13 шагов к Windows Communication Foundation (WCF)

WCF призвана облегчить для разработчиков решение многих задач, а вместе с тем, как и другие новые технологии, требует определенных усилий для своего освоения. Следуя приведенным ниже тринадцати «шагам», вы сможете быстро приобрести необходимые знания в области WCF.

WCF – это новый замечательный компонент в .NET 3.0 Framework, обеспечивающий универсальную модель программирования для распределенных приложений. Разработчики, которым приходилось создавать различный программный код для Web сервисов и для удаленного взаимодействия в .NET, несомненно, смогут оценить те мощные возможности стандартизации, которые предоставляет WCF. Как все новые технологии, WCF ждет своих исследователей и экспериментаторов, совместными усилиями которых он должен превратиться в продуктивное средство.

В статье вниманию читателей предлагается краткий курс по основам Windows Communication Foundation (WCF). Для изучения этого материала не требуется какого-либо предварительного опыта работы с WCF. Последовательно осваивая основные понятия и участвуя в создании кода, читатель познает возможности WCF.

Windows Communication Foundation, являясь одним из важнейших компонентов .NET 3.0 Framework, позволяет устранить сложности в создании программного кода, обусловленные различиями между коммуникационными протоколами.

1 Вступление

Вашему вниманию предлагается путеводитель по созданию распределенных приложений посредством WCF. Иногда в справочных материалах завышаются требования к знаниям читателей в конкретной области. В данном случае эти требования сведены к минимуму. Если вы когда-либо создавали простое WCF приложение, то вряд ли Вы найдете здесь что-то новое (разве что, «освежите» в памяти свой опыт). Это статья – для новичков в WCF. В этом курсе нам предстоит пройти следующие тринадцать шагов:

1. Определить, что необходимо загрузить для разработки WCF приложения.
2. Обозначить требования к демонстрационной версии распределенного приложения с использованием WCF, создание которого будет описано в этой статье.
3. Рассмотреть обзорный материал по архитектуре WCF, а также способы использования WCF с учетом требований к нашему демо-приложению.
4. Разработать первое демо-приложение—WCF сервис с собственным хостингом (self-hosted), в котором мы построим необходимые .NET интерфейсы, предназначенные для различных уровней взаимодействия (часть 1.1).
5. Создать демонстрационный .NET сервер и установить конфигурационный файл для хостинга WCF сервиса (часть 1.2).
6. Построить в .NET клиентскую часть, которой будет предоставлять доступ к WCF сервису (часть 1.3).
7. Модифицировать первое демо-приложение с тем, чтобы вместо конфигурационных файлов более полно использовать в коде объектную модель WCF.

И остальные 6 шагов:

1. Создать второе демо-приложение для построения и хостинга WCF сервиса с помощью IIS; аналогично работе с файлами ASMX (часть 2.1).
2. Обеспечить хостинг WCF сервиса, используя IIS: создание клиентской части (часть 2.2).
3. Создать WCF XML Web сервис и установить конфигурации с тем, чтобы этот сервис использовался приложениями, не являющимися .NET клиентами (часть 3.1).
4. Зарегистрировать WCF XML Web сервис посредством IIS (часть 3.2).
5. Написать код для использования WCF сервиса отличным от .NET клиентом (часть 3.3).
6. Обратиться к некоторым опциям безопасности посредством WCF.

2 Немного о распределенных вычислениях

Прежде всего, попытаемся понять, какое место занимает WCF в распределенных вычислениях.

Распределенные вычисления можно рассматривать как архитектуру, в которой различные компоненты приложения расположены на различных компьютерных системах (их часто называют доменами). К приме-

Comment [A.A.1]: Не следует полагать, что использование WCF возможно только в среде Vista—весь приведенный здесь код можно компилировать и запускать в Windows XP Professional.

ру, вы загрузили SQL Query Analyzer (или SQL Management Studio) со своего компьютера, а базу данных открыли с сервера баз данных фирмы. После ввода запроса или запуска хранимой процедуры получаем результаты. Это очень характерный пример распределенных вычислений, когда пользователь вводит запрос со своего компьютера, а сервер производит необходимую работу и возвращает пользователю результаты. Таким образом, каждый компонент приложения выполняет свою часть общей задачи.

Попробуем взглянуть на вещи с точки зрения разработчиков систем уровня предприятия — необходимо разумно распределить процессы обработки между серверами (доменами). Типичное распределенное приложение может включать в себя клиентские модули, Web сервер, а также серверы приложений и баз данных. Возможно, вы захотите, чтобы определенные процессы выполнялись на определенных доменах — с тем, чтобы выгодно использовать аппаратные особенности, или следуя определенным рекомендациям в разработке, или учитывая ограничения и системные политики, или по каким-либо другим причинам. В этой статье не рассматривается проектирование архитектуры распределенного приложения, и предполагается, что читатель заранее решил для себя, где какие процессы будут происходить.

Разработчики часто используют XML Web сервисы либо .NET протоколы для удаленного доступа (или и то, и другое вместе), и компоненты распределенных приложений должны иметь возможность взаимодействовать друг с другом. До появления WCF разработчикам зачастую приходилось писать и поддерживать программы для различных используемых протоколов. WCF позволяет работать с этими протоколами, используя универсальную модель программирования и таким образом устраняя необходимость писать свой, отличный от других, код для каждого протокола.

Мы пройдем несколько этапов создания простых WCF приложений, в которых используются как Web сервисы, так и протоколы удаленного доступа. И в конце статьи придем к достаточно полному пониманию о том, как использовать модели программирования WCF при написании единого кода независимо от различий в протоколах.

3 Шаг 1-й: Определяем то, что необходимо для WCF

Для начала можно найти основную информацию на главном сайте Microsoft WCF (http://wcf.netfx3.com/).

Comment [A.A.2]: Если Вы использовали NET интерфейсы и/или .NET Generics перед изучением WCF, то это упростит дальнейшее обучение.

Прежде чем обратиться к приводимым в статье примерам, вам нужно загрузить все необходимые компоненты WCF. Вот список этих компонентов:

1. Microsoft .NET Framework версии 3.0. В этой версии представлена вся функциональность для WCF, и, кроме того, она включает в себя Windows Presentation Foundation (WPF) и Windows Workflow Foundation (WF).
2. Microsoft Windows SDK для Windows Vista и компоненты .NET Framework 3.0 Runtime (для загрузки и инсталляции потребуется достаточно много времени).
3. Расширения Visual Studio 2005 для .NET 3.0 Framework. Они позволяют собирать WCF приложения в Visual Studio 2005.

