Chapter 8

Interface-Based Web Service Development

The goal of education is the advancement of knowledge and the dissemination of truth. —John F. Kennedy

Interface-based programming was popularized with component-based development in the 1990s. Using technologies like COM, you could define an interface and have several components implement it. Clients could utilize any of those components by programming against the interface. As your Web services evolve and mature, you will find it necessary to factor out Web service methods into interfaces, implement existing standard interfaces on your Web services, and program clients against an interface rather than a specific Web service. Interfaces can also be useful for versioning Web services by leaving the old interface intact and implementing a new interface on the same service.

WSDL bindings make this possible. In Chapter 4 you learned about WSDL bindings and how they define a concrete set of operations and provide the information needed to invoke those operations. A Web service implements one or more bindings and exposes them at a particular location defined by the port. Even if you haven't read Chapter 4, you can read this chapter and learn how to do interface-based programming. However, you will gain much more from this chapter if you read Chapter 4 first.

8.1 Defining Interfaces

The first step in interface-based programming is to define the interfaces you want to implement. When you build a Web service, you should always start with defining the interface. Today, tools like Visual Studio .NET do not provide direct support for this. I am hopeful that future versions will provide the needed support for defining Web service interfaces.

Although you can use Notepad to create a WSDL document from scratch, you'll probably want a more productive and less error-prone way to define your interfaces. An easy way to define a Web service interface is to create a Web service and define the Web methods you want the interface to have. If you have parameters with complex types, you define those types in schemas, then use xsd.exe to generate classes from the schemas (see Chapter 2 for more information on xsd.exe).

By default, all of a Web service's methods belong to the same binding. That binding (interface) has the same name as the Web service class with the word Soap appended. If you've created COM components in Visual Basic, you may know that each component you create has a default interface that is given the name *_ClassName.* Therefore, the concept of auto-generated interfaces shouldn't be new to you.

To control the binding's name and namespace, you use the `WebServiceBinding` attribute on the Web service class to specify that binding's name and namespace. On each Web method that the service exposes, you add `SoapDocumentMethod` or `SoapRpcMethod` and set its `Binding` property to the binding name. Listing 8.1 shows an example class called `SupplierIface1` that exposes its methods in a binding called `ISupplier`.

Listing 8.1 A Web service example that exposes a binding called ISupplier (VBWSBook\Chapter8\Supplier1.asmx.vb)

```
Namespace Supplier1
    Public Structure Order
        Public CustomerEmail As String
        Public ShipVia As Shipper
        Public ShipName As String
        Public ShipAddress As String
    Public ShipCity As String
    Public ShipState As String
    Public ShipZipCode As String
    Public OrderItems() As OrderItem 'array of OrderItems
End Structure
Public Structure OrderItem
    Public ProductID As Integer
    Public Quantity As Integer
End Structure
Public Enum Shipper
    FedEx = 1
    UPS
    USPS
End Enum
Public Enum OrderStatus
    Pending
    Shipped
    Delivered
End Enum
Public Structure OrderInfo
    Public Status As OrderStatus
    Public ShippingType As String
    Public DeliveredDate As Date
    Public DeliveredTo As String
End Structure
Public Structure QuoteInfo
    Public ProductCost As Double
    Public Tax As Double
    Public Shipping As Double
    Public TotalCost As Double
End Structure
<WebServiceBinding( _
Name:="ISupplier", _
[Namespace]:="http://LearnXmlWS.com/Supplier"), _
WebService([Namespace]:="http://LearnXmlWS.com/Supplier", _
 Description:="The supplier's Web service")> _
Public Class SupplierIface1
    Inherits System.Web.Services.WebService
    <WebMethod( _
        Description:= _
        "Places the order then returns the new order id"), _
    SoapDocumentMethod(Binding:="ISupplier")> _
    Public Function PlaceOrder(ByVal newOrder As Order) _
                               As String
```

```
            'returns a new order id
        End Function
        <WebMethod(), _
        SoapDocumentMethod(Binding:="ISupplier")> _
        Public Function CheckStatus(ByVal OrderId As String) _
                                    As OrderInfo
            'returns an orderinfo structure
        End Function
        <WebMethod(), _
        SoapDocumentMethod(Binding:="ISupplier")> _
        Public Function GetPriceQuote(ByVal newOrder As Order)
                                      As QuoteInfo
            'returns an orderinfo structure
        End Function

    End Class
End Namespace
```

The first part of Listing 8.1 defines the data types that will be used by the service methods (for example, `Order`, `OrderItem`, `Shipper`, `OrderStatus`, `OrderInfo`, and `QuoteInfo`). The `SupplierIface1` class has two attributes applied to it. `WebServiceBinding` has its name property set to `ISupplier` and its namespace property set to `http://LearnXmlWS.com/Supplier`. Each of the Web service methods has a `SoapDocumentMethod` applied to it with the `Binding` property set to `ISupplier`, making the methods part of the interface called `ISupplier`. The resulting WSDL document contains the binding definition and the service definition.

To get a pure interface definition, you can save this WSDL document to disk and remove the `<service>` element, which specifies a particular implementation for the interface. The edited WSDL document, shown in Listing 8.2 now contains your interface definition, which you can give to other developers who can use it to implement the same binding (interface) on their services.

