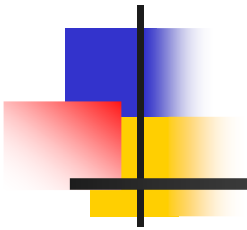
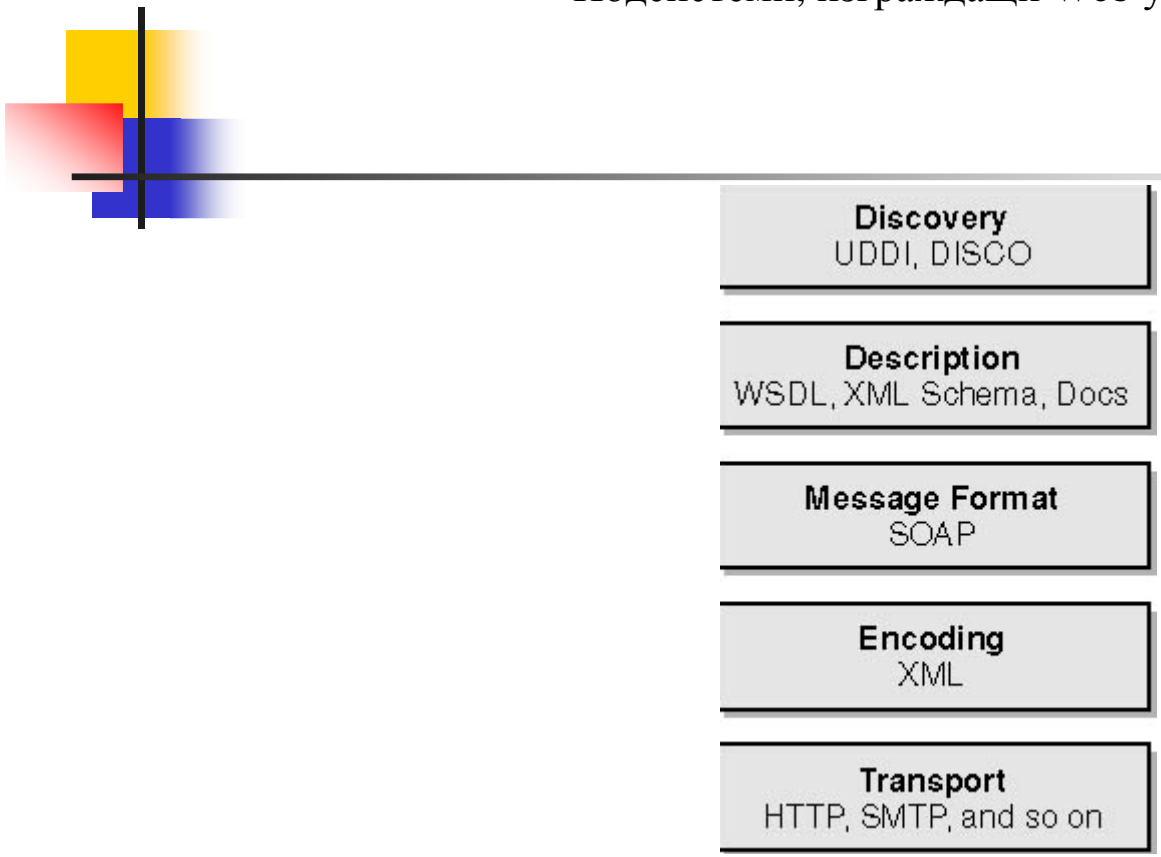
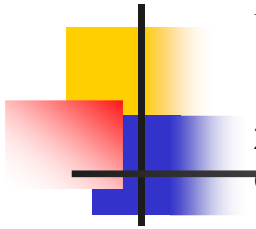
Въведение в web-services (web- услуги)



1. **Междуплатформеност.**
2. **Съвместимост с Интернет програмните технологии.**
3. **Предлагане на силно-типизирани интерфейси. Необходимо е да се избегне двусмислието относто типовете на изпращани и получавани данни към/от клиент на отдалечени Web услуги. Освен това експонираните типове в тези услуги следва добре да се съвместяват с типовете данни, дефинирани и поддържани от повечето програмни езици.**
4. **Използване и съвместимост със съществуващите стандарти в Интернет пространството, както по отношение на Интернет протоколите, така и по отношение на програмните технологии.**
5. **Поддръжка и възможност за вграждане при всеки език.**
6. **Предлаганото решение не трябва да се обвързва със задължителна инфраструктура, каквато изискваше компонентната технология. Не следва да се изисква инсталиране, закупуване и поддръжка на нови среди за разработка или изпълнение на приложение, експониращо Web услуги.**

Подсистеми, изграждащи Web услугите





1. **Откриване на функционалност**

Приложението клиент, което се нуждае от експонираната от Web услугите функционалност, трябва да притежава начин да открие местоположението на реализираната Web услуга. Този процес се нарича discovery.