Comment [A.A.3]: Несмотря на то, что WCF можно строить и в XP Professional, но использовать для разработки Visual Studio 2003 нельзя.

4 Шаг 2-й: Определим требования к демо-приложению

Приложение состоит из двух простых классов: customer и customer order status. Класс customer возвращает информацию для определенного клиента, а класс customer order status возвращает основную информацию о заказе от определенного клиента.

Для доступа к этим двум классам можно использовать любой из следующих протоколов:

- TCP из приложений на .NET Windows Forms (подобно удаленному взаимодействию в .NET);
- HTTP из приложений на .NET Windows Forms и на Web Forms (с возможностью перехода от одного к другому посредством конфигурационных файлов);
- HTTP и SOAP из отличного от .NET приложения.

5 Шаг 3-й: Обзор архитектуры WCF

Сначала поговорим немного об основных концепциях. На самом высоком уровне WCF можно использовать в следующих областях:

- Web-сервисы;
- коммуникации от .NET к .NET (посредством HTTP, TCP);
- распределенные транзакции;
- WS-* спецификации;
- обмен сообщениями (используются очереди сообщений).

В сущности, это означает, что использование модели WCF распространяется на достаточно широкий круг областей, для которых ранее требовались различные подходы.

WCF содержит четыре ключевых компонента, которые мы будем обсуждать далее в статье:

- контракты (contract):
- биндинги (binding):
- описания конечной точки (endpoint definition);
- среда для хостинга (hosting environment).

Не пугайтесь, *контракт* – это не совсем новый термин. Здесь речь идет о “давнем знакомом”, интерфейсе .NET, который, по определению, задает контракт о том, что к нему должен “прикрепляться” класс, реализующий этот интерфейс. Следовательно, WCF-контракт определяет операции, которые могут производиться сервисом. На 4-м шаге мы познакомимся с особенностями построения интерфейса и «заключеиия» WCF контрактов.

Биндинги (bindings) – это коммуникационные каналы между WCF-сервисом и клиентом (такие как TCP, HTTP и т.п.). В таблице 1 перечислены биндинги WCF, которые используются в приводимых здесь примерах кода.

Таблица 1: Биндинги WCF

Биндинги
BasicHttpBinding
WSHttpBinding
WSDualHttpBinding
WSFederationHttpBinding
MsmqIntegrationBinding
NetMsmqBinding
NetNamedPipeBinding
NetPeerTcpBinding
NetTcpBinding

Описания конечной точки (endpoint definitions) ссылаются на адреса URI (Uniform Resource Identifier) для соединения с WCF сервисом и состоят из базового адреса, а также биндинга и информации контракта. В файлах конфигурации на 5-м и 6-м шагах будут примеры описаний конечной точки (endpoint definitions). WCF позволяет определять столько конечных точек, сколько необходимо, к примеру, специальную конечную точку для .NET-клиента, а другую – для отличного от .NET клиента.

Среда для хостинга (hosting environment) определяет, какая архитектура используется для хостинга WCF-сервисов. Хостинг WCF сервиса может осуществляться или в .NET приложении, или в службе Windows. Для этого также может использоваться IIS, что мы будем обсуждать далее (с 9-го по 11-ый шаг). Обратите внимание на то, что в данном случае как минимум одно из описаний конечной точки должно быть типа HTTP. И, наконец, для хостинга WCF сервиса можно использовать новый Windows Activation Service в IIS 7.0

Вот что будет обсуждаться на следующих шагах:

- Шаги с 4-го по 6-й посвящены построению простого WCF приложения с собственным хостингом, где используются конфигурационные файлы.
- На 7-м шаге рассмотрим построение такого же приложения, но уже без использования конфигурационных файлов. Здесь будет представлено немного больше кода для демонстрации объектной модели WCF.
- На 8-м шаге предстоит создать IIS-hosted WCF сервис, а затем мы узнаем, как добавить ссылку на WCF сервис для .NET клиента, использующего этот сервис (подобно тому, как нужно было добавлять ссылку на Web сервис ранее до появления WCF).
- Шаги с 9-го по 11-й посвящены описанию другого WCF сервиса с IIS-хостингом, на этот раз предназначенного для отличного от .NET клиента.

6 Шаг 4: Первое демо-приложение, Построение интерфейса (часть 1.1)

Наконец наступило время программирования! Начнем с построения интерфейса для двух методов, GetCustomer и GetOrderHistory, описанных ранее (на шаге 2-м). Оба метода получают customerID типа integer в качестве параметра и возвращают строку XML как результат. Мы будем конструировать интерфейс, который должен устанавливать контракты WCF для любого клиента, использующего данный сервис.

Итак, действовать будем в такой последовательности:

1. Запустим Visual Studio 2005 и выберем опцию создания нового проекта библиотеки классов под названием DemoInterfaces.
2. В Solution Explorer щелкнем на references и добавим ссылку .NET на System.ServiceModel. Это основная DLL в .NET 3.0 Framework для WCF. Не забывайте добавлять эту ссылку .NET на System.ServiceModel всякий раз при написании кода с использованием объектной модели WCF.
3. Добавьте код из Листинга 1 или Листинга 2 (ICustomer.cs или ICustomer.vb).
4. Откомпилируйте проект, в результате создается библиотека DemoInterfaces.DLL.

Код public-интерфейса в Листинге 1 и в Листинге 2 содержит два новых ключевых слова-атрибута: ServiceContract и OperationContract.

```
[ServiceContract]
public interface ICustomer
{
   [OperationContract]
   string GetCustomer(int CustomerID);

   [OperationContract]
   string GetOrderHistory(int CustomerID);
}
```

Атрибут ServiceContact определяет набор операций, которые могут производиться сервисом, а другой атрибут, OperationContract, идентифицирует конкретные методы. .NET CLR транслирует эти интерфейсы в SOAP типы.

Мы будем использовать этот интерфейс как на сервере для хостинга WCF сервиса, так и на клиенте, использующем этот сервис. Интерфейс является контрактом, установленным между клиентом и сервером.

7 Шаг 5: Первое демо-приложение, Построение бизнес-объекта и сервера (часть 1.2)

После того как мы установили интерфейс, следующим шагом будет построение бизнес-объекта, к которому будет иметь доступ клиентская сторона, а затем перейдем к созданию кода небольшого процесса на стороне сервера для хостинга сервиса.