Listing 8.2 The interface WSDL after removing the `<service>` element (VBWSBook\Chapter8\SingleInterface.wsdl)

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
```

```
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:s0="http://LearnXmlWS.com/Supplier"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://LearnXmlWS.com/Supplier"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <s:schema elementFormDefault="qualified"
        targetNamespace="http://LearnXmlWS.com/Supplier">
      <s:element name="PlaceOrder">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1"
                       name="newOrder" type="s0:Order" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="Order">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1"
                     name="CustomerEmail" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1"
                     name="ShipVia" type="s0:Shipper" />
          <s:element minOccurs="0" maxOccurs="1"
                     name="ShipName" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1"
                     name="ShipAddress" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1"
                     name="ShipCity" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1"
                     name="ShipState" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1"
                     name="ShipZipCode" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1"
                     name="OrderItems"
                     type="s0:ArrayOfOrderItem" />
        </s:sequence>
      </s:complexType>
      <s:simpleType name="Shipper">
        <s:restriction base="s:string">
          <s:enumeration value="FedEx" />
          <s:enumeration value="UPS" />
    <s:enumeration value="USPS" />
  </s:restriction>
</s:simpleType>
<s:complexType name="ArrayOfOrderItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
               name="OrderItem" type="s0:OrderItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="OrderItem">
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1"
               name="ProductID" type="s:int" />
    <s:element minOccurs="1" maxOccurs="1"
               name="Quantity" type="s:int" />
  </s:sequence>
</s:complexType>
<s:element name="PlaceOrderResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
                 name="PlaceOrderResult" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="CheckStatus">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
                 name="OrderId" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="CheckStatusResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1"
                 name="CheckStatusResult"
                 type="s0:OrderInfo" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:complexType name="OrderInfo">
```

```
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1"
               name="Status" type="s0:OrderStatus" />
    <s:element minOccurs="0" maxOccurs="1"
               name="ShippingType" type="s:string" />
    <s:element minOccurs="1" maxOccurs="1"
               name="DeliveredDate" type="s:dateTime" />
    <s:element minOccurs="0" maxOccurs="1"
               name="DeliveredTo" type="s:string" />
  </s:sequence>
</s:complexType>
<s:simpleType name="OrderStatus">
  <s:restriction base="s:string">
    <s:enumeration value="Pending" />
    <s:enumeration value="Shipped" />
    <s:enumeration value="Delivered" />
  </s:restriction>
</s:simpleType>
<s:element name="GetPriceQuote">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="newOrder"
                 type="s0:Order" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="GetPriceQuoteResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1"
                 name="GetPriceQuoteResult"
                 type="s0:QuoteInfo" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:complexType name="QuoteInfo">
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1"
               name="ProductCost" type="s:double" />
    <s:element minOccurs="1" maxOccurs="1"
               name="Tax" type="s:double" />
    <s:element minOccurs="1" maxOccurs="1"
               name="Shipping" type="s:double" />
```

```
        <s:element minOccurs="1" maxOccurs="1"
                   name="TotalCost" type="s:double" />
      </s:sequence>
    </s:complexType>
  </s:schema>
</types>
<message name="PlaceOrderSoapIn">
  <part name="parameters" element="s0:PlaceOrder" />
</message>
<message name="PlaceOrderSoapOut">
  <part name="parameters" element="s0:PlaceOrderResponse" />
</message>
<message name="CheckStatusSoapIn">
  <part name="parameters" element="s0:CheckStatus" />
</message>
<message name="CheckStatusSoapOut">
  <part name="parameters" element="s0:CheckStatusResponse" />
</message>
<message name="GetPriceQuoteSoapIn">
  <part name="parameters" element="s0:GetPriceQuote" />
</message>
<message name="GetPriceQuoteSoapOut">
  <part name="parameters" element="s0:GetPriceQuoteResponse" />
</message>
<portType name="ISupplier">
  <operation name="PlaceOrder">
    <documentation>
      Places the order then returns the new order id
    </documentation>
    <input message="s0:PlaceOrderSoapIn" />
    <output message="s0:PlaceOrderSoapOut" />
  </operation>
  <operation name="CheckStatus">
    <input message="s0:CheckStatusSoapIn" />
    <output message="s0:CheckStatusSoapOut" />
  </operation>
  <operation name="GetPriceQuote">
    <input message="s0:GetPriceQuoteSoapIn" />
    <output message="s0:GetPriceQuoteSoapOut" />
  </operation>
</portType>
<binding name="ISupplier" type="s0:ISupplier">
```

```
      <soap:binding
        transport=
        "http://schemas.xmlsoap.org/soap/http" style="document" />
      <operation name="PlaceOrder">
        <soap:operation
          soapAction="http://LearnXmlWS.com/Supplier/PlaceOrder"
          style="document" />
        <input>
          <soap:body use="literal" />
        </input>
        <output>
          <soap:body use="literal" />
        </output>
      </operation>
      <operation name="CheckStatus">
        <soap:operation
          soapAction="http://LearnXmlWS.com/Supplier/CheckStatus"
          style="document" />
        <input>
          <soap:body use="literal" />
        </input>
        <output>
          <soap:body use="literal" />
        </output>
      </operation>
      <operation name="GetPriceQuote">
        <soap:operation
            soapAction=
            "http://LearnXmlWS.com/Supplier/GetPriceQuote"
            style="document" />
        <input>
          <soap:body use="literal" />
        </input>
        <output>
          <soap:body use="literal" />
        </output>
      </operation>
    </binding>
  </definitions>
```

8.2 Implementing an Interface

Whether you define the interfaces yourself or you work with interfaces defined by someone else, you'll eventually want to implement them. Although you can read the WSDL document and write all the corresponding VB code from scratch, including the `WebServiceBinding` attribute, something tells me you're not going to want to do this. Instead, you can use wsdl.exe with the /server switch to tell it you want to create a service that implements the specified interface. wsdl.exe takes the WSDL document's URL, the language to use for generated code, and the output file name:

```
wsdl.exe /server http://VBWSServer/vbwsbook/Chapter8/Single-
➥Interface.wsdl /l:VB /out:CSupplier.vb
```

Listing 8.3 shows the interesting part of the resulting code in CSupplier.vb.

Listing 8.3   A Web service implementation generated by wsdl.exe when using /server switch (VBWSBook\Chapter8\InterfaceImpl\CSupplier.vb)

