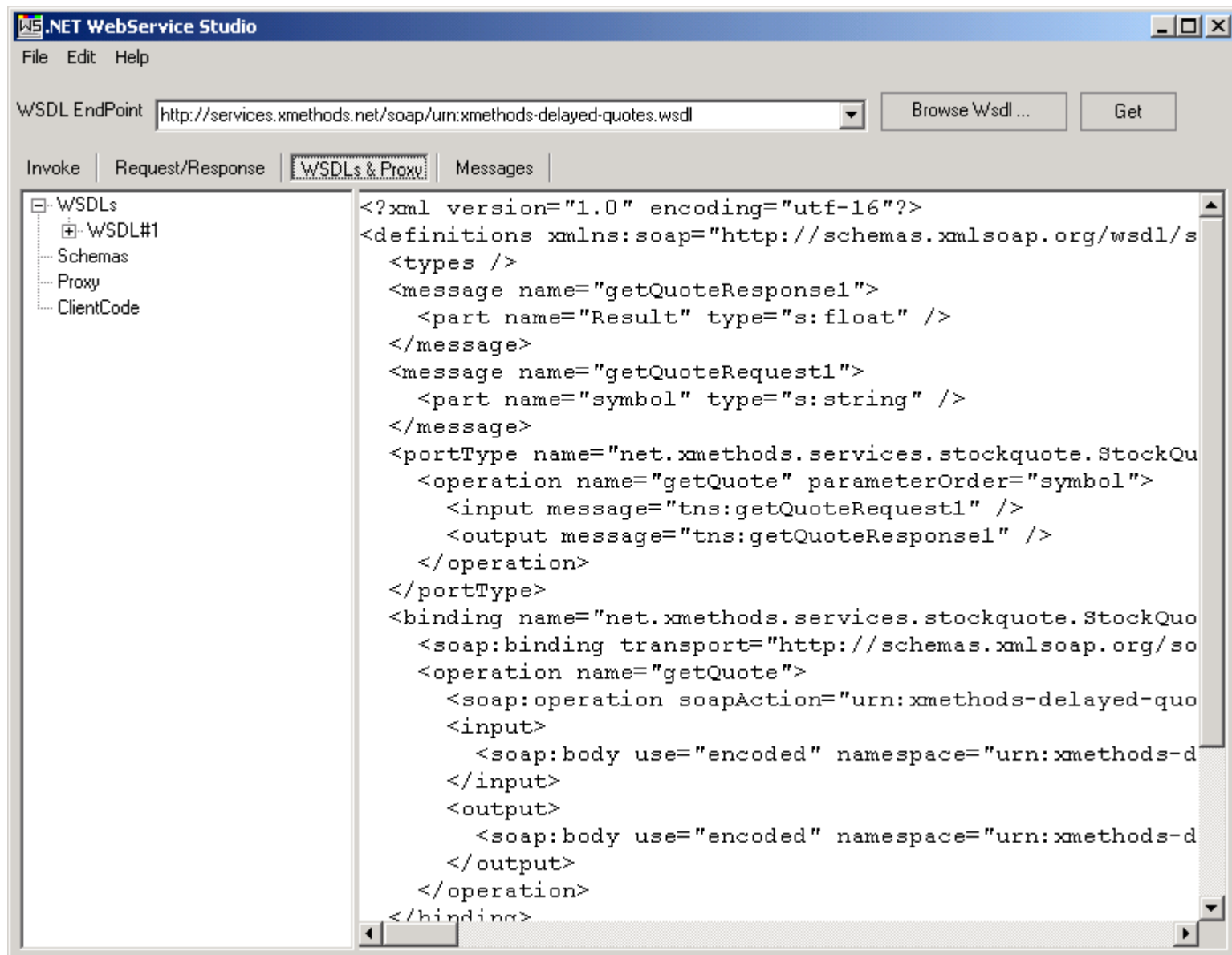
The **Request** pane shows the following XML:

```
<?xml version="1.0" encoding="utf-16"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope"
  <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding"
    <q1:getQuote xmlns:q1="urn:xmethods-delayed-quotes">
      <symbol xsi:type="xsd:string">IONA</symbol>
    </q1:getQuote>
  </soap:Body>
</soap:Envelope>
```

The **Response** pane shows the following XML:

```
Content-Length:490
<?xml version="1.0" encoding="utf-16"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope"
  <soap:Body>
    <n:getQuoteResponse xmlns:n="urn:xmethods-delayed-quotes"
      <Result xsi:type="xsd:float">7.25</Result>
    </n:getQuoteResponse>
  </soap:Body>
</soap:Envelope>
```

WSDL File display



The screenshot shows the .NET WebService Studio interface. The title bar reads ".NET WebService Studio". The menu bar includes "File", "Edit", and "Help". The "WSDL EndPoint" field contains the URL "http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl". There are "Browse Wsdl ..." and "Get" buttons to the right of the field. Below the field are four tabs: "Invoke", "Request/Response", "WSDLs & Proxy", and "Messages". The "WSDLs & Proxy" tab is selected. On the left, a tree view shows the WSDL structure: "WSDLs" (expanded) containing "WSDL#1", "Schemas", "Proxy", and "ClientCode". The main area displays the XML content of the WSDL file:

```
<?xml version="1.0" encoding="utf-16"?>
<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/s
  <types />
  <message name="getQuoteResponse1">
    <part name="Result" type="s:float" />
  </message>
  <message name="getQuoteRequest1">
    <part name="symbol" type="s:string" />
  </message>
  <portType name="net.xmethods.services.stockquote.StockQu
    <operation name="getQuote" parameterOrder="symbol">
      <input message="tns:getQuoteRequest1" />
      <output message="tns:getQuoteResponse1" />
    </operation>
  </portType>
  <binding name="net.xmethods.services.stockquote.StockQuo
    <soap:binding transport="http://schemas.xmlsoap.org/so
    <operation name="getQuote">
      <soap:operation soapAction="urn:xmethods-delayed-quo
      <input>
        <soap:body use="encoded" namespace="urn:xmethods-d
      </input>
      <output>
        <soap:body use="encoded" namespace="urn:xmethods-d
      </output>
    </operation>
  </binding>
```

A Different Stock Symbol ... and result

The screenshot shows the .NET WebService Studio interface. The WSDL EndPoint is set to `http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl`. The `Invoke` tab is selected, showing the `getQuote` method in the `netxmethodsservicesstockquoteStock` namespace.

Input:

- `getQuote`
 - Headers
 - Body
 - String symbol = IBM

Value:

Type	System.String
Value	IBM
IsNull	False

Output:

- `getQuote`
 - Headers
 - Body
 - Single result = 99.23

Value:

Type	System.Single
Value	99.23
IsNull	False

An `Invoke` button is visible in the bottom right corner of the output section.

A Different Stock Quote – the SOAP messages

The screenshot shows the .NET WebService Studio interface. The WSDL EndPoint is set to `http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl`. The Request/Response tab is active, displaying the following SOAP messages:

Request

```
<?xml version="1.0" encoding="utf-16"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <q1:getQuote xmlns:q1="urn:xmethods-delayed-quotes">
      <symbol xsi:type="xsd:string">IBM</symbol>
    </q1:getQuote>
  </soap:Body>
</soap:Envelope>
```

Response

```
Content-Length:491
<?xml version="1.0" encoding="utf-16"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <n:getQuoteResponse xmlns:n="urn:xmethods-delayed-quotes">
      <Result xsi:type="xsd:float">99.23</Result>
    </n:getQuoteResponse>
  </soap:Body>
</soap:Envelope>
```

The left pane shows the Request Properties table:

Property	Value
AllowAutoRedirection	False
AllowWriteStream	False
BasicAuthPassword	
BasicAuthUserName	
ContentType	text/xml; charset=utf-16
HttpProxy	
KeepAlive	False
Method	POST
Pipelined	False
PreAuthenticate	False
SendChunked	False
SOAPAction	"urn:xmethods-delayed-quotes"
Timeout	100000
Url	http://66.28.98.12
UseCookieContainer	True
UseDefaultCredentials	False

Creating Web Services Clients Automatically

Writing Client Code – Simple Examples

- Using Visual Basic within Visual Studio.NET
- Simple, unpolished examples
- Illustrate the underlying theme of ease of access to Web Services from most development environments
- Major building step
 - Add a “[Web Reference](#)” to the current project
 - Based solely on a WSDL file specification:
 - Local/remote file system
 - Remote URL

HelloWorld VB.NET Client

- Use a remotely stored WSDL file ([HelloWorld.wsdl](#)) on our demo system
 - Simple data exchange service
 - Only two operations – greetMe and sayHi
- Create a simple client (using VB.NET)
- Execute the client to invoke a remote service
 - Server written in C++ using IONA Artix product running on a remote server
- Let's do it!

The screenshot shows the Microsoft Development Environment (MDE) Start Page. The browser window title is "Microsoft Development Environment [design] - [Start Page]". The address bar shows "vs:/default.htm". The page features a blue "Start" banner at the top left. Below the banner is a "Get Started" sidebar with links: "What's New", "Online Community", "Headlines", "Search Online", "Downloads", "XML Web Services", "Web Hosting", and "My Profile". The main content area has two tabs: "Projects" (selected) and "Find Samples". Below the tabs is a table with two columns: "Name" and "Modified". The table lists four projects: "WSClient" (modified 1/20/2004), "COMDEXWS" (modified 12/12/2003), "WS" (modified 11/16/2003), and "COMDEX" (modified 11/13/2003). Below the table are two buttons: "Open Project" and "New Project". At the bottom of the window is an "Output" pane, which is currently empty, and a status bar showing "Done".

Microsoft Development Environment [design] - [Start Page]

File Edit View Tools Window Help

vs:/default.htm

Start

Get Started

- What's New
- Online Community
- Headlines
- Search Online
- Downloads
- XML Web Services
- Web Hosting
- My Profile

Projects

Find Samples

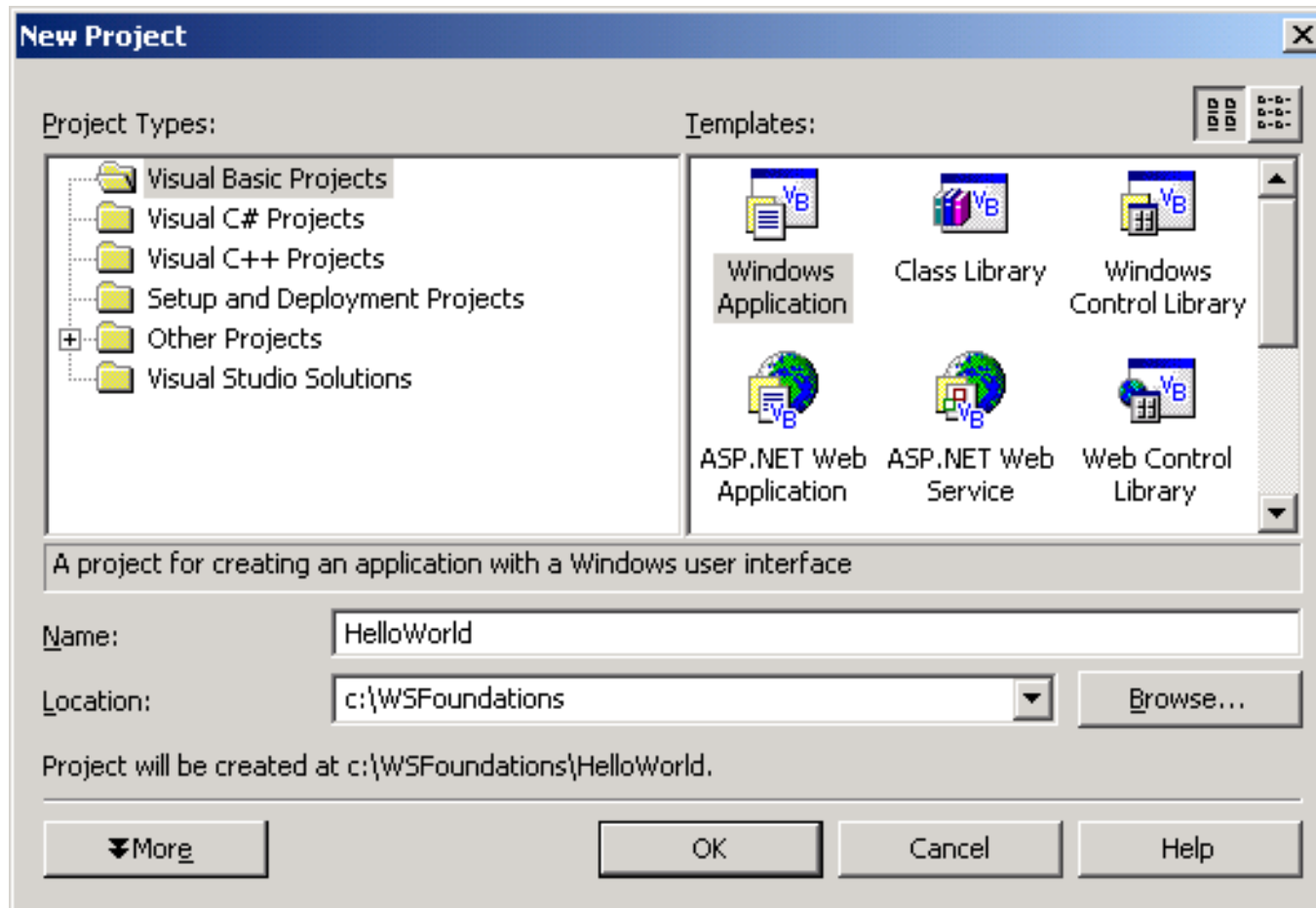
Name	Modified
WSClient	1/20/2004
COMDEXWS	12/12/2003
WS	11/16/2003
COMDEX	11/13/2003