Бизнес-объект будем создавать в такой последовательности:

1. Создадим новый проект библиотеки классов с именем DemoCustomerBz.
2. В Solution Explorer кликнем правой кнопкой на references и добавим ссылку .NET к библиотеке DemoInterfaces.DLL, созданной на предыдущем шаге в интерфейсном проекте.
3. Добавим код из Листинга 3 или Листинга 4 (CustomerBz.cs или CustomerBz.vb).
4. Скомпилируем проект, в результате создается библиотека DemoCustomerBz.DLL.

Единственный заслуживающий внимания аспект бизнес-объекта (помимо того факта, что его методы являются тестирующими и возвращают фиктивные тестовые данные) состоит в том, что класс реализует интерфейс ICustomer из предыдущего шага.

```
public class CustomerBz : ICustomer
```

Одним из предъявляемых к демо-приложению требований является возможность доступа к серверному бизнес-объекту посредством различных коммуникационных протоколов (TCP, HTTP и т.п.). До появления WCF разработчикам, применяющим удаленное взаимодействие в .NET, зачастую приходилось использовать System.MarshalByRefObject наряду с интерфейсами, что позволяло устанавливать необходимый обмен сообщениями между прокси (proxy) клиента и удаленным объектом.

В WCF для удаленного TCP используется описание биндинга конечной точки (endpoint binding definition) и интерфейс; то есть System.MarshalByRefObject больше не потребуется, и серверный класс выглядит так же, как любой другой класс, реализующий интерфейс. Иначе говоря, появилась возможность отделить интерфейс от реализации.

Теперь можно построить небольшое серверное приложение, выполняющее хостинг сервиса, для чего проделаем следующее:

1. Создадим новый проект Windows Forms с именем DemoServer.
2. В Solution Explorer, кликнув правой кнопкой на references, добавим ссылки .NET для обеих DemoInterfaces.DLL и DemoCustomerBz.DLL, а также как для System.ServiceModel.DLL.
3. Кликнем правой кнопкой мыши и добавим новый пункт, а именно, Application Configuration File (App.Config) и вставим в него текст из Листинга 5.
4. На главной форме приложения Windows Forms установим две командные кнопки и метку (label), затем добавим код из Листинга 6 или Листинга 7 (DemoServer.cs или DemoServer.vb).

Хостинг на стороне сервера очень простой. Код включает в себя ссылку App.Config на сервис (начинается с имени бизнес-класса):

```
<system.serviceModel>
  <services>
   <service name="DemoCustomerBz.CustomerBz">
```

Кроме того, App.Config содержит адрес конечной точки, биндинг и имя контракта. В нашем первом примере используется базовый TCP биндинг для localhost-порта 8228. В контракте назначается интерфейс. Затем мы будем добавлять другие адреса конечных точек и покажем, как множество клиентов может получить доступ к различным адресам и биндингам.

```
    <endpoint
      address="net.tcp://localhost:8228/CustomerBz"
      binding="netTcpBinding"
      contract="DemoInterfaces.ICustomer" />
    </service>
  </services>
</system.serviceModel>
```

И, наконец, код приложения открывает сервис, используя класс ServiceHost:

```
ServiceHost oHost = new ServiceHost(typeof(CustomerBz));
oHost.Open();

// TO CLOSE
oHost.Close();
```

Итак, мы создали программу-listener, self-hosting приложение, позволяющее клиентскому приложению применять заданный TCP адрес для соединения, а также использовать серверный класс CustomerBz, реализующий контракт для интерфейса ICustomer. Следующим шагом будет создание клиентского приложения.

8 Шаг 6: Первое демо-приложение, Построение клиентской части (часть 1.3)

Чтобы сконструировать простое .NET приложение, которое получило бы доступ к сервису для его использования, проделаем следующее:

1. Создадим новый проект Windows Forms с именем DemoClient.
2. Клиентское приложение будет пользоваться интерфейсом ICustomer и объектной моделью WCF. Учитывая это, кликнув правой кнопкой мыши, добавим в Solution Explorer ссылки .NET на DemoInterfaces.DLL, а также на System.ServiceModel.DLL.
3. Кликнем правой кнопкой мыши и добавим новый пункт как Application Configuration File (App.Config), а затем и вставим текст из Листинга 8.
4. На главной форме для Windows Forms приложения установим две командные кнопки и метку (label), затем добавим код из Листинга 9 или Листинга 10.

Обратите внимание: относящиеся к клиентской части конфигурационные установки для App.Config почти полностью идентичны соответствующим установкам для серверной части, описанным на предыдущем этапе.

```
<client>
  <endpoint
    address="net.tcp://localhost:5555/CustomerBz"
    binding="netTcpBinding"
    contract="DemoInterfaces.ICustomer"
    name="Customer" />
</client>
```

В приведенном коде также задано имя переменной (в имени тега), которая будет использоваться в приложении для относящейся к соединению информации. Эту переменную необходимо применять, чтобы предоставить пользователю множественные коммуникационные опции.

Для активации сервиса на стороне клиента в коде используется класс ChannelFactory, который играет роль “краеугольного камня” в построении клиентской части. ChannelFactory представляет собой класс-фабрику (factory class), который создает для клиента коммуникационный канал заданного типа для отправки сообщений в конечную точку, которая была задана в конфигурации.

```
ChannelFactory<ICustomer> customersFactory =
```

```
new ChannelFactory<ICustomer>("Customer");
```

Это можно представить себе таким образом: мы создаем класс-фабрику (factory class) с именем customersFactory. Это класс типа интерфейса ICustomer, наследующий всю относящуюся к конечной точке информацию, сопоставленную тегу "Customer".

Пока еще мы не получили доступ к сервису WCF— только создали класс от ChannelFactory, который позволяет создать экземпляр строго типизированного прокси для последующего использования при коммуникации с сервисом. В следующей строке кода создается типизированный proxy (customersProxy) типа ICustomer с помощью ChannelFactory, базового метода CreateChannel:

```
ICustomer customersProxy =
   customersFactory.CreateChannel();
```

Теперь с помощью customersProxy открывается доступ к двум методам, GetCustomer и GetOrderHistory:

```
string CustomerString =
   customersProxy.GetCustomer(1);
string CustomerHistory =
   customersProxy.GetOrderHistory(1);
```

Кроме того, если нужно изменить коммуникационый биндинг, чтобы использовать HTTP или же другой TCP порт, достаточно будет всего лишь модифицировать и «растиражировать» файлы app.config.

```
<endpoint
  address="http://localhost:8080/CustomerBz"
  binding="basicHttpBinding"
  contract="DemoInterfaces.ICustomer" />
```

9 Шаг 7: Написание кода вместо конфигурационных файлов

В шагах с 4-го по 6-й мы использовали конфигурационные файлы, тем самым сокращая размер разрабатываемого кода. Но, что представляется более существенным, при этом сохранялась возможность осуществлять все необходимые изменения и тиражировать их посредством конфигурационных файлов и не требовалось перекомпилировать, а затем тиражировать новые программные модули. Однако, в некоторых случаях может потребоваться “более прямой” доступ к объектной модели WCF.