```
'
'This source code was auto-generated by wsdl
'
<WebServiceBindingAttribute(Name:="ISupplier", _
[Namespace]:="http://LearnXmlWS.com/Supplier")> _
Public MustInherit Class ISupplier
    Inherits WebService

    <WebMethodAttribute(), _
     SoapDocumentMethodAttribute( _
     "http://LearnXmlWS.com/Supplier/PlaceOrder", _
     RequestNamespace:="http://LearnXmlWS.com/Supplier", _
     ResponseNamespace:="http://LearnXmlWS.com/Supplier", _
     Use:=Description.SoapBindingUse.Literal, _
     ParameterStyle:=SoapParameterStyle.Wrapped)> _
    Public MustOverride Function PlaceOrder( _
        ByVal newOrder As Order) As String

    <WebMethodAttribute(), _
     SoapDocumentMethodAttribute( _
       "http://LearnXmlWS.com/Supplier/CheckStatus", _
        RequestNamespace:="http://LearnXmlWS.com/Supplier", _
        ResponseNamespace:="http://LearnXmlWS.com/Supplier", _
        Use:=Description.SoapBindingUse.Literal, _
        ParameterStyle:=SoapParameterStyle.Wrapped)> _
    Public MustOverride Function CheckStatus( _
                             ByVal OrderId As String) As _
```

```
            <XmlElementAttribute(IsNullable:=False)> OrderInfo

        <WebMethodAttribute(), _
         SoapDocumentMethodAttribute( _
            "http://LearnXmlWS.com/Supplier/GetPriceQuote", _
            RequestNamespace:="http://LearnXmlWS.com/Supplier", _
            ResponseNamespace:="http://LearnXmlWS.com/Supplier", _
            Use:=Description.SoapBindingUse.Literal, _
            ParameterStyle:=SoapParameterStyle.Wrapped)> _
        Public MustOverride Function GetPriceQuote( _
            ByVal newOrder As Order) As QuoteInfo
    End Class
```

Note that the class name is by default the same as the binding name, that is, `ISupplier`. You'll also see a `WebServiceBinding` attribute applied to `ISupplier` to set the binding's name and namespace. Each method has a `SoapDocumentMethod` attribute that specifies things like the request and response namespaces and the fact that message parts are literal and wrapped.

Notice also that the class is abstract (MustInherit). While you can easily put implementation code in the class itself, it is generally a good idea to put implementation code in a class that inherits from it. This way you will not be confused about which methods are part of the original interface you are implementing and which ones you added yourself. Also, keeping the interface methods in a separate class means there's less chance that you'll accidentally modify one or more of the interface methods as you are implementing the Web service.

Listing 8.4 shows an example Web service that implements the ISupplier interface by inheriting from the ISupplier abstract class that wsdl.exe generated.

Listing 8.4 An example Web service that implements the ISupplier interface (VBWSBook\Chapter8\InterfaceImpl\SingleInterfaceImple.asmx.vb)

```
Imports System.Web.Services
Imports System.Web.Services.Protocols

<WebServiceBinding( _
    Name:="ISupplier", _
    [Namespace]:="http://LearnXmlWS.com/Supplier", _
     Location:= _
"http://vbwsserver/vbwsbook/chapter8/SingleInterface.wsdl"), _
    WebService(Namespace:="somenamespace")> _
```

```
Public Class SingleInterfaceImpl
    Inherits ISupplier

    <WebMethodAttribute(), _
    SoapDocumentMethodAttribute( _
    "http://LearnXmlWS.com/Supplier/CheckStatus", _
    RequestNamespace:="http://LearnXmlWS.com/Supplier", _
    ResponseNamespace:="http://LearnXmlWS.com/Supplier", _
    Use:=Description.SoapBindingUse.Literal, _
    ParameterStyle:=Protocols.SoapParameterStyle.Wrapped, _
    Binding:="ISupplier")> _
    Public Overrides Function CheckStatus( _
               ByVal OrderId As String) As OrderInfo

    End Function

    <WebMethodAttribute(), _
    SoapDocumentMethodAttribute( _
    "http://LearnXmlWS.com/Supplier/GetPriceQuote", _
    RequestNamespace:="http://LearnXmlWS.com/Supplier", _
    ResponseNamespace:="http://LearnXmlWS.com/Supplier", _
    Use:=Description.SoapBindingUse.Literal, _
    ParameterStyle:=Protocols.SoapParameterStyle.Wrapped, _
    Binding:="ISupplier")> _
    Public Overrides Function GetPriceQuote( _
               ByVal newOrder As Order) As QuoteInfo

    End Function

    <WebMethodAttribute(), _
    SoapDocumentMethodAttribute( _
        "http://LearnXmlWS.com/Supplier/PlaceOrder", _
        RequestNamespace:="http://LearnXmlWS.com/Supplier", _
        ResponseNamespace:="http://LearnXmlWS.com/Supplier", _
        Use:=Description.SoapBindingUse.Literal, _
        ParameterStyle:=Protocols.SoapParameterStyle.Wrapped, _
        Binding:="ISupplier")> _
        Public Overrides Function PlaceOrder( _
                   ByVal newOrder As Order) As String

    End Function
End Class
```

The code in Listing 8.4 is part of a Web project called InterfaceImpl. In this project, you'll find the CSupplier.vb file that was generated by wsdl.exe. The Web service class in Listing 8.4 inherits from ISupplier (which is defined in CSupplier.vb). To implement the Web service interface as defined by ISupplier, you

- Add a WebServiceBinding attribute on your Web service class (the class name is SingleInterfaceImpl in Listing 8.4), set the WebServiceBinding's Name property to ISupplier, and set its Location property to the URL of the interface WSDL. This way you specify that the interface definition should be imported from that URL rather than duplicated in your Web service's WSDL.
- Override each of the ISupplier class methods: CheckStatus, GetQuote and PlaceOrder.
- Set the Binding property of the SoapDocumentMethod attribute to ISupplier on each of these methods. Here you're specifying that each of these methods belongs to the ISupplier interface.

Once you perform these steps and compile your Web service, the resulting WSDL will look like the one in Listing 8.5.

Listing 8.5   WSDL for a Web service that implements the ISupplier interface. This WSDL imports the ISupplier definitions from SingleInterface.wsdl.