2. **Описание на услугата (description)**

След като е открито местоположението на Web услугата, клиентското приложение се нуждае от допълнителна информация, за да взаимодества с нея. Описанието на Web услугата включва както описание на сигнатурата на експонираните функции, така и документация необходима за клиентската страна. COM технологията експонираше описанието на своите интерфейси в типови библиотеки (Type Library). Там в метаданнов формат се описва информация касаещи свойства, методи, изброими типове, интерфейсни идентификатори и други експонирани в компонента елементи. Аналогична информация може да се счита, че се конструира и поддържа автоматизирано от Web слугите в т.нар. WSDL документ.

3.. **Формат на съобщението**

За да могат да бъдат обменяни данни, клиентът и сървърът трябва да синхронизират начина на форматиране на съобщението. Такъв стандартизиран подход дава сигурност, че изпратените от клиента данни ще бъдат правилно интерпретирани от сървъра. Без такъв стандарт разработчиците трябва тясно да обвързват приложението с поддържащите протоколи от ниско ниво. Този стандарт на съобщението е наречен SOAP.

1. **Кодиране**

Прехвърляните данни между клиента и сървъра следва да бъдат кодирани в тялото на съобщението, споменато по-горе. Използва се универсалната технология XML (Extensible Markup Language).

5. **Транспорт**

След като съобщението, подлежащо на обмен, е форматирано и данните са кодирани в тялото му, съобщението трябва да бъде прехвърлено между клиента и сървъра с помощта на някакъв транспортен протокол. Целта е използване на установени в Интернет пространството протоколи. Популярните Hyper Text Transfer Protocol (HTTP) и Simple Mail Transfer Protocol (SMTP) са доказали своята работоспособност в различни компютърни среди. HTTP дефинира стандарт за обмен на съпбщения, заявки и отговори, а SMTP дефинира лесен за рутиране протокол, базиран на съобщения и подходящ за асинхрона комуникация.

Избор на технология за форматиране на съобщението

Често е необходимо включване на допълнителни метаданни, извън полезната информация, кодирана в тялото на съобщението. Възможно е да е необходима информация, касаеща типа на услугите, което позволява на отсрещната страна коректно да структурира заявки. XML не предоставя механизъм за различаване на полезната, съдържателна част на съобщението от асоциираната служебна информация. На помощ идва Simple Object Access Protocol (SOAP). Протоколът притежава средства за асоцииране на служебна информация към тялото на съобщението.

Всяко SOAP съобщение се дефинира като плик (envelope), който притежава тяло (body), съдържащо съдържателната част на съобщението, и заглавие (header), което съдържа асоциирани с него метаданни.

Необходим е стандартизиран начин за форматиране на дистанционното повикване (RPC), реализирано в съобщението, както и за кодиране на списъка параметри към повикваната функция. Стандарт за това се предоставя в спецификацията на SOAP услугите. В нея са описани и стандартизирани конвенции за именуване и кодиране, подходящи за процедурно ориентирани съобщения.

Механизми за описание

SOAP протоколът описва стандартен начин за форматиране на съобщението. Освен това, клиентът се нуждае от допълнителна информация за да може да структурира коректно заявка в рамките на еспортираната услуга и да интерпретира коректно отговора. XML Schema е механизмът, предоставящ средства за описание на съдържанието на съобщението. Механизмът на XML схемите позволява изграждане на собствени типове данни, както и описание структурата на съобщението.
Web услугите използват схемата не само за определяне на типовете данни, обменяни през съобщения, но изобщо за валидизация ан входящите и изходящи съобщения.

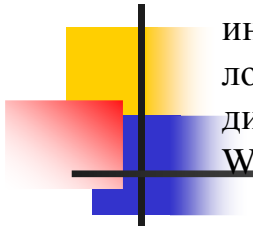
Схемата от своя страна също не е достатъчна, за да даде пълна информация, описваща Web услугата. Така например, в схемите не може да се опише шаблонът на всички съобщения, които могат да се обменят между клиента и сървъра. Клиентът може да се нуждае от информация дали да очаква отговор на изпратената си заявка или пък информация за адреса, на който са ситуирани определени Web услуги. Тези данни се предоставят от т. нар. Web Services Description Language (WSDL) документ. Това е по същество XML структуриран документ, който описва конкретна Web услуга. Вградени средства в .NET средата използват описанието на WSDL и създават проксита, подменящи реалните повиквания.

Клиентът трябва да има начин да получи описанието на услугата- т.е. начина по който да вика експонираните функции и набора аргументи с които това да стане. COM ползваше за това типовата библиотека (type library), съсредоточена в отделен файл с описание на интерфейси и сигнатури на методи.
Еквивалент на типовата библиотека и нейния описателен език IDL (Interface Definition Language) в .NET Framework е Web Service Description Language (WSDL). Чрез него се описват експонираните в WEB услугата методи, както и техните параметри. Както веяе очаквате, тази информация е структурирана с използване на XML. Форматът на WSDL е сложен и за радост не се налага да бъде изучаван. Microsoft Visual Studio.NET предоставя развойни средства за целта.