В Листинге 6 и Листинге 7 объект ServiceHost открывается посредством конфигурационных установок в app.config. Для того чтобы полностью "расписать" код, нужно проделать нижеследующее.

Сначала определим текущий адрес хоста:

Comment [A.A.4]: Помните о необходимости добавлять ссылку .NET на System.ServiceModel всякий раз при кодировании с использованием объектной модели WCF.

```
// C# код
IPHostEntry ips = Dns.GetHostEntry(
   Dns.GetHostName());
IPAddress _ipAddress = ips.AddressList[0];
string urlService = "net.tcp://" +
   _ipAddress.ToString() +
   ":8228/MyService";

' VB.NET
Dim ips As IPHostEntry = Dns.GetHostEntry(
   Dns.GetHostName())
Dim _ipAddress As IPAddress = ips.AddressList(0)
Dim urlService As String = "net.tcp://" + _
   _ipAddress.ToString() + ":8228/MyService"
```

Теперь создадим новый экземпляр объекта NetTcpBinding (или WsHttpBinding, или любого другого нужного нам биндинг-объекта) и установим все необходимые свойства:

```
// C# код
NetTcpBinding tcpBinding =
new NetTcpBinding();
tcpBinding.Security.Transport.ProtectionLevel =
   System.Net.Security. ProtectionLevel.EncryptAndSign;
tcpBinding.Security.Mode = SecurityMode.None;


'  VB.NET код
```

```
Dim tcpBinding As New NetTcpBinding()
tcpBinding.Security.Transport.ProtectionLevel =
   System.Net.Security.ProtectionLevel.EncryptAndSign;
tcpBinding.Security.Mode = SecurityMode.None;
```

Наконец создадим экземпляр объекта ServiceHost, и с помощью метода AddServiceEndpoint определим ссылку (type reference) на контракт сервиса (интерфейс), биндинг-объект и адрес:

```
// C# код
ServiceHost host = new ServiceHost(
   typeof(ICustomers));
host.AddServiceEndpoint(
   typeof(ICustomers), tcpBinding,
   urlService);
host.Open();

'  VB.NET код
Dim host As New ServiceHost( _
   GetType(ICustomers))
host.AddServiceEndpoint( _
   GetType(ICustomers), _
   tcpBinding, urlService)
host.Open();
```

10 Шаг 8: Построение и хостинг WCF сервиса в IIS (часть 2.1)

Те, кто разрабатывал Web-сервисы в стиле ASMX, могут задаться вопросом: каким образом осуществить то же самое в WCF: а именно, построить WCF сервис, в котором бы использовался интерфейс и бизнес-класс, но разместить его в IIS. Для этого используются, вообще говоря, аналогичные шаги, что свидетельствует об унифицированности обоих подходов.

Этот шаг (8) будет посвящен построению WCF Web сервиса, а следующий шаг – построению клиентской части с доступом к этому сервису.

Прежде всего, создадим новый проект Web-сайта с помощью шаблона (template) WCF Service (см. рисунок 1). Назовем этот проект WCFService_IISHosted.

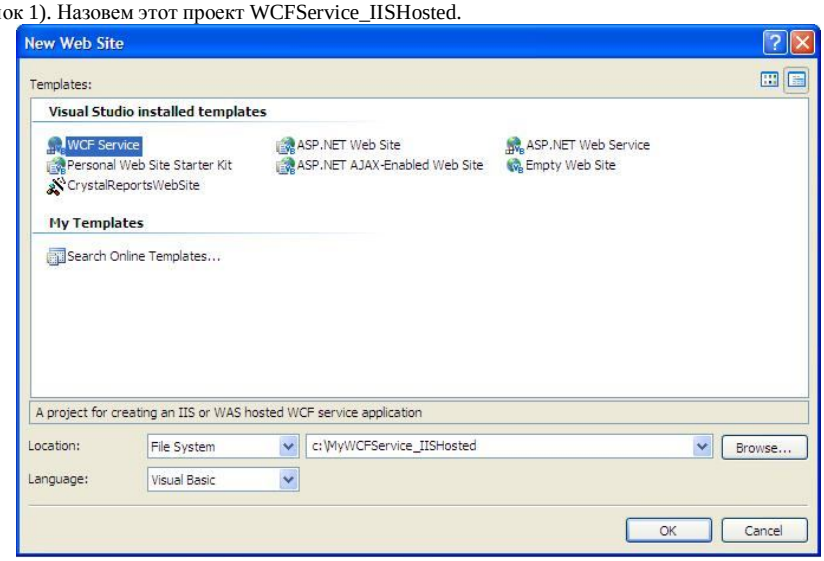

Рисунок 1. Добавление WCF сервиса. Выбираем шаблон WCF сервиса с нового диалога Web сайта.

Далее, находясь в Solution Explorer, добавим .NET ссылки на две DLL (библиотеки интерфейса и бизнес-класса, созданные нами на Шаге 4-м и Шаге 5-м: DemoInterfaces.DLL и DemoCustomerBz.DLL). После этих действий отображение Solution Explorer на экране примет вид, показанный на рисунок 2.

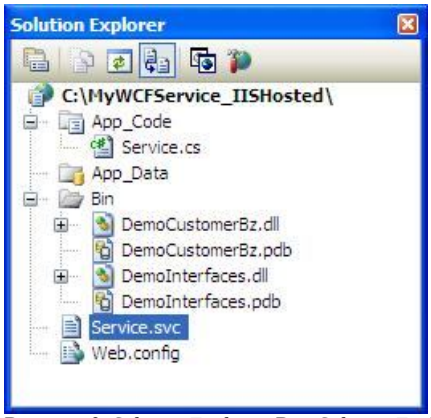

Рисунок 2. Solution Explorer: Вид Solution Explorer в Visual Studio во время создания проекта на основе WCF сервиса в IIS.