```
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:i0="http://LearnXmlWS.com/Supplier"
xmlns:tns="somenamespace"
targetNamespace="somenamespace"
xmlns="http://schemas.xmlsoap.org/wsdl/">
<import namespace="http://LearnXmlWS.com/Supplier"
  location=
"http://vbwsserver/vbwsbook/chapter8/SingleInterface.wsdl" />
 <types />
  <service name="SingleInterfaceImpl">
    <port name="ISupplier" binding="i0:ISupplier">
      <soap:address
       location="http://vbwsserver/vbwsbook/chapter8/
➥InterfaceImpl/SingleInterfaceImpl.asmx" />
    </port>
  </service>
</definitions>
```

The WSDL in Listing 8.5 is lacking most of what you're used to seeing in a WSDL document. It does not contain message, port, portType, or binding definitions. Instead, it imports the SingleInterface.wsdl document that contains the ISupplier interface definition. By having implementations reference the interface in this way, you avoid duplicating the interface definition with all the associated maintenance headaches.

8.3 Implementing Multiple Interfaces

The next logical step is to implement multiple interfaces on the same Web service. You do this by applying multiple `WebServiceBinding` attributes to the class implementing the Web service. On each Web method you set the `SoapDocumentService` or `SoapRpcService Binding` property to the name of the binding that contains this method. Taking the `SupplierIface1` class in Listing 8.1, you might want to factor out its methods into two interfaces: `IOrderMgmt` and `IQuoteMgmt`. Listing 8.6 shows the code used to achieve this.

Listing 8.6 Defining multiple bindings (interfaces) (VBWSBook\Chapter8\Supplier2.asmx.vb)

```
<WebServiceBinding( _
Name:="IOrderMgmt", _
[Namespace]:="http://LearnXmlWS.com/Supplier"), _
WebServiceBinding( _
Name:="IQuoteMgmt", _
[Namespace]:="http://LearnXmlWS.com/Supplier"), _
WebService([Namespace]:="http://LearnXmlWS.com/Supplier")> _
Public Class SupplierIface2
    Inherits System.Web.Services.WebService
    <WebMethod( _
    Description:= _
    "Places the order then returns the new order id"), _
    SoapDocumentMethod(Binding:="IOrderMgmt")> _
    Public Function PlaceOrder( _
                    ByVal newOrder As Order) As String
        'returns a new order id
    End Function
    <WebMethod(), _
    SoapDocumentMethod(Binding:="IOrderMgmt")> _
    Public Function CheckStatus( _
                        ByVal OrderId As String) As OrderInfo
            'returns an orderinfo structure
        End Function

        <WebMethod(), _
        SoapDocumentMethod(Binding:="IQuoteMgmt")> _
        Public Function GetPriceQuote( _
                        ByVal newOrder As Order) As QuoteInfo()
            'returns an orderinfo structure
        End Function
    End Class
```

Note that there are two bindings with different names—both in the same namespace. Also, `GetPriceQuote` is now part of the `IQuoteMgmt` binding.

Listing 8.7 shows the bindings defined in the resulting WSDL document.

Listing 8.7 The resulting WSDL document with two bindings (VBWSBook\Chapter8\MultiInterface.wsdl)

```
<binding name="IQuoteMgmt" type="s0:IQuoteMgmt">
 <soap:binding
       transport="http://schemas.xmlsoap.org/soap/http"
        style="document" />
  <operation name="GetPriceQuote">
    <soap:operation
      soapAction="http://LearnXmlWS.com/Supplier/GetPriceQuote"
      style="document" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
 </binding>

<binding name="IOrderMgmt" type="s0:IOrderMgmt">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
  <operation name="PlaceOrder">
    <soap:operation
       soapAction="http://LearnXmlWS.com/Supplier/PlaceOrder"
       style="document" />
    <input>
```

```
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
    <operation name="CheckStatus">
      <soap:operation
        soapAction="http://LearnXmlWS.com/Supplier/CheckStatus"
        style="document" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
  </binding>
```

To implement these interfaces you run wsdl.exe with the /server parameter just as you did for the single interface case. However, the output this time is different: You get one file that contains two classes—one for each binding. The first class you get, `IQuoteMgmt`, has one method called `GetPriceQuote`. The second class, `IOrderMgmt`, has two methods called `PlaceOrder` and `CheckStatus`. You can add this file to your Web service project and create classes that inherit from each of `IQuoteMgmt` and `IOrderMgmt`, and then start implementing each interface's methods. For example, Listing 8.8 shows a Web service that implements IQuoteMgmt.

Listing 8.8 An example Web service that implements IQuoteMgmt (VBWSBook\Chapter8\InterfaceImpl\QuoteMgmtImpl.asmx.vb)

```
Imports System.Web.Services
Imports System.Web.Services.Protocols

<WebServiceBinding( _
 Name:="IQuoteMgmt", _
 Namespace:="http://LearnXmlWS.com/Supplier", _
 Location:= _
 "http://vbwsserver/vbwsbook/chapter8/MultiInterface.wsdl"), _
 WebService(Namespace:="http://tempuri.org/")> _
Public Class QuoteMgmtImpl
```