[Open Project](#) [New Project](#)

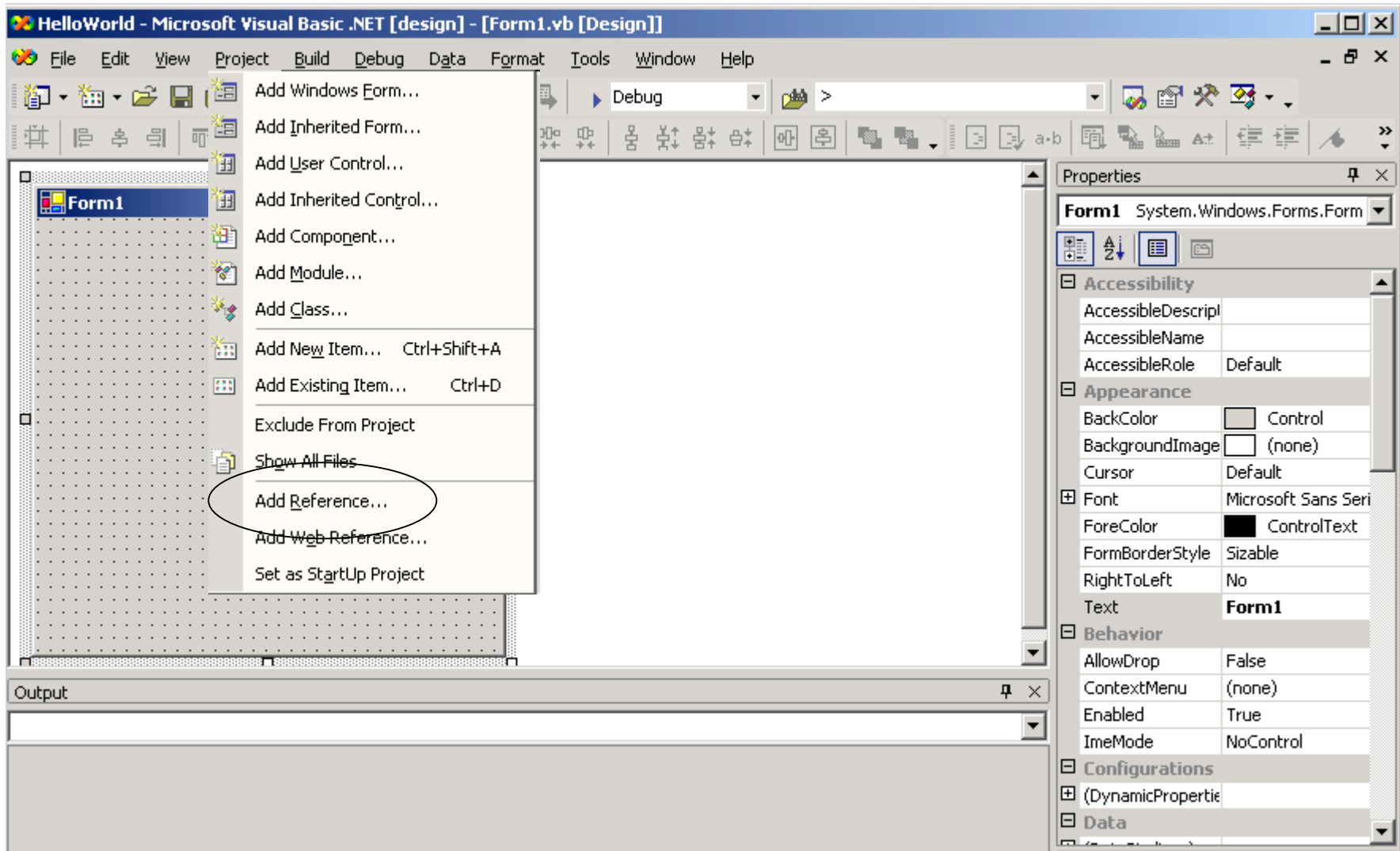
Output

Done

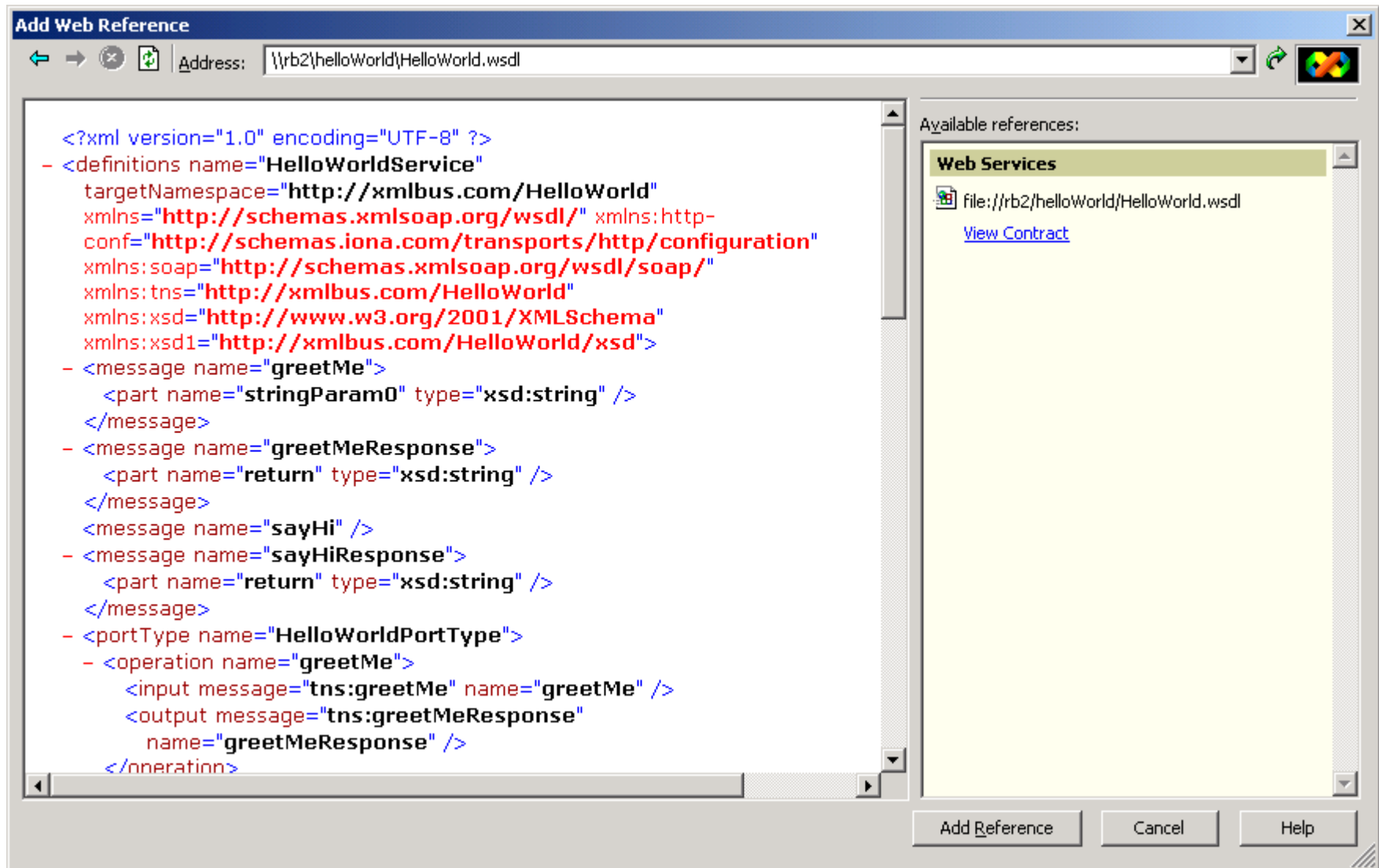
Windows Application for our Examples



Key Step: Add a Web Reference



Add Web Reference Dialog

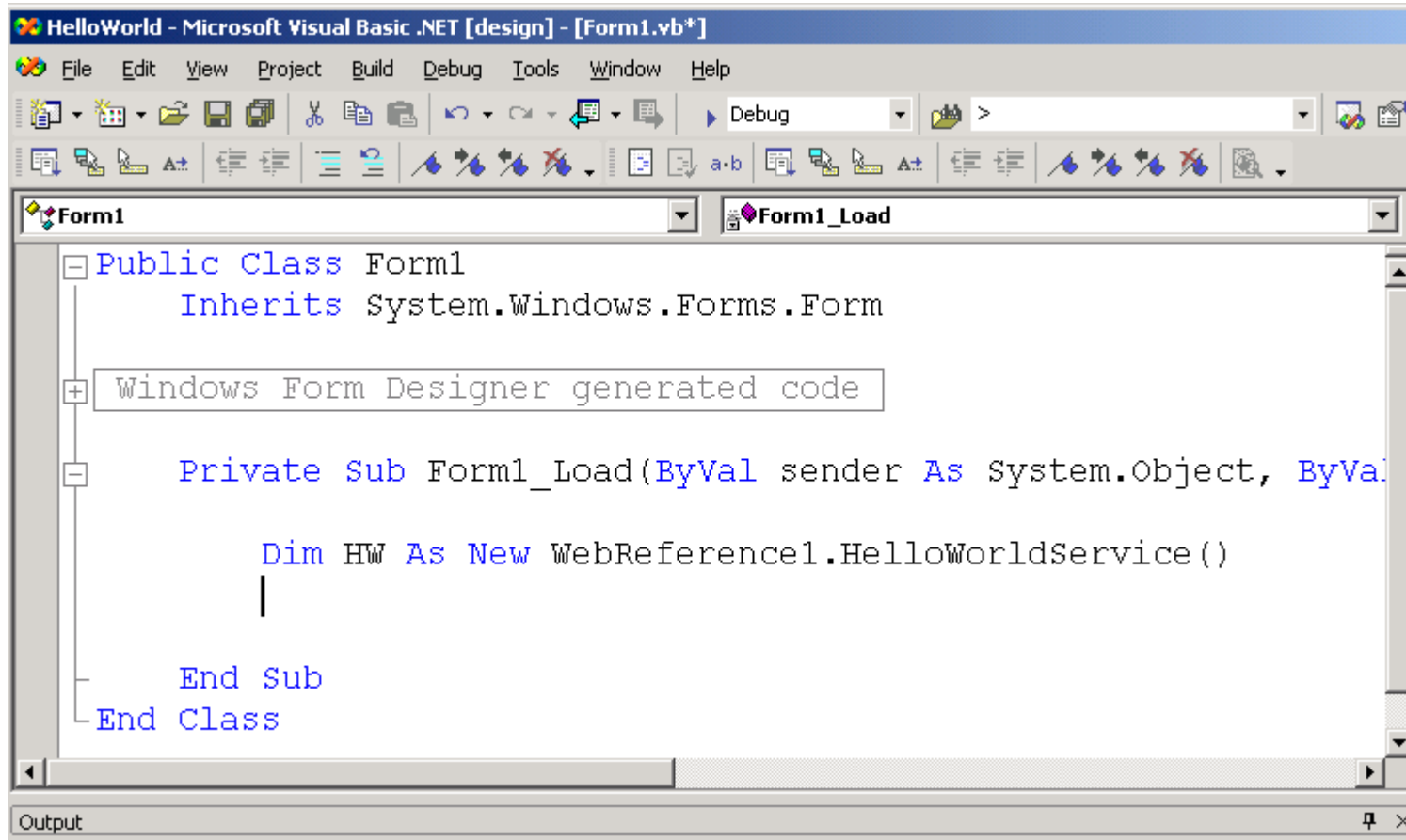


The screenshot displays the 'Add Web Reference' dialog box. The address bar shows the file path: \\rb2\helloWorld\HelloWorld.wsdl. The main text area contains the following WSDL code:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <definitions name="HelloWorldService"
  targetNamespace="http://xmlbus.com/HelloWorld"
  xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:http-
  conf="http://schemas.iona.com/transport/http/configuration"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://xmlbus.com/HelloWorld"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsd1="http://xmlbus.com/HelloWorld/xsd">
- <message name="greetMe">
  <part name="stringParam0" type="xsd:string" />