В верхней части рисунка можно заметить имя файла сервиса Service.svc. Подобно этому, до появления WCF, мы использовали бы ASMX файл. Приведем содержимое файла сервиса:

```
<% @ServiceHost Language=C# Debug="true"
   Service="MyService"
   CodeBehind="~/App_Code/Service.cs" %>
```

Нам нужно сделать в нем два изменения: ссылку сервиса установить на файл CustomerBz.DLL и убрать code-behind. (Многие онлайновые демонстрационные программы используют простой code-behind файл, который содержит код и для интерфейса, и для класса, но в данном примере показано, как задать внешнюю DLL).

```
<% @ServiceHost Language=C# Debug="true"
   Service="DemoCustomerBz.CustomerBz"
```

Следующий шаг – тестирование сервиса в браузере. Если наш Web-проект запустить из среды Visual Studio 2005 (или сделать каталог доступным вне этой среды), то, скорее всего, мы увидим экран браузера (см. рисунок 3), сообщающий нам, что "Metadata publishing is currently disabled". Включим опцию Metadata publishing, добавив следующую строку к web.config в области serviceBehavior:

```
<serviceMetadata  httpGetEnabled="true"/>
```

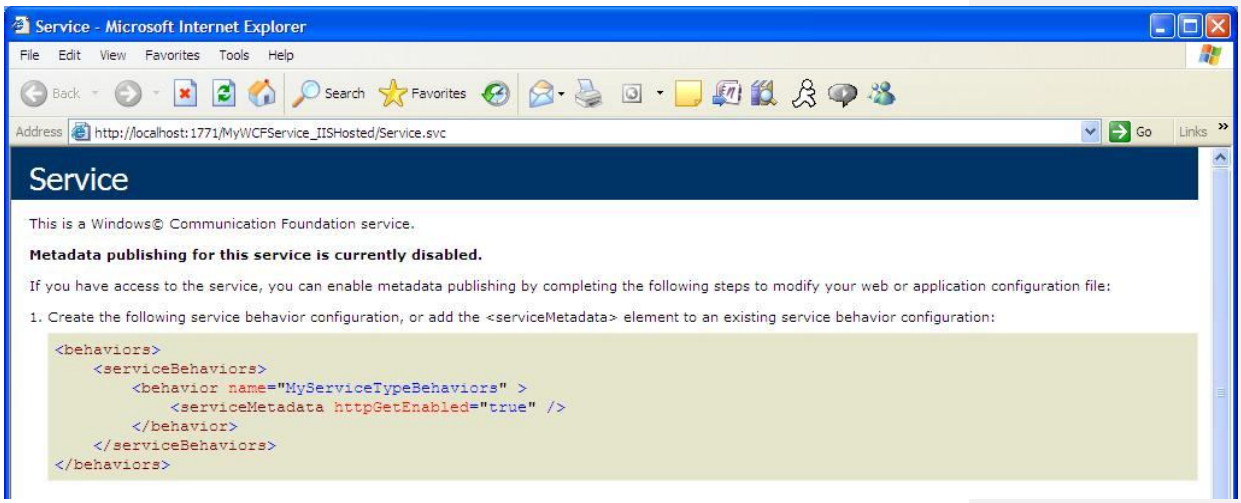

Рисунок 3. Metadata Publishing Disabled: Такое сообщение появляется, если при попытке загрузки сервиса опция “metadata publishing” отключена.

Теперь после запуска сервиса в браузере мы увидим корректное отображение (см. рисунок 4).

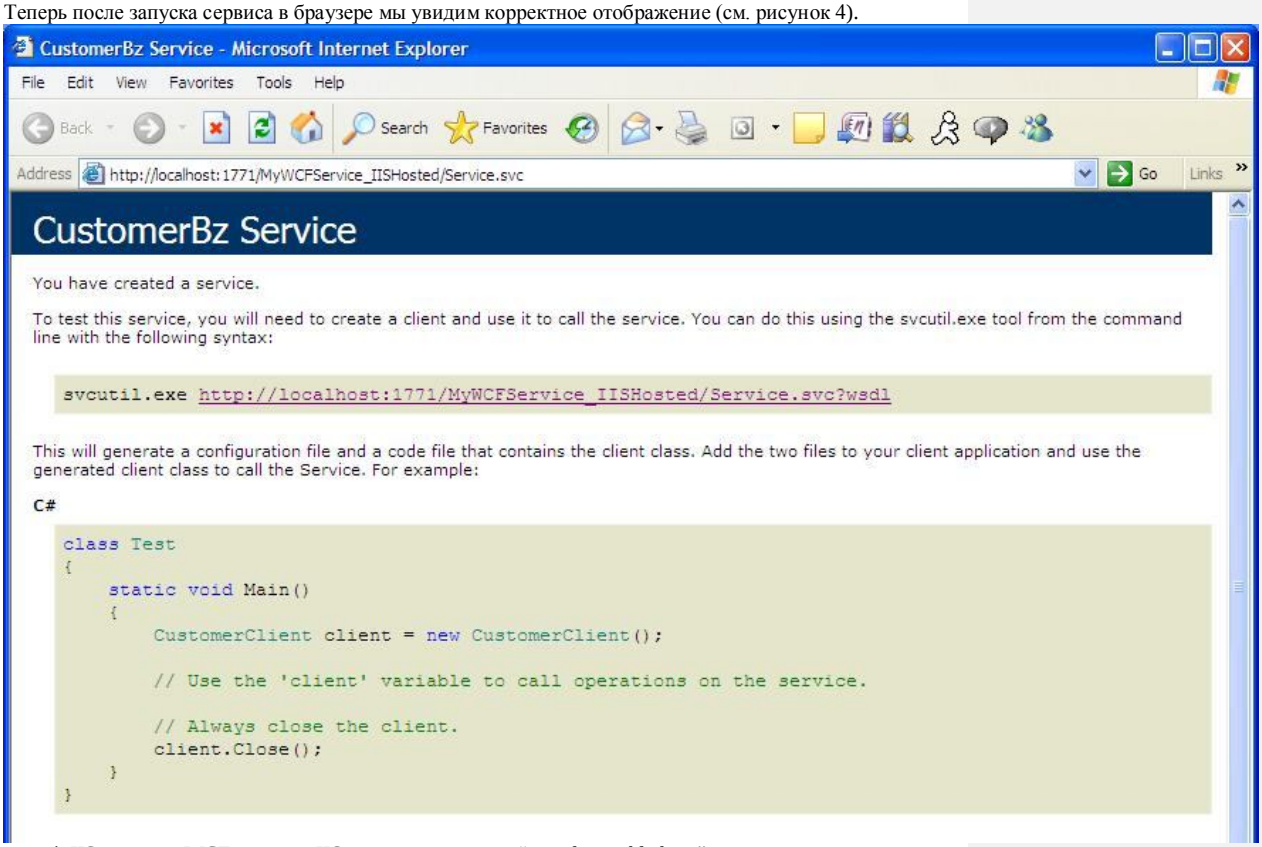

Рисунок 4. IIS хостинг: WCF сервис в IIS после разрешения “metadata publishing”.