```
        Inherits MultiIface1.IQuoteMgmt

        <WebMethodAttribute(), _
        SoapDocumentMethodAttribute( _
        "http://LearnXmlWS.com/Supplier/GetPriceQuote", _
         RequestNamespace:="http://LearnXmlWS.com/Supplier", _
         ResponseNamespace:="http://LearnXmlWS.com/Supplier", _
         Use:=Description.SoapBindingUse.Literal, _
         ParameterStyle:=SoapParameterStyle.Wrapped, _
         Binding:="IQuoteMgmt")> _
         Public Overrides Function GetPriceQuote( _
                ByVal newOrder As InterfaceImpl.MultiIface1.Order) _
                        As InterfaceImpl.MultiIface1.QuoteInfo()

        End Function
    End Class
```

Since .NET supports single inheritance, a class cannot inherit from more than one base class. This means you cannot create a Web service that inherits from both classes generated by wsdl.exe (the classes IQuoteMgmt and IOrderMgmt). You will end up with a Web service class for each binding you want to implement.

This is not exactly what comes to mind when I think of multiple interfaces. Ideally, we can have one Web service class that implements both IQuoteMgmt and IOrderMgt. While it's possible to achieve this, you can't do it by inheriting from abstract classes generated by wsdl.exe. Instead, you must create a Web service class, and then manually specify the bindings and the methods. The process is almost identical to the single-interface implementation case except you are not overriding any base class methods. The steps for implementing multiple interfaces on a single Web service class are:

- Create a Web service.
- Add a WebServiceBinding attribute to the Web service class for each interface you want to implement. Specify the binding's name, namespace, and location (the URL of a WSDL document where the binding is defined).
- Create WebMethods on this Web service that correspond to the operations defined in each binding. You can get some help from wsdl.exe by running it with the /server flag then copying the abstract method definitions it creates

and pasting them into your Web service. If you do this, be sure to remove the MustOverride keyword from those methods.

- Specify the binding name for each Web method. This name must match one of the binding names defined by SoapServiceBinding attributes on the Web service class.

Listing 8.9 shows an example Web service class that implements both IQuoteMgmt and IOrderMgmt.

Listing 8.9 A single Web service class that implements both IQuoteMgmt and IOrderMgmt interfaces (VBWSBook\Chapter8\InterfaceImpl\MultipleInterfaceImpl.asmx.vb)

```
Imports System.Web.Services
Imports System.Web.Services.Protocols

<WebService(Namespace:="http://tempuri.org/"), _
WebServiceBindingAttribute(Name:="IQuoteMgmt", _
 [Namespace]:="http://LearnXmlWS.com/Supplier", _
 Location:= _
 "http://vbwsserver/vbwsbook/chapter8/MultiInterface.wsdl"), _
 WebServiceBindingAttribute(Name:="IOrderMgmt", _
 [Namespace]:="http://LearnXmlWS.com/Supplier", _
Location:= _
"http://vbwsserver/vbwsbook/chapter8/MultiInterface.wsdl")> _
Public Class MultipleInterfaceImpl
    Inherits WebService

    <WebMethodAttribute(), _
        SoapDocumentMethodAttribute( _
        "http://LearnXmlWS.com/Supplier/GetPriceQuote", _
        RequestNamespace:="http://LearnXmlWS.com/Supplier", _
        ResponseNamespace:="http://LearnXmlWS.com/Supplier", _
        Use:=Description.SoapBindingUse.Literal, _
        ParameterStyle:=SoapParameterStyle.Wrapped, _
        Binding:="IQuoteMgmt")> _
    Public Function GetPriceQuote( _
                ByVal newOrder _
                As InterfaceImpl.MultiIface1.Order) _
                As InterfaceImpl.MultiIface1.QuoteInfo()
    End Function

    <WebMethodAttribute(), _
```

```
         SoapDocumentMethodAttribute( _
         "http://LearnXmlWS.com/Supplier/PlaceOrder", _
         RequestNamespace:="http://LearnXmlWS.com/Supplier", _
         ResponseNamespace:="http://LearnXmlWS.com/Supplier", _
         Use:=Description.SoapBindingUse.Literal, _
         ParameterStyle:=SoapParameterStyle.Wrapped, _
         Binding:="IOrderMgmt")> _
        Public Function PlaceOrder( _
                ByVal newOrder As InterfaceImpl.MultiIface1.Order) _
                 As String

        End Function


        <WebMethodAttribute(), _
         SoapDocumentMethodAttribute( _
         "http://LearnXmlWS.com/Supplier/CheckStatus", _
         RequestNamespace:="http://LearnXmlWS.com/Supplier", _
         ResponseNamespace:="http://LearnXmlWS.com/Supplier", _
         Use:=Description.SoapBindingUse.Literal, _
         ParameterStyle:=SoapParameterStyle.Wrapped, _
         Binding:="IOrderMgmt")> _
        Public Function CheckStatus(ByVal OrderId As String) _
               As InterfaceImpl.MultiIface1.OrderInfo

        End Function
    End Class
```

The Web service class in Listing 8.9 has two WebServiceBinding attributes for IQuoteMgmt and ISupplierMgmt. Inside the class, you'll see the GetPriceQuote Web method that belongs to the IQuoteMgmt interface. You'll also see the IOrderMgmt interface methods: PlaceOrder and CheckStatus. Listing 8.10 shows the resulting WSDL.

Listing 8.10 The WSDL for the service in Listing 8.9

```
<?xml version="1.0" encoding="utf-8"?>
<definitions
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:i0="http://LearnXmlWS.com/Supplier"
xmlns:tns="http://tempuri.org/"
targetNamespace="http://tempuri.org/"
```