</message>
- <message name="greetMeResponse">
  <part name="return" type="xsd:string" />
</message>
<message name="sayHi" />
- <message name="sayHiResponse">
  <part name="return" type="xsd:string" />
</message>
- <portType name="HelloWorldPortType">
  - <operation name="greetMe">
    <input message="tns:greetMe" name="greetMe" />
    <output message="tns:greetMeResponse"
      name="greetMeResponse" />
  </operation>
```

On the right side, the 'Available references:' panel shows a list of 'Web Services' with one entry: 'File://rb2/helloWorld/HelloWorld.wsdl'. Below this entry is a 'View Contract' link. At the bottom of the dialog, there are three buttons: 'Add Reference', 'Cancel', and 'Help'.

Create a Proxy object to WS server

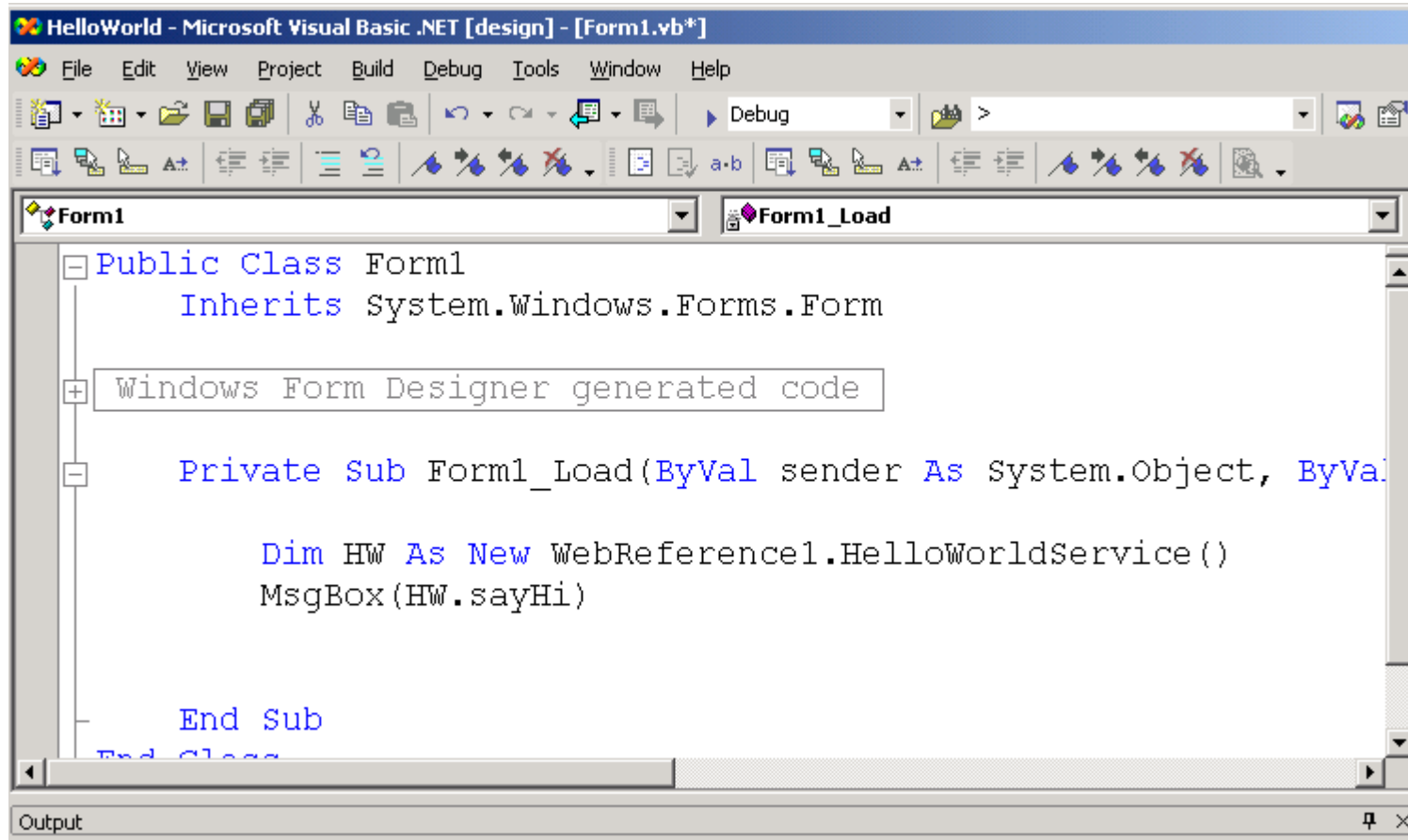


```
Public Class Form1
    Inherits System.Windows.Forms.Form

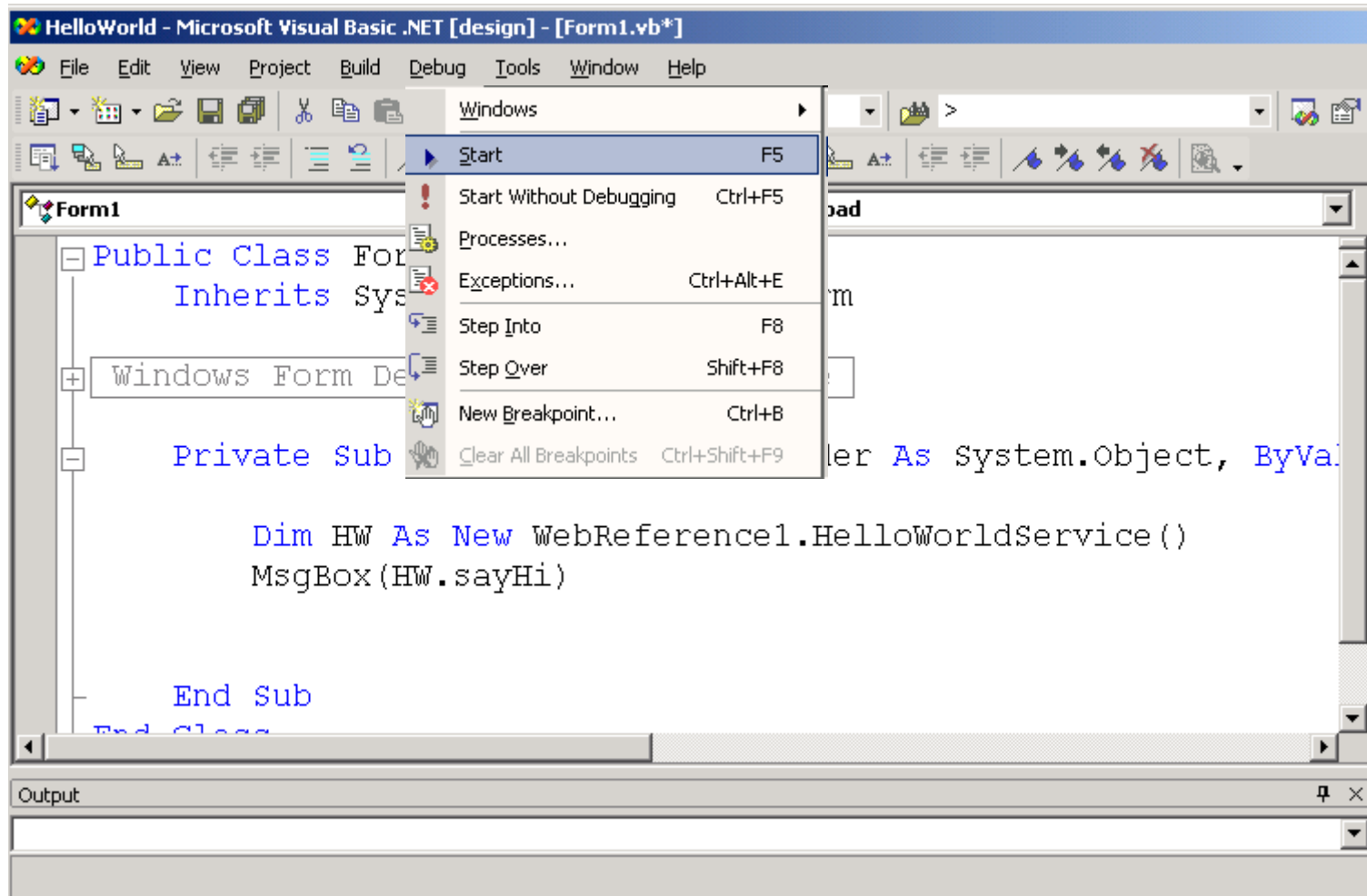
    Windows Form Designer generated code