В действительности, мы не можем запустить сервис и сразу же просмотреть результаты теста в браузере, и вообще, не рекомендуется тестировать сервис в том же месте, где мы его создали.

Следующим шагом будет создание простого .NET клиента, для которого, собственно, и предназначен этот сервис, размещаемый в IIS.

11 Шаг 9: Создание .NET клиента с доступом к WCF сервису и хостингом в IIS (часть 2.2)

Создаваемый .NET клиент будет использовать уже созданный на предыдущем шаге WCF сервис, размещенный в IIS. В этот .NET клиент мы добавим сервисную ссылку к WCF сервису, подобно тому, как до появления WCF нужно было добавлять Web ссылку.

Итак, создадим приложение Windows Forms, и в Solution Explorer выберем пункт References, нажав на правую кнопку мыши, чтобы добавить Service Reference (см. рисунок 5). Visual Studio 2005 предложит ввести адрес WCF сервиса (см. рисунок 6), а локальную ссылку на сервис возьмет по умолчанию. При желании это можно изменить, но для нашего демо-приложения оставим localhost.

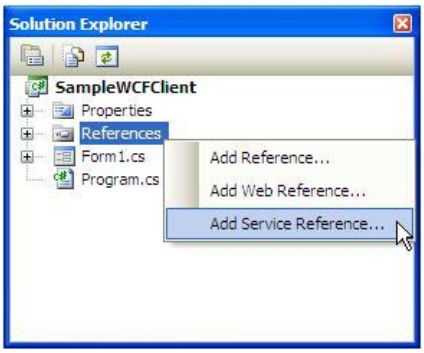

Рисунок 5. Добавление ссылки на WCF Service: кликните правой кнопкой на пункт References.

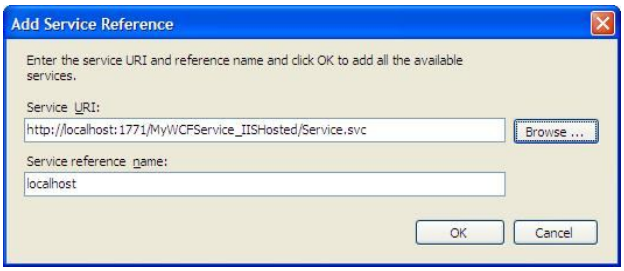

Рисунок 6. Добавление ссылки на сервис в окне диалога: заполните поля для ссылки и для имени (по умолчанию).

После добавления ссылки на сервис Visual Studio 2005 произведет три действия:

- Автоматически добавит ссылку на System.ServiceModel.
- Добавит в локальный файл app.config информацию о биндинге, основываясь на информации от сервиса.
- Добавит к WCF сервису интерфейсный прокси (proxy), доступ к которому организуется программными средствами (подобно тому, как осуществляется доступ к Web ссылке прокси).

На рисунок 7 показан Solution Explorer после перечисленных дополнений.

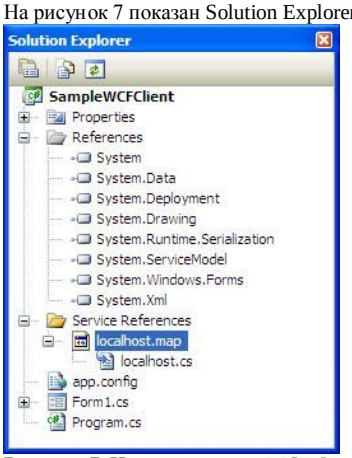

Рисунок 7. Новая ссылка на WCF Service: так выглядит Solution Explorer после добавления ссылки.

Наконец, посредством прокси объекта теперь можно будет использовать WCF сервис и вызывать классы серверной части. И, опять-таки, если вы ранее пользовались сервисами ASMX, этот код покажется вам знакомым:

```
// C# код
localhost.CustomerClient oClient =
```

```
   new localhost.CustomerClient();
string cResults = oClient.GetOrderHistory(1);

'VB.NET код
Dim Client As New localhost.CustomerClient()
Dim cResults As String = _
   oClient.GetOrderHistory(1)
```

Замечание напоследок: такой подход по существу как бы "вытягивает" ссылку на WCF сервис из IIS. В других случаях может потребоваться "выталкивание" сервисных ссылок. В комплекте Microsoft Windows SDK есть утилита svcutil.exe (сокращение от ServiceModel MetaData Utility), которая, используя метаданные сервиса, генерирует модель кода сервиса.

На рисунок 8 показана командная строка утилиты svcutil.exe. Необходимо указать полный URL для сервиса, после чего утилита генерирует два файла: прокси сервиса и файл output.config, которые затем можно включить в состав клиентского приложения.

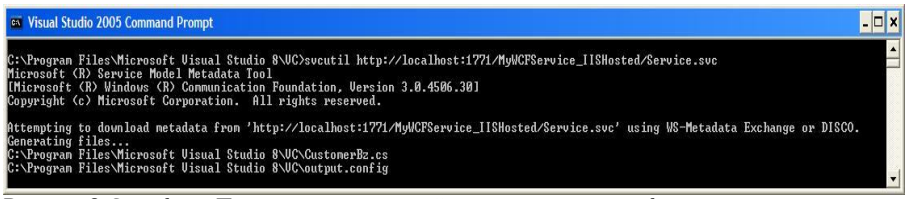

Рисунок 8. Svcutil.exe: Приглашение командной строки утилиты svcutil.exe.

12 Шаг 10: WCF сервисы под IIS для отличных от .NET клиентов (часть 3.1)—создание сервиса и конфигурации

Ниже приведен полный листинг модифицированного файла web.config для размещенного в IIS сервиса с множеством биндингов:

```
<configuration>
  <system.serviceModel>

    <services>
      <service name="MyService"
        behaviorConfiguration=
          "FoxWcfServiceBehaviors">
        <endpoint contract="IMyService"
          binding="basicHttpBinding"/>
        <endpoint contract="IMyService"
          binding="wsHttpBinding"
          address="ws" />
        <endpoint address="mex"
          binding="mexHttpBinding"
          contract="IMetadataExchange" />
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="FoxWcfServiceBehaviors" >
          <serviceDebug
            includeExceptionDetailInFaults="true" />
          <serviceMetadata  httpGetEnabled="true"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
  <system.web>
    <compilation debug="true"/>
    <authentication mode="Windows"/>
    <identity impersonate="true"/>
  </system.web>
</configuration>
```

Одна из важных особенностей WCF состоит в том, что можно определить столько конечных точек, сколько нужно для разработки, это может быть специфическая конечная точка для .NET клиента, и другая конечная точка для клиента иной природы.