```
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <import namespace="http://LearnXmlWS.com/Supplier"
  location=
"http://vbwsserver/vbwsbook/chapter8/MultiInterface.wsdl" />
  <types />
  <service name="MultipleInterfaceImpl">
    <port name="IOrderMgmt" binding="i0:IOrderMgmt">
      <soap:address
location="http://vbwsserver/vbwsbook/chapter8/_
InterfaceImpl/MultipleInterfaceImpl.asmx" />
    </port>
    <port name="IQuoteMgmt" binding="i0:IQuoteMgmt">
      <soap:address
location="http://vbwsserver/vbwsbook/chapter8/_
InterfaceImpl/MultipleInterfaceImpl.asmx" />
    </port>
  </service>
</definitions>
```

The most interesting feature of the WSDL in Listing 8.10 is the presence of two `<port>` elements inside the `<service>` element, indicating that the service implements two different interfaces.

8.4 Interfaces in Different Namespaces

Here's an interesting twist on the above scenario: What if we placed the bindings in different namespaces, for example, `IQuoteMgmt` in a namespace called `http://LearnXmlWS.com/QuoteMgmt` and `IOrderMgmt` in another namespace called `http://LearnXmlWS.com/OrderMgmt`? Listing 8.11 shows the code for `SupplierIface3` that has two bindings, each in its own namespace.

Listing 8.11 A Web service with two bindings in two different namespaces (VBWSBook\Chapter8\Supplier3.asmx.vb)

```
<WebServiceBinding( _
Name:="IOrderMgmt", _
[Namespace]:="http://LearnXmlWS.com/OrderMgmt"), _
WebServiceBinding( _
Name:="IQuoteMgmt", _
[Namespace]:="http://LearnXmlWS.com/QuoteMgmt"), _
WebService([Namespace]:="http://LearnXmlWS.com/Supplier")> _
```

```
Public Class SupplierIface3
        Inherits System.Web.Services.WebService
        <WebMethod( _
  Description:="Places the order then returns the new order id"),
_
        SoapDocumentMethod(Binding:="IOrderMgmt")> _
        Public Function PlaceOrder(ByVal newOrder As Order) _
                         As String
            'returns a new order id
        End Function
        <WebMethod(), _
        SoapDocumentMethod(Binding:="IOrderMgmt")> _
        Public Function CheckStatus(ByVal OrderId As String) _
                          As OrderInfo
            'returns an orderinfo structure
        End Function

        <WebMethod(), _
        SoapDocumentMethod(Binding:="IQuoteMgmt")> _
        Public Function GetPriceQuote(ByVal newOrder As Order) _
                           As QuoteInfo()
            'returns an orderinfo structure
        End Function
 End Class
```

The result is that each binding (interface) is defined in its own WSDL document. This is because bindings belong to the WSDL document's `targetNamespace` and you can have one targetNamepace only per WSDL document. These separate WSDL documents are imported into the main WSDL document using the `<import>` element as shown in Listing 8.12.

Listing 8.12 Each binding is defined in a separate WSDL document that is then imported into the main WSDL document using `<import>`.

```
<definitions targetNamespace="http://LearnXmlWS.com/Supplier"
➥...>
  <import namespace="http://LearnXmlWS.com/QuoteMgmt"
     location="http://vbwsserver/vbwsbook/Chapter8/
➥supplier3.asmx?schema=schema1"/>
  <import namespace="http://LearnXmlWS.com/OrderMgmt"
   location="http://vbwsserver/vbwsbook/Chapter8/
➥supplier3.asmx?schema=schema2"
/>
```

```
  <import namespace="http://LearnXmlWS.com/QuoteMgmt"
    location="http://vbwsserver/vbwsbook/Chapter8/_
supplier3.asmx?wsdl=wsdl1" />
  <import namespace="http://LearnXmlWS.com/OrderMgmt"
    location="http://vbwsserver/vbwsbook/Chapter8/_
supplier3.asmx?wsdl=wsdl2" />
  <types />
  <service name="SupplierIface3">
    <port name="IQuoteMgmt" binding="i2:IQuoteMgmt">
      <soap:address
         location="http://vbwsserver/vbwsbook/Chapter8/_
supplier3.asmx" />
    </port>
    <port name="IOrderMgmt" binding="i1:IOrderMgmt">
      <soap:address location=
        "http://vbwsserver/vbwsbook/Chapter8/supplier3.asmx" />
    </port>
  </service>
</definitions>
```

To access the `IQuoteMgmt` binding definition, you navigate to the .asmx file with a query string of `wsdl=wsdl1` (for example, `supplier3.asmx?wsdl=wsdl1)`. Similarly, `IOrderMgmt` definition is at `supplier3.asmx?wsdl=wsdl2`. Similarly, types used by the methods of each interface, for example, `Order` and `QuoteInfo`, are defined in the schemas located at `supplier3.asmx?schema=schema1` and `supplier3.asmx?schema=schema2`. In this case, the interfaces, type definitions, and implementation definition (the service) are all separated in different locations. For complex Web services that implement many interfaces, separating interface definitions makes them easier to maintain and be independent of one another.

Running wsdl.exe with /server on the WSDL in Listing 8.12 generates two separate classes: one for each binding, as in the previous case where both bindings belonged to the same class. The difference in this case is that class has a `WebServiceBinding` attribute on it with its `Namespace` property set to `http://LearnXmlWS.com/QuoteMgmt` for `IQuoteMgmt` and `http://LearnXmlWS.com/OrderMgmt` for `IOrderMgmt`.

When you run wsdl.exe on the WSDL document in Listing 8.12, it needs to get the other WSDL documents referenced by `<import>` elements. This works fine as long as you tell wsdl.exe to access the main WSDL document using http, (for example

```
wsdl.exe http://localhost/SupplierIface3.asmx?WSDL).
```

If however, you give wsdl.exe a file path instead of a URL, it will not be able to retrieve the imported WSDL documents. So if you save the WSDL document in Listing 8.12 to disk and issue the following command:

```
wsdl.exe D:\documents\SupplierIface3.wsdl
```

you will get an error about undefined types because wsdl.exe cannot read the schema or the WSDL documents referenced by `<import>`s.

8.5 Programming Against Interfaces

The previous section showed you how to define and implement interfaces. This section completes the picture by showing you how to code clients against those interfaces.

8.5.1 *Generating Proxies from Interfaces*

On the client side, a Web service proxy exposes the set of methods that reflects the binding's operations. As far as the client is concerned, this set of methods is considered the Web service's interface.

When you add a Web reference to an interface-only WSDL document, Visual Studio .NET recognizes an interface-only WSDL and generates the proxy class for you (beta versions didn't do this). Since the WSDL document doesn't contain a `<service>` element, the generated proxy class will not have a Web service URL in its constructor. You can also use wsdl.exe to generate the proxy class:

```
wsdl.exe /l:VB /out:MyClass.vb SingleInterface.wsdl
```

Listing 8.13 shows two example proxy classes one for `IOrderMgmt` and one for `IQuoteMgmt`. These proxies were generated using wsdl.exe.

Listing 8.13 Proxy classes generated based on IOrderMgmt and IQuoteMgmt interface definitions (VBWSClientCode\Chapter8\Interfaces\InterfaceClient\Supplier3.vb)

```
<WebServiceBindingAttribute( _
    Name:="IOrderMgmt", _
    [Namespace]:="http://LearnXmlWS.com/OrderMgmt")> _
Public Class COrderMgmt
    Inherits SoapHttpClientProtocol

    Public Sub New()
        MyBase.New()
    End Sub

    <SoapDocumentMethod( _
        "http://LearnXmlWS.com/Supplier/PlaceOrder", _
        RequestNamespace:="http://LearnXmlWS.com/Supplier", _
        ResponseNamespace:="http://LearnXmlWS.com/Supplier")> _
    Public Function PlaceOrder(ByVal newOrder As Order) As String
        Dim results() As Object = Me.Invoke("PlaceOrder", _
                            New Object() {newOrder})
        Return CType(results(0), String)
    End Function


    <SoapDocumentMethod( _
     "http://LearnXmlWS.com/Supplier/CheckStatus", _
     RequestNamespace:="http://LearnXmlWS.com/Supplier", _
     ResponseNamespace:="http://LearnXmlWS.com/Supplier")> _
     Public Function CheckStatus( _
                 ByVal OrderId As String) As OrderInfo
        Dim results() As Object = Me.Invoke("CheckStatus", _
                            New Object() {OrderId})
        Return CType(results(0), OrderInfo)
    End Function

End Class

<WebServiceBinding( _
    Name:="IQuoteMgmt", _
    [Namespace]:="http://LearnXmlWS.com/QuoteMgmt")> _
Public Class CQuoteMgmt
    Inherits SoapHttpClientProtocol
```

```
        <SoapDocumentMethod( _
            "http://LearnXmlWS.com/Supplier/GetPriceQuote", _
            RequestNamespace:="http://LearnXmlWS.com/Supplier", _
            ResponseNamespace:="http://LearnXmlWS.com/Supplier")> _
        Public Function GetPriceQuote( _
                        ByVal newOrder As InterfaceClient.Order) _
                    As InterfaceClient.QuoteInfo()
            Dim results() As Object = Me.Invoke("GetPriceQuote", _
                                New Object() {newOrder})
            Return CType(results(0), QuoteInfo())
        End Function
    End Class
```

I specifically renamed the classes to begin with a "C" to make it clear that these are classes and not interfaces. The first class, `COrderMgmt`, represents the `IOrderMgmt` binding as indicated by its `WebServiceBinding` attribute. The class inherits from `SoapHttpClientProtocol` and adds two methods, `PlaceOrder` and `CheckStatus`, each with a `SoapDocumentMethod` attribute that defines the request and response namespaces as defined in the WSDL. Similarly, the second class, `CQuoteMgmt`, represents the `IQuoteMgmt` binding. It also inherits from `SoapHttpClientProtocol` and adds a function called `GetPriceQuote` as defined in the WSDL.

The classes in Listing 8.13 are definitely good enough for the client to call the service based on the two interfaces IOrderMgmt and IQuoteMgmt. However, some developers might want to take it a step further and program against interfaces, instead of classes, on the client side.

Unfortunately, `WebServiceBinding` cannot be applied to interfaces. One way around this is to define manually two interfaces on the client side called `IOrderMgmt` and `IQuoteMgmt` as shown in Listing 8.14.

Listing 8.14 Defining interfaces on the client side and creating a class that implements these interfaces and forwards method calls to the Web service proxy classes (VBWSClientCode\Chapter8\Interfaces\InterfaceClient\Supplier3.vb)

```
    Interface IOrderMgmt
        Function PlaceOrder(ByVal newOrder As Order) As String
        Function CheckStatus(ByVal OrderId As String) As OrderInfo
    End Interface
```

```
Interface IQuoteMgmt
    Function GetPriceQuote(ByVal newOrder As Order) As QuoteInfo()
End Interface
Public Class SupplierProxy
    Implements IOrderMgmt
    Implements IQuoteMgmt
    Private _ordermgmt As COrderMgmt
    Private _quotemgmt As CQuoteMgmt

    Public Function CheckStatus( _
                ByVal OrderId As String) As OrderInfo _
            Implements InterfaceClient.IOrderMgmt.CheckStatus
        Return _ordermgmt.CheckStatus(OrderId)
    End Function

    Public Function PlaceOrder(ByVal newOrder As Order) _
           As String _
            Implements InterfaceClient.IOrderMgmt.PlaceOrder
        Return _ordermgmt.PlaceOrder(newOrder)
    End Function