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim HW As New WebReference1.HelloWorldService()
    End Sub
End Class
```

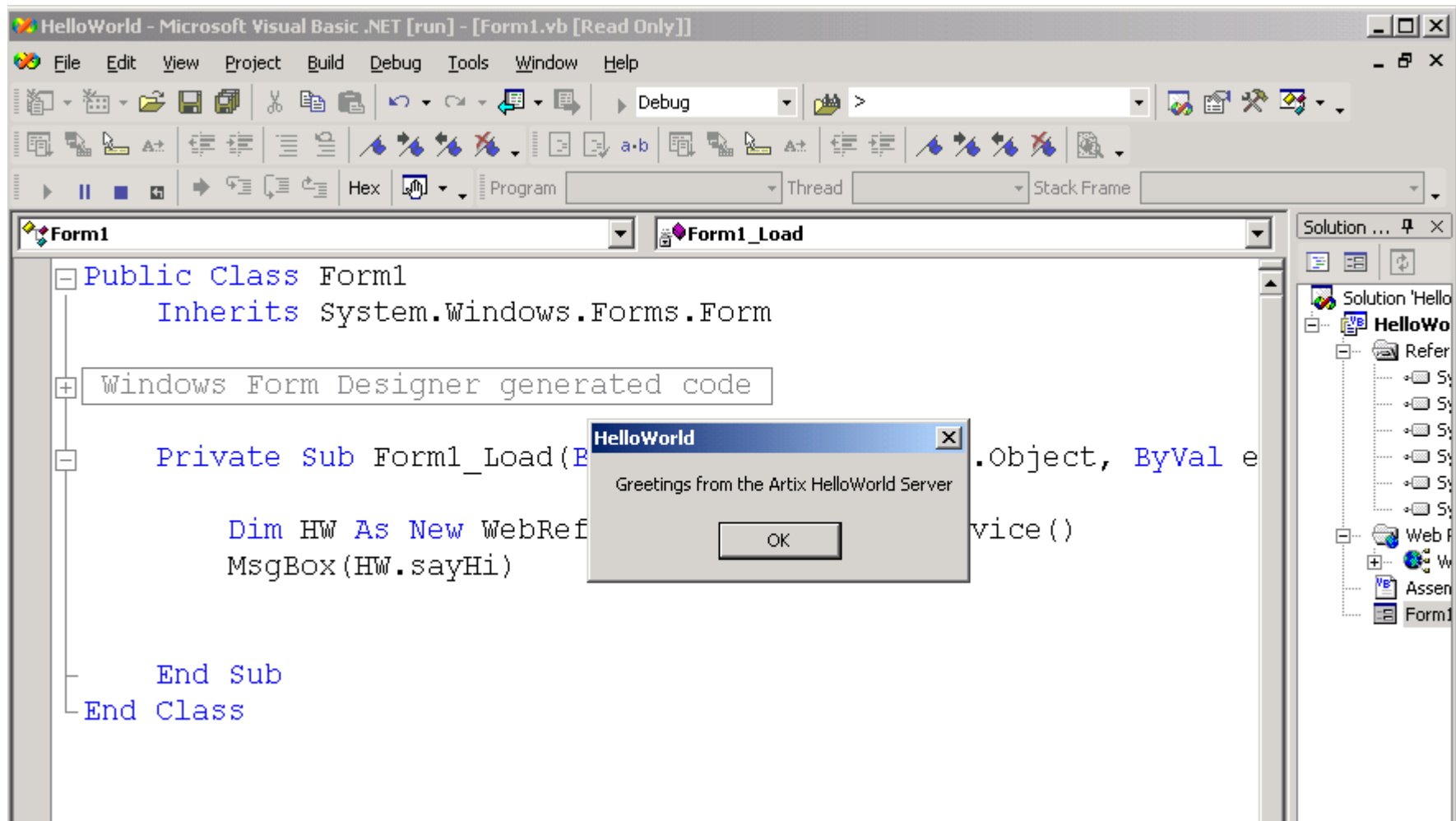
Invoke operation via Proxy Object



Invoke using the Debug menu



Web Service response



The screenshot displays the Microsoft Visual Basic .NET IDE in a 'run' state. The main window shows the code for a class named `Form1` which inherits from `System.Windows.Forms.Form`. The code includes a `Form1_Load` event handler that creates a `WebReference` object and calls `sayHi` on it, with the result displayed in a `MessageBox`.

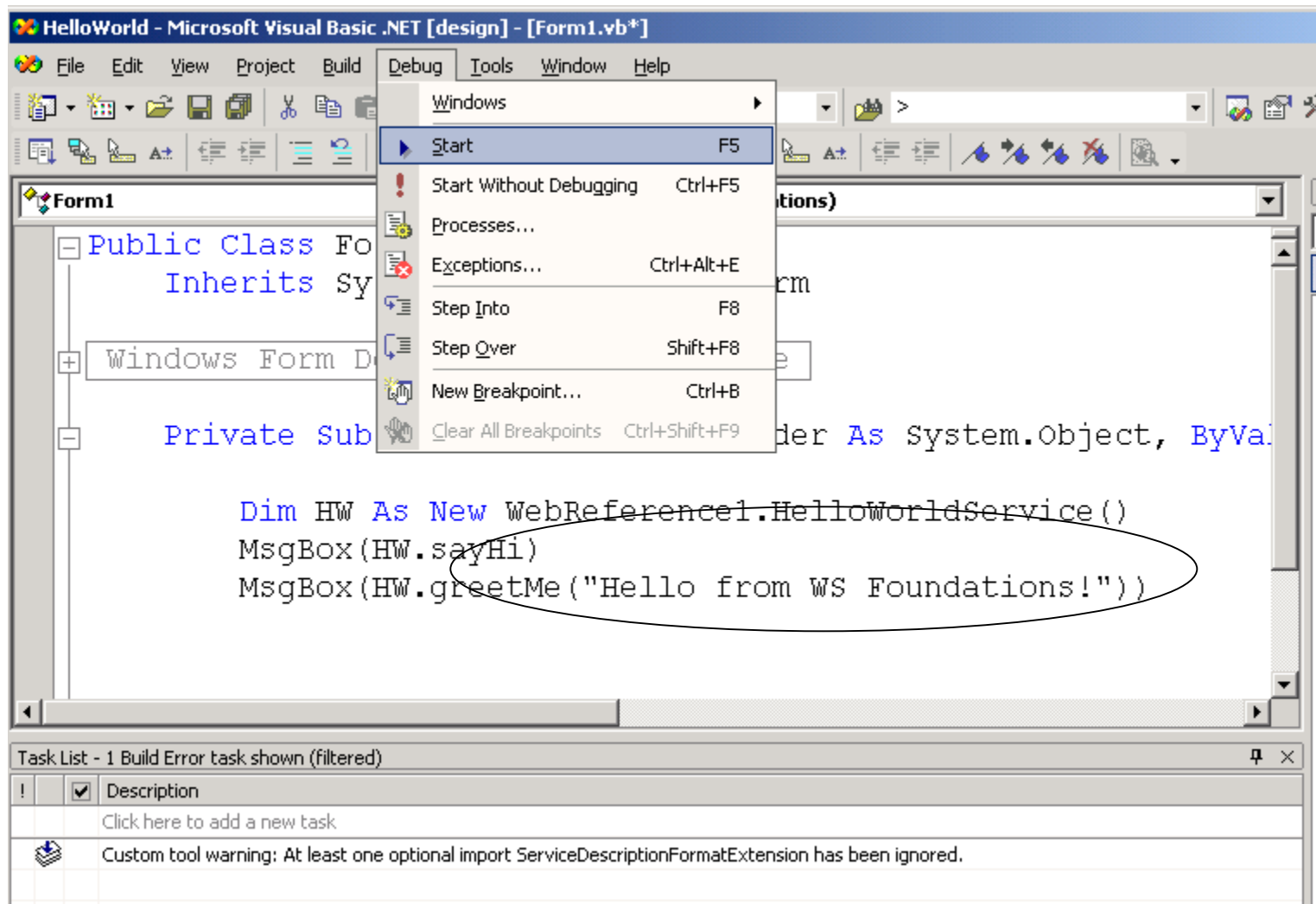
```
Public Class Form1
    Inherits System.Windows.Forms.Form

    Windows Form Designer generated code

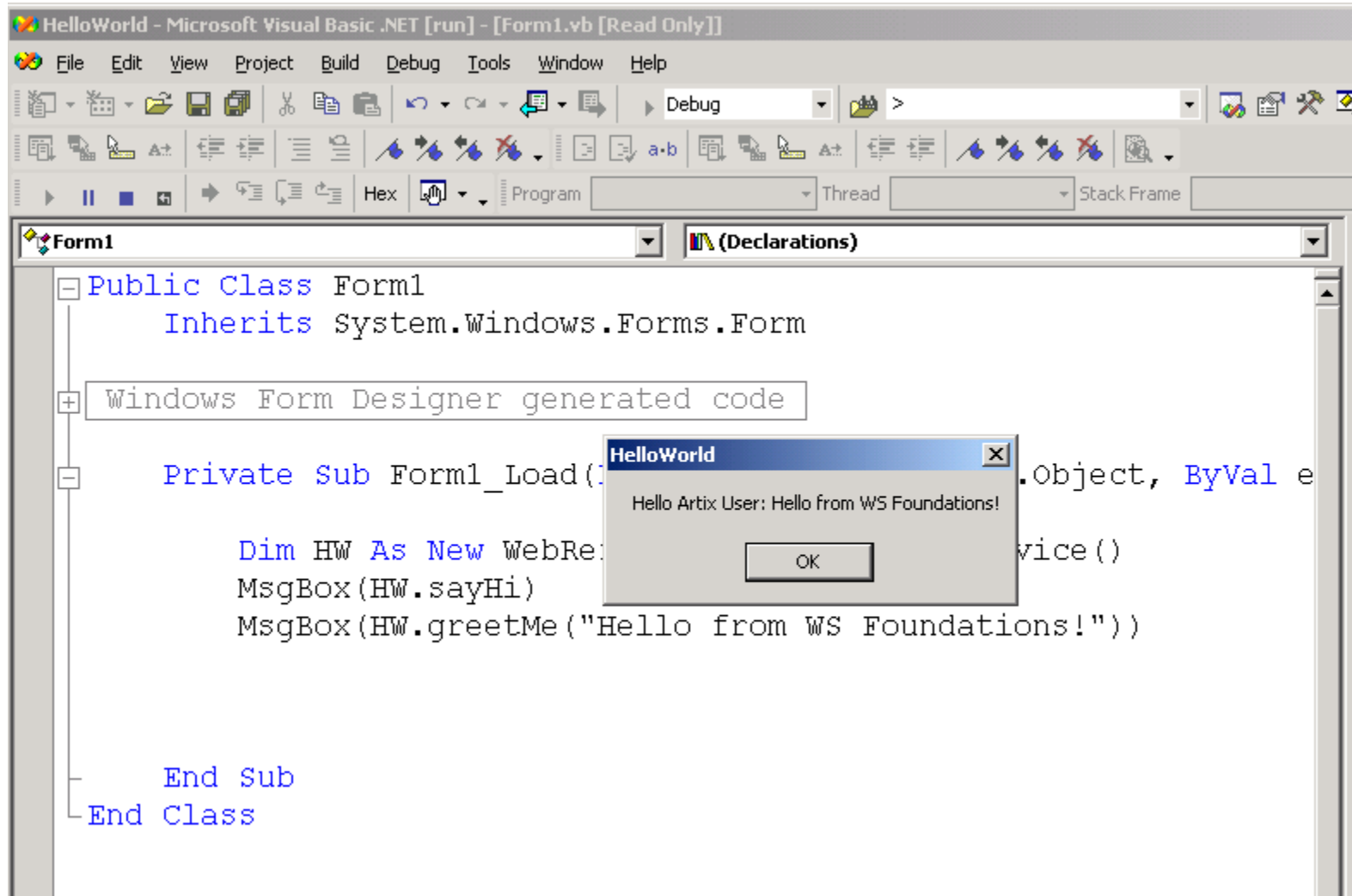
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim HW As New WebReference1
        MsgBox(HW.sayHi)
    End Sub
End Class
```

A dialog box titled "HelloWorld" is overlaid on the code, displaying the message "Greetings from the Artix HelloWorld Server" and an "OK" button. The Solution Explorer on the right shows the project structure, including a `WebReference1` folder and a `Form1` form.

Invoke another operation



Service response



The screenshot shows the Microsoft Visual Basic .NET IDE in the 'run' state. The main window displays the code for a class named 'Form1'. The code includes a 'Private Sub Form1_Load' event handler that creates a new 'WebRe' object, calls 'sayHi', and then calls 'greetMe' with the message 'Hello from WS Foundations!'. A modal message box titled 'HelloWorld' is overlaid on the code, displaying the text 'Hello Artix User: Hello from WS Foundations!' and an 'OK' button.