13 Шаг 11: WCF сервисы для отличных от .NET клиентов (часть 3.2). Регистрация сервиса

Прежде, чем использовать сервис для отличного от .NET клиента, необходимо этот сервис зарегистрировать посредством IIS и ASP.NET 2.0. В составе .NET 3.0 Framework есть утилита ServiceModelReg.exe, которую можно использовать таким образом:

```
-- Location for ServiceModelReg.EXE
-- "\Windows\Microsoft.NET\Framework\v3.0\
      Windows Communication Foundation"
-- Syntax:
ServiceModelReg.EXE -s:W3SVC/1/ROOT/TestWCFService
```

ServiceModelReg регистрирует сервис и создает необходимые скриптовые отображения.

14 Шаг 12: WCF сервисы для отличных от .NET клиентов (часть 3.3) (Visual FoxPro)

И, наконец, можно получить доступ к WCF XML Web сервису, используя отличный от .NET клиент. Покажем, как сервис может использоваться для Visual FoxPro (VFP). Вот небольшой фрагмент кода:

```
-- VFP code to consume a WCF service,
-- in the same way as an ASMX
LOCAL loBasicHttpBinding_IMyService
  AS "XML Web Service"
LOCAL loException, lcErrorMsg, loWSHandler
loWSHand = NEWOBJECT("WSHandler",HOME() +
     "FFC\_ws3client.vcx")
loBasicHttpBinding = loWSHand.SetupClient(
   "http://localhost/WCFDemo/Service.svc?wsdl",
   "DemoService", "BasicHttpBinding_IMyService")
```

Это, в сущности, напоминает код для использования ASMX сервисов в VFP.

15 Шаг 13: Опции безопасности WCF

Сообщения SOAP посредством WCF отправляются по всем поддерживаемым протоколам (TCP, HTTP и т.д.). Следовательно, необходимо реализовать такие функции как аутентификация, шифрование и другие, относящиеся к информационной безопасности. Безопасность в WCF является достаточно емкой темой, которая выходит за рамки этой статьи.

В таблице 2 перечислены основные режимы безопасности для всех коммуникационных биндингов WCF.

Таблица 2: Опции безопасности WCF

Контракт
None
Transport
Message
Both
TransportWithMessageCredential
TransportCredentialOnly

Опции безопасности могут быть заданы в конфигурационных файлах, которые мы создавали на предыдущих шагах. Ниже приводится пример, в котором задаются требования о передаче сообщений с пользовательскими полномочиями:

```
<wsHttpBinding>
  <binding name="wsHttp">
    <security mode="Message">
      <message clientCredentialType=
         "UserName" />
    </security>
```

```
        </binding>
      </wsHttpBinding>
```

Это был последний шаг. Теперь вы убедились, что WCF позволяет быстро и без лишний усилий создавать распределенные приложения.

Статью подготовили Рувинская Виктория и Беркович Леонид

Листинг 1. Интерфейс и контракты (C#)

```
// Убедитесь что добавили System.ServiceModel как .NET reference!
using System;
using System.Collections.Generic;
using System.Text;
using System.ServiceModel;

namespace DemoInterfaces
{
    [ServiceContract]
    public interface ICustomer
    {
        [OperationContract]
        string GetCustomer(int CustomerID);
        [OperationContract]
        string GetOrderHistory(int CustomerID);
    }
}
```

Листинг 2. Интерфейс и контракты (VB.NET)

```
// Убедитесь что добавили System.ServiceModel как .NET reference!
using System;
using System.Collections.Generic;
using System.Text;
using System.ServiceModel;

namespace DemoInterfaces
{
    [ServiceContract]
    public interface ICustomer
    {
        [OperationContract]
        string GetCustomer(int CustomerID);
        [OperationContract]
        string GetOrderHistory(int CustomerID);
    }
}
```

```
Imports System
Imports System.Collections.Generic
Imports System.Text
Imports System.ServiceModel

Namespace DemoInterfaces
<ServiceContract()> _
Public Interface ICustomer
   <OperationContract()> _
Function GetCustomer(ByVal CustomerID As Integer) As String

<OperationContract()> _
Function GetOrderHistory(ByVal CustomerID As Integer)
                         As String
End Interface
End Namespace
```

Листинг 3. CustomerBz.cs (C#)

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using DemoInterfaces;

namespace DemoCustomerBz
{
    public class CustomerBz : ICustomer
    {
        public string GetCustomer(int CustomerID)
        {
            DataSet dsCust = new DataSet();
            DataTable dt = new DataTable();
            dt.Columns.Add("CustomerID", typeof(Int32));
            dt.Columns.Add("CustomerName", typeof(String));
            dt.Columns.Add("CustomerNumber", typeof(String));

            dt.Rows.Add(CustomerID,    "Customer " +
            CustomerID.ToString(),
                        "ID" + CustomerID.ToString());
            dsCust.Tables.Add(dt);

            return dsCust.GetXml();
        }

        public string GetOrderHistory(int CustomerID)
        {
            DataSet dsCustOrder = new DataSet();
            DataTable dt = new DataTable();
            dt.Columns.Add("CustomerID", typeof(Int32));
            dt.Columns.Add("OrderID", typeof(String));
            dt.Columns.Add("OrderDate", typeof(DateTime));
            dt.Columns.Add("OrderAmount", typeof(Decimal));
            dt.Columns.Add("OrderStatus", typeof(String));

            dt.Rows.Add(CustomerID, 1001,
                  new DateTime(2006, 11, 21), 1100, "Completed");
            dt.Rows.Add(CustomerID, 1523,
                  new DateTime(2007, 1, 3), 1100, "Pending");

            dsCustOrder.Tables.Add(dt);

            return dsCustOrder.GetXml();
        }

    }
}
```

Листинг 4. CustomerBz.cs (VB.NET)

```
Imports System
Imports System.Collections.Generic
Imports System.Text
Imports System.Data
Imports DemoInterfaces