    Public Function GetPriceQuote( _
                ByVal newOrder As Order) As QuoteInfo() _
            Implements InterfaceClient.IQuoteMgmt.GetPriceQuote
        Return _quotemgmt.GetPriceQuote(newOrder)
    End Function

    Public Sub New(ByVal Url As String)
        _ordermgmt = New COrderMgmt()
        _quotemgmt = New CQuoteMgmt()
        _ordermgmt.Url = Url
        _quotemgmt.Url = Url
    End Sub
End Class
```

Each interface includes methods of the corresponding binding as defined in the WSDL document. A class, called `SupplierProxy`, implements both interfaces and contains instances of `COrderMgmt` and `CQuoteMgmt`. You can build the code in Listing 8.14 in a separate .dll and distribute it to client developers as a prepackaged interface-based proxy for your Web service. This is valuable if you are responsible for maintaining the Web service and the proxy classes and you want to abstract client developers from the details of your Web service's interface.

Listing 8.15 shows example client code that is designed to work against the interfaces `IOrderMgmt` and `IQuoteMgmt`.

Listing 8.15 An example client that works against the interfaces not the Web service proxy (VBWSClientCode\Chapter8\Interfaces\InterfaceClient\Form1.vb)

```
Private Const SERVICE_URL As String = _
    "http://VBWSServer/vbwsbook/Chapter8/Supplier3.asmx"
Private Sub btnPlaceOrder_Click(ByVal sender As System.Object, _
           ByVal e As System.EventArgs) Handles
btnPlaceOrder.Click
    Dim ws As IOrderMgmt
    ws = New SupplierProxy(SERVICE_URL)
    ws.PlaceOrder(MakeOrder())
End Sub
Private Sub btnQuote_Click(ByVal sender As System.Object, _
          ByVal e As System.EventArgs) Handles btnQuote.Click
    Dim ws As IQuoteMgmt
    ws = New SupplierProxy(SERVICE_URL)
    ws.GetPriceQuote(MakeOrder())
End Sub
```

To invoke the methods of the `IOrderMgmt` binding, the client declares a variable of type `IOrderMgmt`, then sets it to a new instance of `SupplierProxy` and proceeds to call `PlaceOrder` or `CheckStatus`. Similarly, to call methods of `IQuoteMgmt`, the client declares a variable of type `IQuoteMgmt` and sets it to a new instance of `SupplierProxy` and calls `GetPriceQuote`.

The code in Listing 8.15 makes it very clear that the client is programmed against an interface and not a specific implementation. However, there's still one glaring problem with this code: The Web service URL, which is the location of a specific implementation of `IOrderMgmt` and `IQuoteMgmt`, is hard-coded as a constant. The next sections deal with determining the Web service URL at runtime based on external configuration settings.

8.6 Choosing Implementations at Runtime

Instead of hard-coding the service URL in client code, you can add it to the application's XML configuration file. For Web applications, this is the web.config file that's in the application's vroot or the folder where your Web form is located. For all other application types, it's a file with the same name as the main application's executable, but with a .config extension. For example, if your Windows application is called myapp.exe, the configuration file is called mayapp.exe.config and resides in the same folder as myapp.exe. The application's configuration file has an `<appSettings>` section where you can add your own configuration information. For example, if you know the Web service's URL, you can add it like this:

```
<appSettings>
   <add key="WSUrl" value="http://hostname/service.asmx" />
</appSettings>
```

At runtime, you use `System.Configuration.ConfigurationSettings` to read the URL from the config file:

```
theProxy.Url= _
   System.Configuration.ConfigurationSettings _
  .AppSettings ("WSUrl")
```

Instead of writing this code yourself, when you run wsdl.exe to generate the proxy class, use the /appsettingurlkey switch like this:

```
wsdl.exe /l:VB /out:proxy.vb http://localhost/service.asmx?wsdl
/➥appsettingurlkey:urlkeyname
```

where `urlkeyname` is the name you used for the Web service URL configuration key, (`WSUrl` in this example). Alternatively, if you add a Web reference with VS .NET, select the Web reference and open its properties. Change the URL Behavior property to Dynamic (default is Static). The resulting proxy class contains the code to read from `AppSettings` as shown in Listing 8.16.

Listing 8.16 The proxy class generated by wsdl.exe reads from AppSettings in the constructor

```
Public Sub New()
    MyBase.New
    Dim urlSetting As String = _
      System.Configuration.ConfigurationSettings.AppSettings( _
                  "VBWSServer.DataService")
    If (Not (urlSetting) Is Nothing) Then
        Me.Url = String.Concat(urlSetting, "")
    Else
        Me.Url = _
         "http://vbwsserver/vbwsbook/Chapter8/DataService.asmx"
    End If
End Sub
```

As a rule, you should not leave the Web service URL hard-coded in a production client. By making it configurable, you can avoid client recompilation, testing, and deployment when the Web service URL changes.

8.7 Summary

In this chapter you were introduced to the concept of interface-based programming as it applies to .NET Web services. You learned how to develop Web services based on interfaces by separating the WSDL binding definition from the Web service implementation, then using wsdl.exe to generate Web service implementations from a WSDL document. You also learned how to implement multiple interfaces (bindings) using a single class and what the resulting WSDL looks like. Finally, you saw how clients can program against interfaces and how to dynamically read the Web service's URL from a configuration file rather than hard-coding it in the client.

8.8 Resources

WSDL discussion list http://groups.yahoo.com/group/wsdl/ .

Web Services Description Language (WSDL) 1.1: http://www.w3.org/TR/wsdl.

DevelopMentor's .NET Web discussion list: http://discuss.develop.com/dotnet-web.html.

W3C XML Schema specifications and resources: http://www.w3.org/XML/Schema.