```
Public Class Form1
    Inherits System.Windows.Forms.Form

    Windows Form Designer generated code

    Private Sub Form1_Load(...)
        Dim HW As New WebRe: ...vice()
        MsgBox(HW.sayHi)
        MsgBox(HW.greetMe("Hello from WS Foundations!"))
    End Sub
End Class
```

Web Service Server Log

```
"Artix Hello World Server"
c:\>REM Artix-init-server.bat

c:\>REM Script file to establish the run time environment for an Artix SSL sample

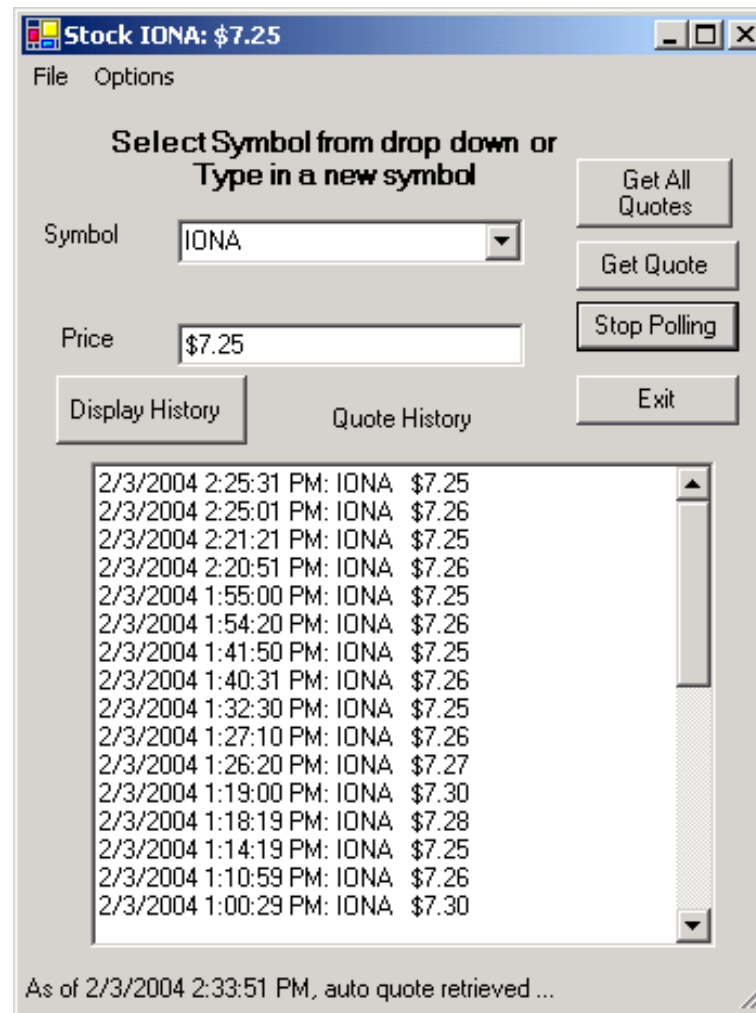
c:\>c:

c:\>set path=c:\openssl-0.9.7b\out32dll;C:\Artix\bin;C:\Artix\artix\1.2\bin;C:\Artix\bin;C:\Artix\artix\1.2\bin;C:\artix\artix\1.2\bin;C:\Perl\bin\;C:\WINNT\system32;C:\WINNT;C:\WINNT\System32\Wbem;C:\Artix\bin;C:\Artix\artix\1.2\bin;C:\Artix\bin;C:\IONA\bin;C:\Program Files\Microsoft Visual Studio\Common\Tools\WinNT;C:\Program Files\Microsoft Visual Studio\Common\MSDev98\Bin;C:\Program Files\Microsoft Visual Studio\Common\Tools;C:\Program Files\Microsoft Visual Studio\VC98\bin;C:\;C:\ERCDTMP;C:\ERCDTMP\BIN;C:\ERCDTMP\TOOLS\PT1

c:\>cd \artix\artix\1.2\bin\

C:\Artix\artix\1.2\bin>call artix_env preserve
Preserving IT_PRODUCT_DIR
Preserving PATH
artix_env complete
HelloWorld Server
HelloWorldImpl::sayHi called
HelloWorldImpl::greetMe called with message: foo
HelloWorldImpl::sayHi called
HelloWorldImpl::greetMe called with message: foo
HelloWorldImpl::sayHi called
HelloWorldImpl::greetMe called with message: Hello from WS Foundations!
```

A VB.NET Client to StockQuote Web Service





Writing Web Service Clients

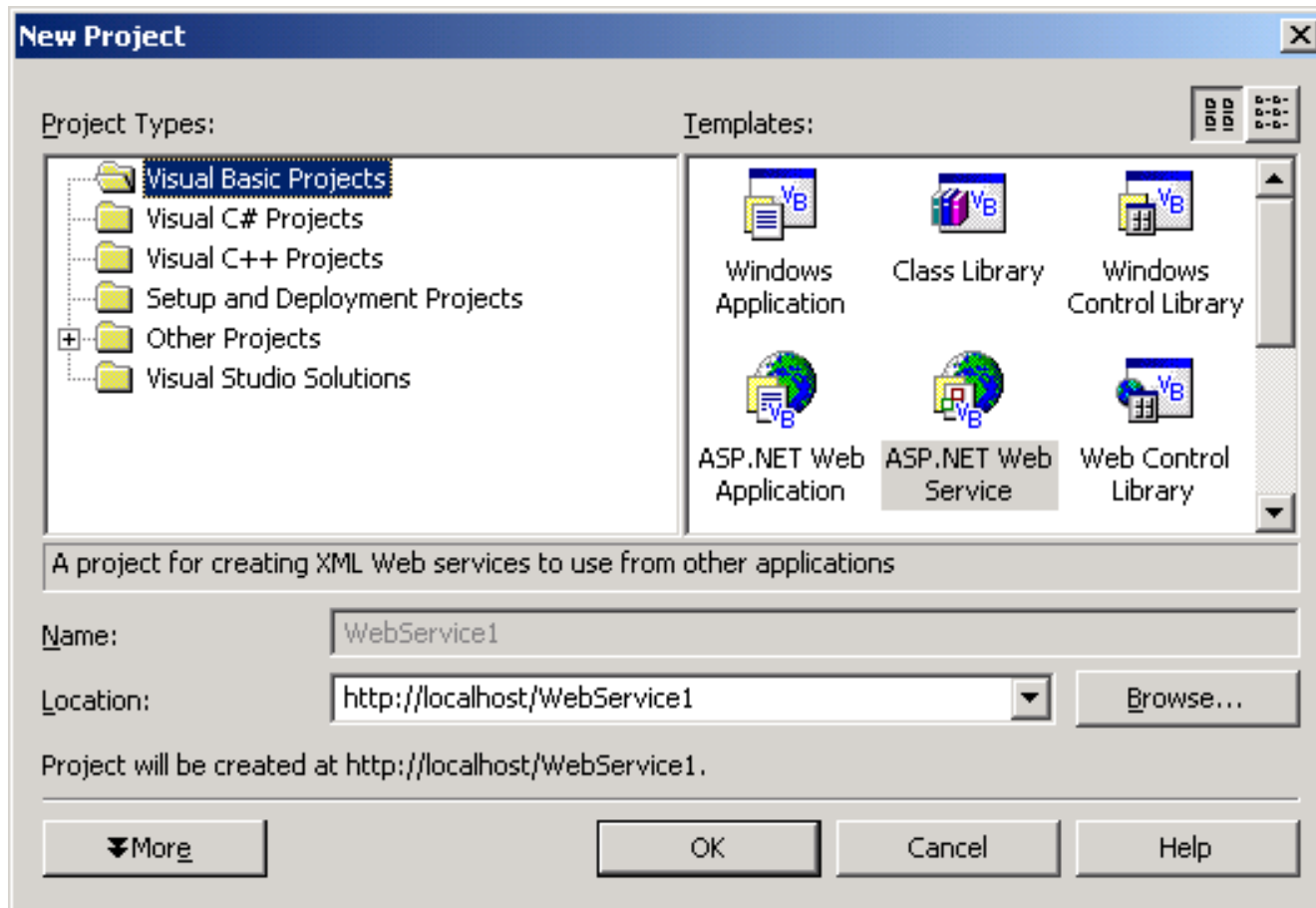
Creating a Web Service from Language Constructs

- Within most Web Service tools, it is easy to expose a function/method as a web service
- With VS.NET, we simply create an ASP.NET Web Service application, and add to any public function declaration(s) the **attribute**:

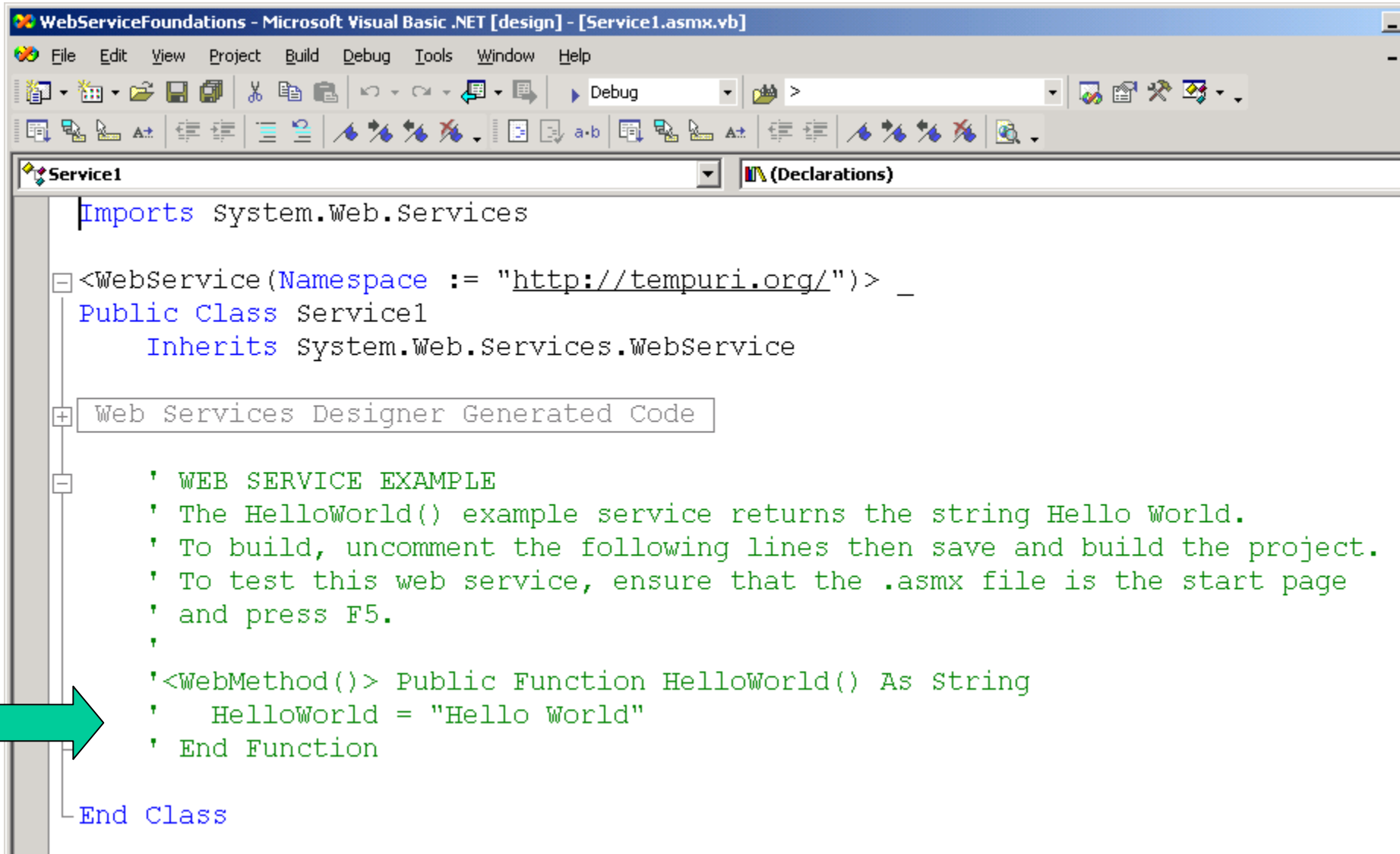
<WebMethod(>

- This allows the .NET framework to supply the necessary WS aspects to expose the method as WS operation and invoke from a remote client
- Let's see simple illustrations of this using VB.NET ...

Create a Web Service Project ...



Generated Source File ...