Namespace DemoCustomerBz
   Public Class CustomerBz
      Implements ICustomer

      Public Function GetCustomer(ByVal CustomerID As Integer)
                                                  As String
         Dim dsCust As New DataSet()
         Dim dt As New DataTable()
```

```
   dt.Columns.Add("CustomerID", GetType(Int32))
   dt.Columns.Add("CustomerName", GetType([String]))
   dt.Columns.Add("CustomerNumber", GetType([String]))

   dt.Rows.Add(CustomerID, "Customer " +
            CustomerID.ToString(), "ID" + CustomerID.ToString())

   dsCust.Tables.Add(dt)
   Return dsCust.GetXml()

End Function

Public Function GetOrderHistory(ByVal CustomerID As Integer)
                                        As String
   Dim dsCustOrder As New DataSet()
   Dim dt As New DataTable()
   dt.Columns.Add("CustomerID", GetType(Int32))
   dt.Columns.Add("OrderID", GetType([String]))
   dt.Columns.Add("OrderDate", GetType(DateTime))
   dt.Columns.Add("OrderAmount", GetType([Decimal]))
   dt.Columns.Add("OrderStatus", GetType([String]))

   dt.Rows.Add(CustomerID, 1001, New DateTime(2006, 11, 21),
                    1100, "Completed")
   dt.Rows.Add(CustomerID, 1523, New DateTime(2007, 1, 3),
                    1100, "Pending")

   dsCustOrder.Tables.Add(dt)
   Return dsCustOrder.GetXml()

End Function
   End Class
End Namespace
```

Листинг 5. Файл конфигурации на стороне сервера.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="DemoCustomerBz.CustomerBz">
        <endpoint address="net.tcp://localhost:8228/CustomerBz"
            binding="netTcpBinding"
            contract="DemoInterfaces.ICustomer" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

Листинг 6. Активация сервиса на сервере (C#).

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.ServiceModel;
using DemoInterfaces;
using DemoCustomerBz;

namespace DemoServer
{
    public partial class Form1 : Form
```

```
    {
        ServiceHost oHost;
        public Form1()
        {
            InitializeComponent();
        }

        private void UpdateStatus(string cLabel)
        {
            this.lblStatus.Text = cLabel;
            this.lblStatus.Visible = true;
            this.lblStatus.Update();
        }

        private void btnStart_Click(object sender, EventArgs e)
        {
            using (oHost = new ServiceHost(typeof(CustomerBz)))
            {
                this.UpdateStatus( "Attempting to start Service");
                oHost.Open();
                this.UpdateStatus("Service Started");
            }

        }

        private void btnStop_Click(object sender, EventArgs e)
        {
            this.UpdateStatus("Service Stopped");
            oHost.Close();
        }
    }
}
```

Листинг 7. Активация сервиса на сервере (VB.NET).

```
' Не забудьте добавить
Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Text
Imports System.Windows.Forms
Imports System.ServiceModel
Imports DemoInterfaces
Imports DemoCustomerBz

Namespace DemoServer
End Namespace

Class Form1
Inherits Form

Private oHost As ServiceHost

Public Sub New()
InitializeComponent()
End Sub 'New


Private Sub UpdateStatus(ByVal cLabel As String)
Me.lblStatus.Text = cLabel
Me.lblStatus.Visible = True
Me.lblStatus.Update()
End Sub 'UpdateStatus
```

```
Private Sub btnStart_Click(ByVal sender As Object,
                           ByVal e As EventArgs)
Dim oHost = New ServiceHost(GetType(CustomerBz))
Me.UpdateStatus("Attempting to start Service")
oHost.Open()
Me.UpdateStatus("Service Started")
End Sub


Private Sub btnStop_Click(ByVal sender As Object,
                          ByVal e As EventArgs)
Me.UpdateStatus("Service Stopped")
oHost.Close()
End Sub 'btnStop_Click
```

Листинг 8. Файл конфигурации на стороне клиента.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <client>
      <endpoint address="net.tcp://localhost:8228/CustomerBz"
                          binding="netTcpBinding"
                          contract="DemoInterfaces.ICustomer"
                          name="Customer" />
    </client>
  </system.serviceModel>
</configuration>
```

Листинг 9. Код активации WCF сервиса через интерфейс на стороне клиента (C#).

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using DemoInterfaces;
using System.ServiceModel;
using System.IO;

namespace DemoClient
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }


        private void btnRetrieve_Click(object sender, EventArgs e)
        {
            using (ChannelFactory<ICustomer> customersFactory =
                 new ChannelFactory<ICustomer>("Customer"))
            {
                ICustomer customersProxy =
                   customersFactory.CreateChannel();
                string CustomerString =
                       customersProxy.GetCustomer(1);
```

```
                string CustomerHistory =
                    customersProxy.GetOrderHistory(1);

                this.dataGridView1.DataSource =
            this.XMLToDs(CustomerString);
                this.dataGridView1.DataSource =
            this.XMLToDs(CustomerHistory);

            }
        }

        private DataSet XMLToDs(string XMLData)
        {
            DataSet dsReturn = new DataSet();
            StringReader sr = new StringReader(XMLData);
            dsReturn.ReadXml(sr, XmlReadMode.InferSchema);

            dsReturn.AcceptChanges();

            return dsReturn;
        }
    }
}
```

Листинг 10. Код активации WCF сервиса через интерфейс на стороне клиента (VB.NET).

```
Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Text
Imports System.Windows.Forms
Imports DemoInterfaces
Imports System.ServiceModel
Imports System.IO

Namespace DemoClient
End Namespace

Class Form1
Inherits Form

Public Sub New()
InitializeComponent()
End Sub 'New

Private Sub btnRetrieve_Click(ByVal sender As Object,
                  ByVal e As EventArgs)

Dim customersFactory As ChannelFactory(Of ICustomer) =
       New ChannelFactory(Of ICustomer)("Customer")

Dim customersProxy As ICustomer =
   customersFactory.CreateChannel()

Dim CustomerString As String = customersProxy.GetCustomer(1)
Dim CustomerHistory As String =
      customersProxy.GetOrderHistory(1)

Me.dataGridView1.DataSource = Me.XMLToDs(CustomerString)
Me.dataGridView1.DataSource = Me.XMLToDs(CustomerHistory)

End Sub 'btnRetrieve_Click
```

```
Private Function XMLToDs(ByVal XMLData As String) As DataSet

Dim dsReturn As New DataSet()
Dim sr As New StringReader(XMLData)
dsReturn.ReadXml(sr, XmlReadMode.InferSchema)
dsReturn.AcceptChanges()
Return dsReturn

End Function
End Class
```