```
WebServiceFoundations - Microsoft Visual Basic .NET [design] - [Service1.asmx.vb]
File Edit View Project Build Debug Tools Window Help
Imports System.Web.Services

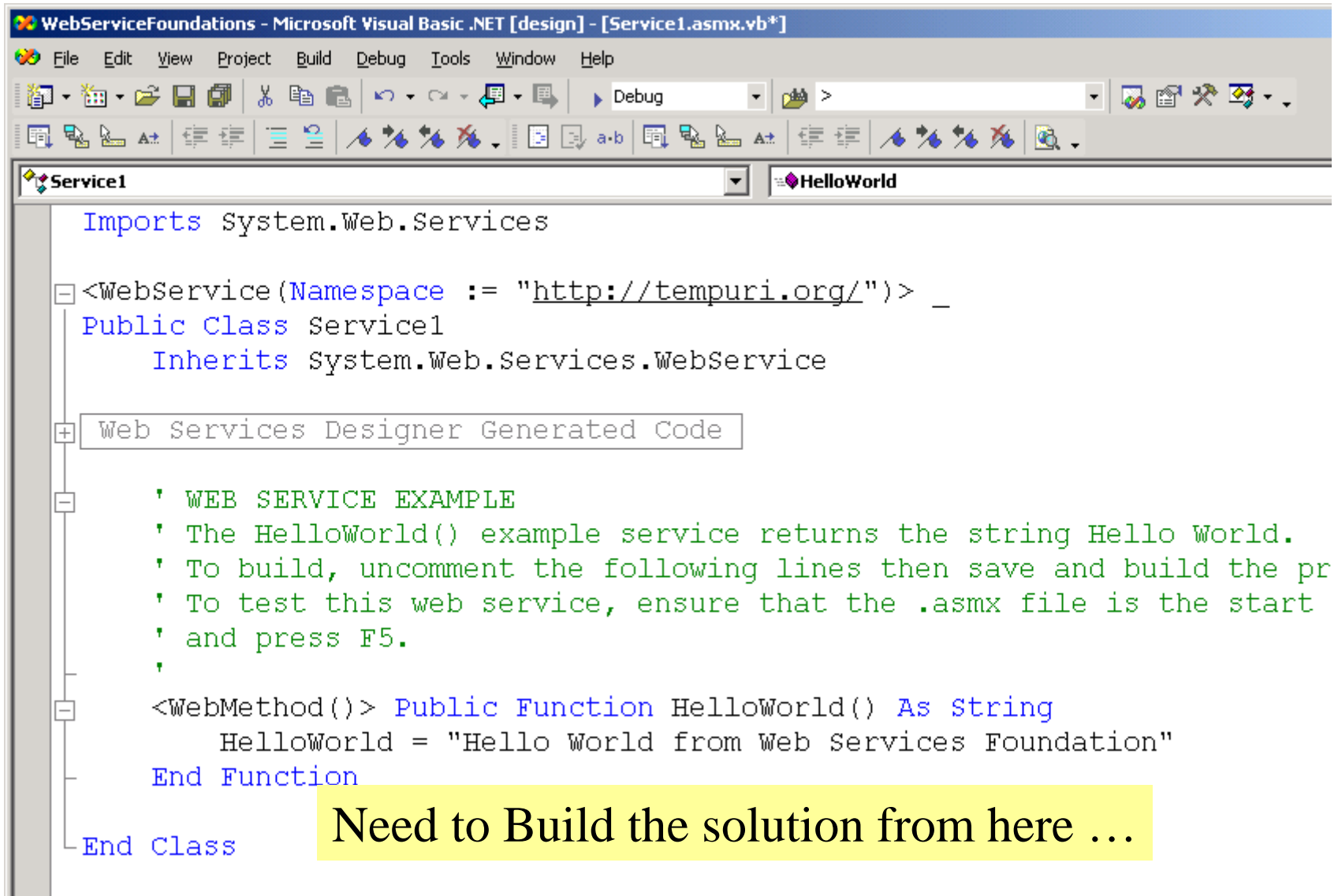
<WebService(Namespace := "http://tempuri.org/")> _
Public Class Service1
    Inherits System.Web.Services.WebService

    Web Services Designer Generated Code

    ' WEB SERVICE EXAMPLE
    ' The HelloWorld() example service returns the string Hello World.
    ' To build, uncomment the following lines then save and build the project.
    ' To test this web service, ensure that the .asmx file is the start page
    ' and press F5.
    '
    '<WebMethod()> Public Function HelloWorld() As String
    '     HelloWorld = "Hello World"
    ' End Function

End Class
```

Add Code by removing comments



WebServiceFoundations - Microsoft Visual Basic .NET [design] - [Service1.aspx.vb*]

File Edit View Project Build Debug Tools Window Help

Debug

Service1 HelloWorld

```
Imports System.Web.Services

<WebService(Namespace := "http://tempuri.org/")> _
Public Class Service1
    Inherits System.Web.Services.WebService

    Web Services Designer Generated Code

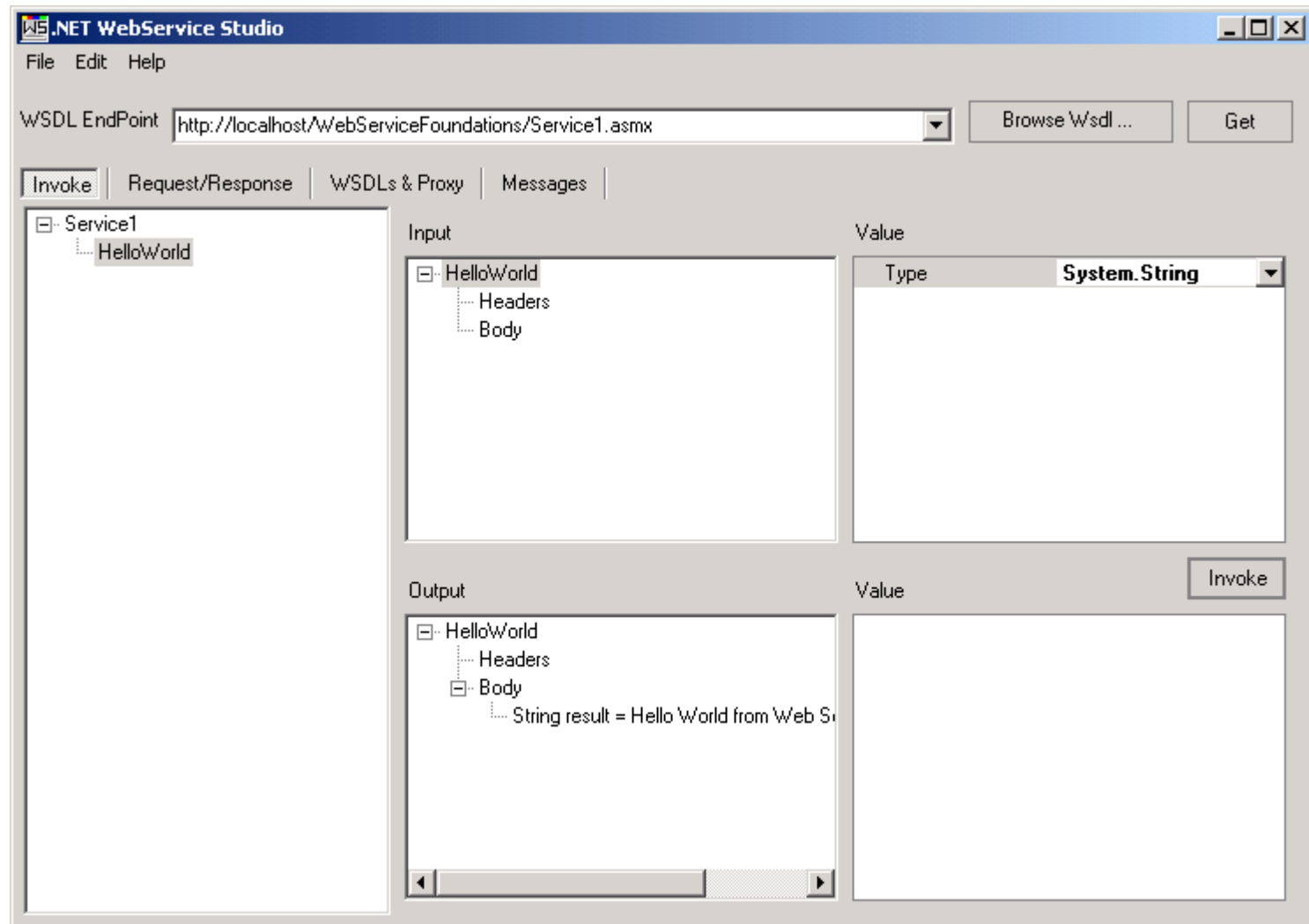
    ' WEB SERVICE EXAMPLE
    ' The HelloWorld() example service returns the string Hello World.
    ' To build, uncomment the following lines then save and build the pr
    ' To test this web service, ensure that the .asmx file is the start
    ' and press F5.
    '

    <WebMethod()> Public Function HelloWorld() As String
        HelloWorld = "Hello World from Web Services Foundation"
    End Function

End Class
```

Need to Build the solution from here ...

Using WS Studio Client – for SOAP



SOAP Messages to/from Server

The screenshot displays the .NET WebService Studio interface. The WSDL EndPoint is set to `http://localhost/WebServiceFoundations/Service1.asmx`. The 'Request/Response' tab is active, showing the SOAP message details.

Request Properties:

Property	Value
AllowAutoRedire	False
AllowWriteStrea	False
BasicAuthPassw	
BasicAuthUserN	
ContentType	text/xml; charset=...
HttpProxy	
KeepAlive	False
Method	POST
Pipelined	False
PreAuthenticate	False
SendChunked	False
SOAPAction	"http://tempuri.org/..."
Timeout	100000
Url	http://localhost/w...
UseCookieCont	True
UseDefaultCred	False

Request SOAP Message:

```
<?xml version="1.0" encoding="utf-16"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <HelloWorld xmlns="http://tempuri.org/" />
  </soap:Body>
</soap:Envelope>
```

Response SOAP Message:

```
Content-Length:392
<?xml version="1.0" encoding="utf-16"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <HelloWorldResponse xmlns="http://tempuri.org/">
      <HelloWorldResult>Hello World from Web Services Foundati...
    </HelloWorldResponse>
  </soap:Body>
</soap:Envelope>
```

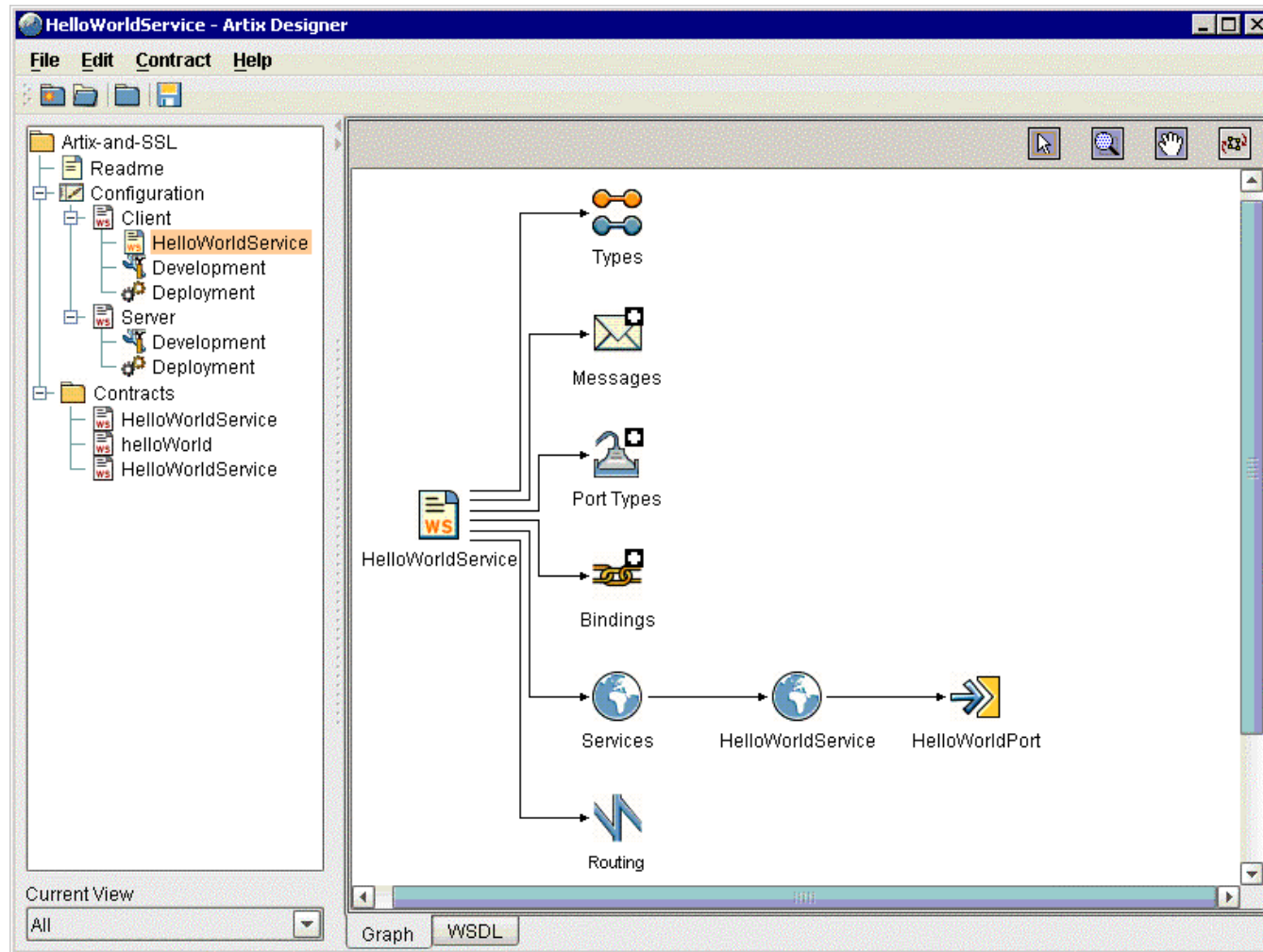
Server Code Generation

- Web Service ToolKits can also generate server side code skeletons directly from WSDL files
 - Java, C++, CORBA and other environments
 - Typically a collection of classes and interfaces
- Then the programmer must add the code to implement the operations (possibly implementing them using existing systems)
- Last step is to deploy the server side implementation into the web service container, to allow it to be invoked by remote clients
- No time to do here, but a later section describes how this was done for a Java-based environment (WSID)

Tools to create WSDL files

- XMLSPY – somewhat generic XML support, but it can be made to work OK for WSDL
- WSDL support embedded in other tool sets
 - Microsoft Visual Studio
 - BEA's WebLogic, etc.
- Visually oriented (GUI) WSDL construction tools becoming available
 - See next few slides for a sample tool (IONA's Artix)

Top level WSDL structure



Detailed WSDL element wizard - Port

Edit Port Properties - Artix Designer

Property Definitions in Port - "HelloWorldPort"

Transport

Transport Type:

Attributes

Address

Attribute	Value
location (REQUIRED)	https://localhost:12345

Client

Attribute	Value
URL	dummy
SendTimeout	
ReceiveTimeout	
AutoRedirect	

Server

Attribute	Value
Port	
SendTimeout	
ReceiveTimeout	
SuppressClientSendErrors	

OK Cancel Apply Help

Summary of Development Demos

- These capabilities are available in most WS toolsets
- Simple to access remote service WSDL files to:
 - Create simplistic clients for immediate access/testing
 - Create simple client in programming language of choice
 - Create stubs for server components
- These tools mean that you don't need to understand the details of WSDL
 - The tools handle the WSDL, and give you stub and skeleton functions that you can use to program your clients and services.

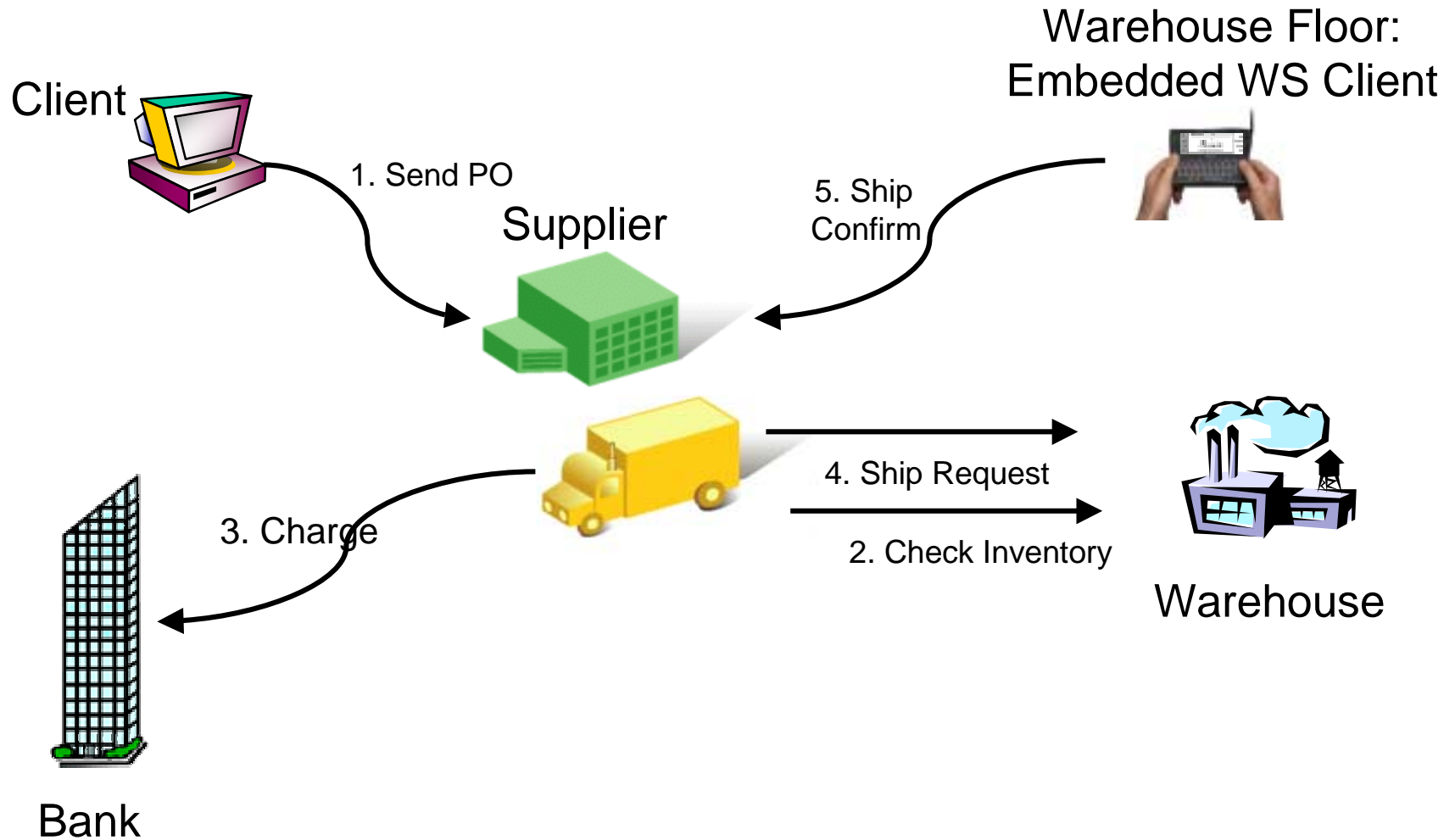
Web Services Execution Demo

Web Services Interoperability Demo (WSID)

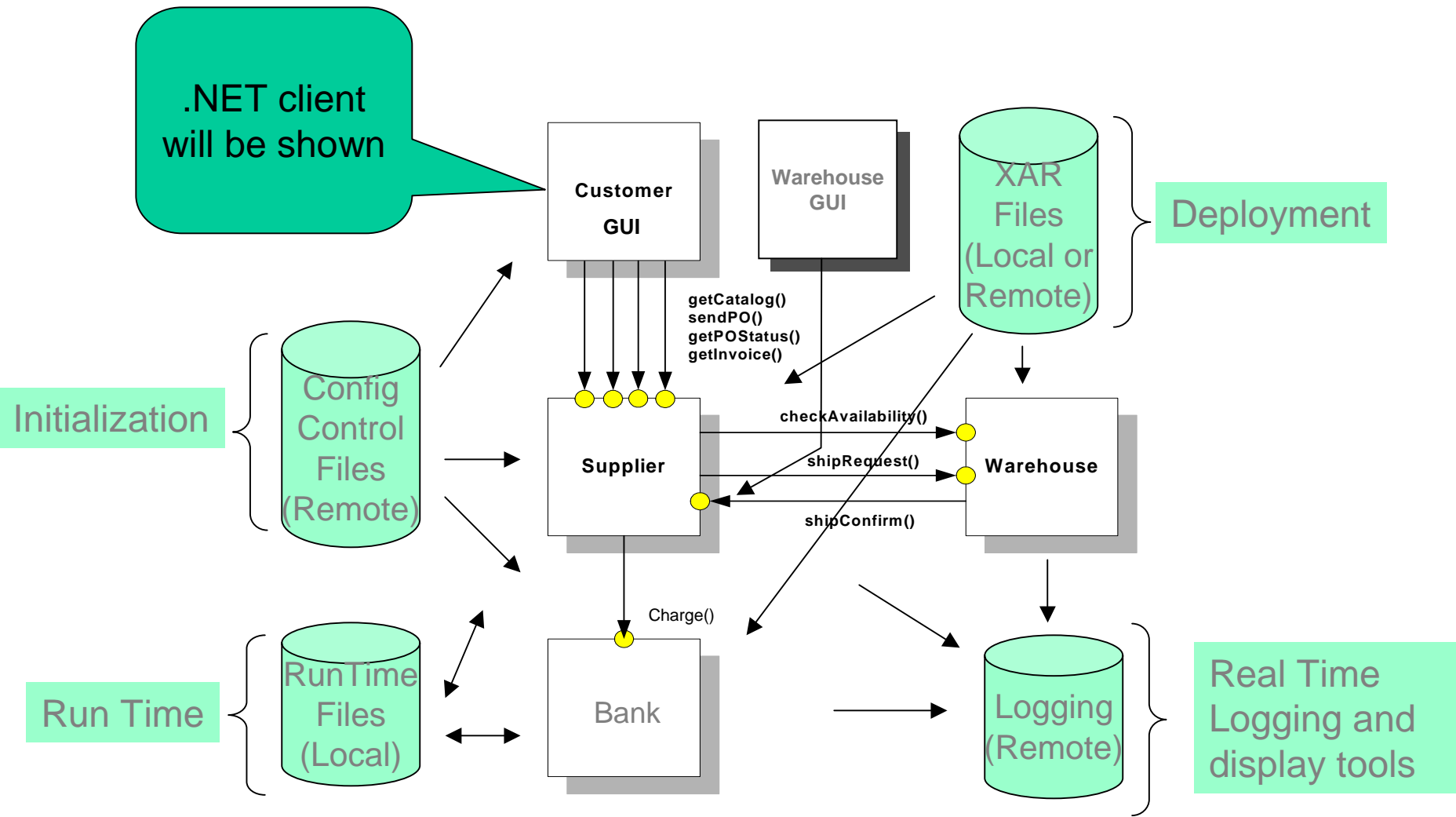
A Web Service Case Study

- Web Services Interoperability Demo (WSID)
- Group of industry vendors (IBM, Microsoft, IONA, The Mind Electric, Amberpoint, WebMethods ...)
- Joint development of specs and implementations
- Development done globally, demonstrated at numerous trade shows (e.g., XML-ONE)
- Multiple platforms, OS, programming languages
 - .NET, several J2EE systems, straight Java, and some unknown!
- WSDL files as the key definition / specification for the project
- Tony Hong of XMETHODS.NET was the technical leader
<http://www.xmethods.net/wsid-03-2003/>

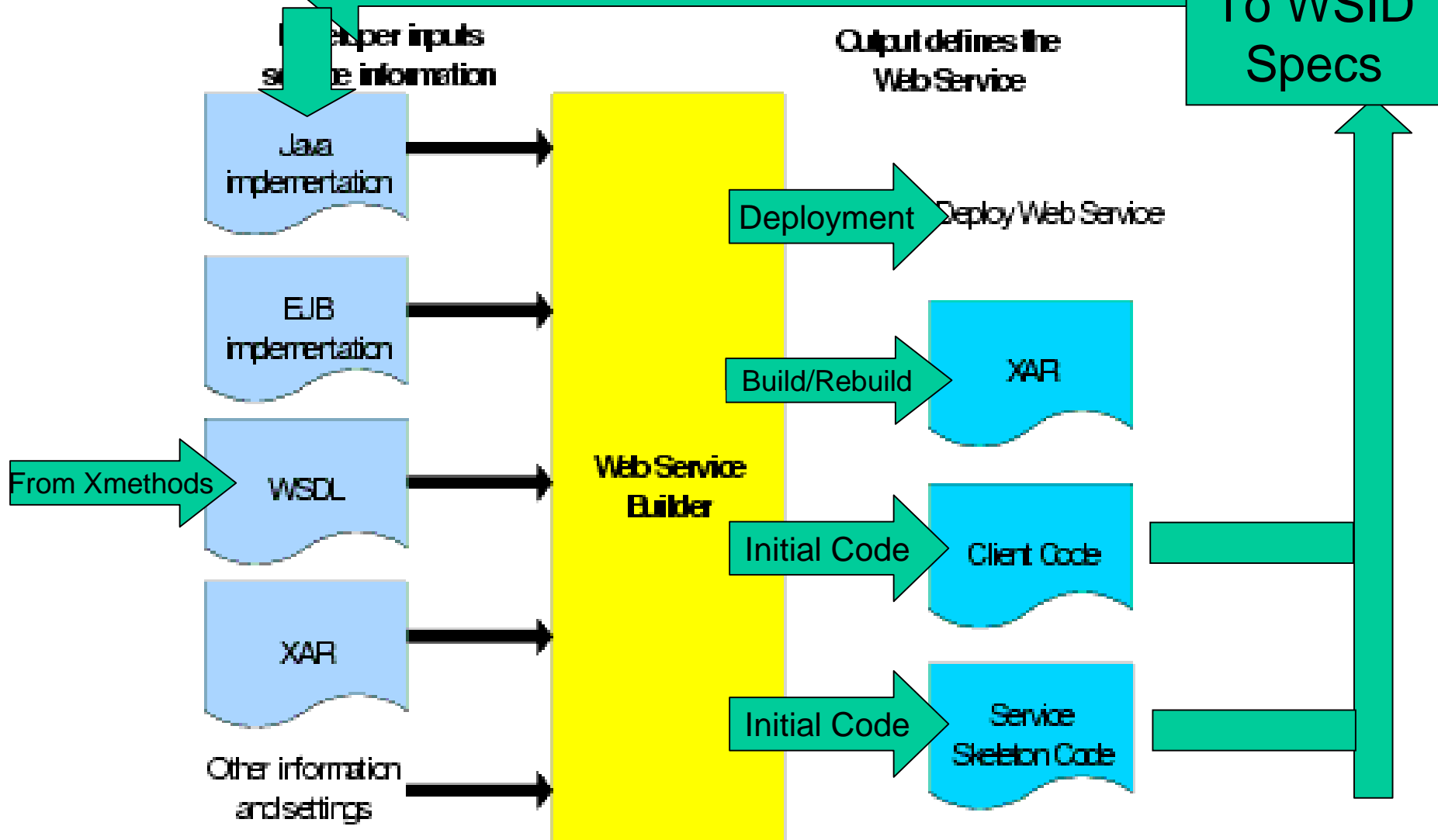
Web Services Interoperability Demo



WSID Architecture



Building Web Service-based Application



.NET (VB) Client for WSID

Artix Web Services Interoperability Demo

EndPoint URL: **Web Service Requests**

Customer ID:

Supplier ID:

PO Number: Result:

Catalog | Purchase Order Form | Invoice

Purchase Order Line Items					
	Item Name	Item Number	Item Measure	Item Price	Item Amount
	Ethernet Cabl	1009	EACH	7.99	10
	Cool Game S	1010	EACH	20	0
	40 GB Disk D	1008	EACH	200	0
	60 GB Hard	1007	EACH	500	0
	Floppy disks	1006	BOX	5	30
	850 MHz Lapt	1005	EACH	999	0
	256 MB Mem	1004	EACH	200	0
▶	512 MB Mem	1003	EACH	600	2
	1.0 GHz CPU	1002	EACH	700	0
	1.4 GHz CPU	1001	EACH	899	0

WSID Order Form

Ship To:

Line 1:

Line 2:

Line 3:

City:

State:

Country:

Postal Code:

Comments:

Send PO Web Service Invocation Completed

Java Swing Client for WSID

Xmethods Demo - Customer Client

File XMethods Demo Help

URL:

GetCatalog Send PO Get PO Status Get Invoice

PONumber: CustomerID: SupplierID:

ShipToCity: ShipToCountry: ShipToLine1:

ShipToLine2: ShipToLine3: Postal Code:

State: Sub Total: No.of Lines:

Product ID	Name	UnitPrice	UOM	Quantity
1009	Ethernet Cable	7.99	EACH	
1010	Cool Game Software	20.00	EACH	3
1008	40 GB Disk Drive	200.00	EACH	
1007	60 GB Hard Drive	500.00	EACH	3
1006	Floppy disks	5.00	BOX	
1005	850 MHz Laptop	999.00	EACH	
1004	256 MB Memory card	200.00	EACH	
1003	512 MB Memory card	200.00	EACH	

Status :ACCEPTED

IONA

Get/Send Clear

Let's see WSID in action ...

Logging screen shot

WSID Log Viewer

File Help

WSID Events

Timestamp	Participant	Role	Event
2004-02-01T13:14:06-08:00	IONA	Customer	Sending PO (PO-487) to supplier endpoint (: http://test01.amberpoint.com/eProxy/ms-supplier
2004-02-01T13:14:07-08:00	Microsoft	Supplier	Processing incoming PO PO-487 from supplier 66.35.200.59
2004-02-01T13:14:07-08:00	Microsoft	Supplier	Checking availability for PO PO-487
2004-02-01T13:14:11-08:00	Microsoft	Warehouse	Processing availability check for PO PO-487 from 66.35.200.59
2004-02-01T13:14:11-08:00	Microsoft	Warehouse	Returning availability indicator: TRUE
2004-02-01T13:14:11-08:00	Microsoft	Supplier	All items in stock: TRUE
2004-02-01T13:14:11-08:00	Microsoft	Supplier	Requesting charge for customer e2a-customer: 2898.31
2004-02-01T13:14:13-08:00	IONA	Bank	Processing charge request for customer: e2a-customer : 2898.31
2004-02-01T13:14:13-08:00	IONA	Bank	Returning charge success indicator: TRUE
2004-02-01T13:14:13-08:00	Microsoft	Supplier	Successful charge: TRUE
2004-02-01T13:14:14-08:00	Microsoft	Supplier	Requesting shipment for PO PO-487
2004-02-01T13:14:14-08:00	Microsoft	Warehouse	Received shipment request for PO PO-487
2004-02-01T13:14:14-08:00	Microsoft	Warehouse	Confirming shipment for PO PO-487
2004-02-01T13:14:14-08:00	Microsoft	Supplier	Received shipment confirmation for PO PO-487
2004-02-01T13:14:14-08:00	Microsoft	Supplier	sendPO worked for PO-487
2004-02-01T13:14:14-08:00	Microsoft	Supplier	Returning status ACCEPTED
2004-02-01T13:14:15-08:00	IONA	Customer	Received PO (PO-487) with status (ACCEPTED) from supplier endpoint (: http://test01.amberp...
2004-02-01T13:15:24-08:00	IONA	Customer	Requesting PO Status for PO (PO-487) from supplier endpoint (: http://test01.amberpoint.com/eP...
2004-02-01T13:15:24-08:00	Microsoft	Supplier	Processing PO status request for PO# PO-487 from supplier 66.35.200.59
2004-02-01T13:15:25-08:00	Microsoft	Supplier	Returning PO status ACCEPTED
2004-02-01T13:15:25-08:00	IONA	Customer	Received PO Status for PO (PO-487) :ACCEPTED
2004-02-01T13:15:28-08:00	IONA	Customer	Requesting Invoice for PO (PO-487) from supplier endpoint (: http://test01.amberpoint.com/eProx...
2004-02-01T13:15:29-08:00	Microsoft	Supplier	Processing invoice request for PO PO-487 from /WSIDWeb/Supplier.asmx
2004-02-01T13:15:29-08:00	IONA	Customer	Error in Requesting Invoice for PO (PO-487) from supplier endpoint (: http://test01.amberpoint.co...
2004-02-01T13:34:35-08:00	IONA	Customer	Requesting PO Status for PO (PO-487) from supplier endpoint (: http://test01.amberpoint.com/eP...
2004-02-01T13:34:35-08:00	Microsoft	Supplier	Processing PO status request for PO# PO-487 from supplier 66.35.200.59
2004-02-01T13:34:36-08:00	Microsoft	Supplier	Returning PO status ACCEPTED
2004-02-01T13:34:36-08:00	IONA	Customer	Received PO Status for PO (PO-487) :ACCEPTED
2004-02-01T13:36:20-08:00	IONA	Customer	Requesting Invoice for PO (PO-487) from supplier endpoint (: http://test01.amberpoint.com/eProx...
2004-02-01T13:36:21-08:00	Microsoft	Supplier	Processing invoice request for PO PO-487 from /WSIDWeb/Supplier.asmx
2004-02-01T13:36:22-08:00	IONA	Customer	Error in Requesting Invoice for PO (PO-487) from supplier endpoint (: http://test01.amberpoint.co...
2004-02-01T13:37:10-08:00	IONA	Supplier	Processing invoice request for PO: e2a-customer_2/1/2004 4:07:03 PM
2004-02-01T13:37:10-08:00	IONA	Supplier	Returning invoice for PO: e2a-customer_2/1/2004 4:07:03 PM

Clear Table

Browser view of logs

The screenshot shows a Microsoft Internet Explorer browser window displaying a log file at the URL `http://www.xmethods.net/idemo/log.cgi`. The browser's address bar and menu bar are visible at the top. On the left side, there is a Favorites sidebar with a list of folders including 'App Development', 'Bioinformatics', 'Business', 'Computer Supplies', 'e-Commerce', 'Holy Cross', 'IONA - Artix', 'IONA - General', 'Media', 'Misc', 'Palm Sites', 'Personal', 'Sports Sites', 'WSID', 'WS-I', and 'Web Service Topics'. The main content area displays a log with the following entries:

2004-02-01T13:13:27	IONA	Supplier	Processing catalog request from e2a-custom
2004-02-01T13:13:27	IONA	Supplier	Returning catalog
2004-02-01T13:13:27	IONA	Customer	Received Catalog from supplier endpoint:
2004-02-01T13:13:41	IONA	Customer	Requesting Catalog Request from e2a-custom
2004-02-01T13:13:42	Microsoft	Supplier	Processing catalog request from 66
2004-02-01T13:13:42	Microsoft	Supplier	Returning catalog
2004-02-01T13:13:42	IONA	Customer	Received Catalog from supplier endpoint:
2004-02-01T13:14:07	IONA	Customer	Sending PO (PO-487) to supplier endpoint (
2004-02-01T13:14:07	Microsoft	Supplier	Processing incoming PO PO-487 from
2004-02-01T13:14:07	Microsoft	Supplier	Checking availability for PO PO-
2004-02-01T13:14:11	Microsoft	Warehouse	Processing availability check fo

The status bar at the bottom of the browser window shows 'Done' on the left and 'Internet' on the right.

WSID Demo Summary

- What we saw in this demo ...
- Multiple interacting web services
- Implementing a typical Supply Chain Example
- Multiple Language GUI Clients
- Web Service server also acting a Web Service client
- Each of these components has been implemented by a set of WS vendors
 - And it has been tested with arbitrary selections of these components, running on machines distributed across the US!

Chapter Summary

- Web Services are real – here and now!
 - You can design, develop and deploy with them now
 - Both for internal use and between enterprise vendors
 - Existing services can be discovered and accessed easily
- Many tools available with varying degrees of maturity and complexity – and more to come
 - Evolution of tool capabilities ultimately drives ease of use of WS
- The most interesting part of designing a system with Web Services is the high-level architecture of the system
 - What services should exist, and how should they interact?
 - What is the right structure for the information that should pass between clients and services?
 - These are the topics of the next chapter