Механизми за откриване на Web услуги

За целта съществуват два общоприети механизма.

Първият от тях Universal Description Discovery and Integration (UDDI) е приет в индустрията стандарт, който осигурява централизирана директорийна услуга, използвана за локализиране мястото на определена Web услуга. UDDI позволява на потребителите да претърсят директория за определена услуга като критериите на търсене са например, име на компания, тип на Web услугата и т.н.



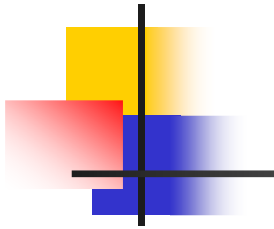
Втората възможност е центрирана върху необходимостта да се определи какви Web услуги са експонирани на определен сървър. За целта е нужен децентрализиран (различен от директорийния) механизъм за локализиране на Web услуги. Това е предназначението на DISCO протокола, разработен от Microsoft. DISCO позволява да откриете какви Web услуги са достъпни на определен компютър, преди да ги използвате. DISCO по същество е XML документ, който се структурира автоматизирано в средата на Visual Studio .NET. Той създава възможност сървърът да рекламира определен тип услуга, разположена върху него, и разположението на файловете, съдържащи описанието й.

DISCO (Discovery of WEB Service) е SOAP- базиран протокол в който се дефинира формат за описване на данните, както и протокол за тяхното извличане. Ако експонираме методи, то в сървъра следва да поместим DISCO файл за целта. Клиентите ползват този файл за намиране подходящия WSDL файл, където е детайлното описание на WEB услугата.

XML Web service discovery е процесът на локализиране и разглеждане на описанията на XML Web услугата. Клиенти на XML Web service научават, че дадена XML описана Web услуга съществува и как да си взаимодействат с нея в процеса discovery .

Файлове с разширения .wsdl, .xsd, .disco или .discomap създавани автоматично в сървъра, се използват като вход на Web Services Description Language Tool (Wsdl.exe) в процеса на създаване на Web service клиент.

Създаване на примерна Web услуга



Ще създадем функционалност, предоставяща възможност за плащания.

Нека в началото приложението бъде създадено като самостоятелен проект. (*В среда на .NET може да се създаде проект независимо на какъв език, например C#, да се структурира Web форма и съответен клас, в чиито полета потребител да зададе нужната за вализизиране на кредитна карта информация: за примера - номер на кредитна карта и срок на валидност. Натискането на бутон в Web формата би следвало да изпраща информацията за кредитната карта и да предизвиква проверка за нейната валидност.*

Кодът за това може да бъде свързан със събитието Click на бутона с име подай заявка (PlaceOrder), т. е. това събитие предизвиква стартирането на метода PlaceOrder_Click().)

Примерен код за метод от горния тип е:

```
public void PlaceOrder_Click(object sender. System.EventArgs e)

{   if(Validate(TextBox1.Text, Calendar1.SelectedDate))
                {      Status.Text = "Благодарим за заявката Ви";   }
                 else
                {   Status.Text = "Невалидна кредитна карта.";   }
}
```

Самата проверка за валидност се изпълнява от отделен метод, викан от горния:

```
public bool Validate(string cardNumber, DateTime expDate)
{   if(expDate >= DateTime.Today)
{               // някакъв алгоритъм за валидизация на кредитната карта
                return true;   }
                   else
{      return false;   }}
```

Създаване на Web услуга, базирана на горното приложение

Нека променим малко структурата на създаденото приложение. За целта ще създадем ново приложение, отново на C# за поддържане на Web услуга. Следва да укажем както името на приложението, така и URL адреса на местоположението на сървъра, който ще поддържа услугата. Тази информация ще бъде необходима при откриване на услугата от клиента.

NET средата създава необходимия клас, обхващащ Web услугата. След това копираме кода на метода Validate() от предишното приложение в току що структурирания клас. Добавяме преди него атрибут за Web метод: указва на .NET средата необходимостта от вграждане на код за експониране на метода в Web пространството, както и генериране на документация за него.

```
[WebMethod]
public bool Validate(string cardNumber, DateTime expDate)
{   }
```

Записът указва на .NET средата необходимостта от вграждане на код за експониране на метода в Web пространството, както и генериране на документация за него (във формата на Web Services Description Language - WSDL).

Друга възможност, която .NET средата предоставя при създаването на Web услуги е автоматичното генериране на документатация за услугата, както и насоки за тестването й. До тази документация може да се достигне посредством Internet Explorer чрез въвеждане на споменатия URL адрес и търсене файл с разширение .asmx. Генерираната страница изброява функциите, експонирани от Web услугата. Документацията включва и описание на генерираното пространство от имена, което служи за уникално идентифициране името на определена функция в цялото Web пространство, както и насоки за неговата промяна.

В документацията се предвидени средства за тестване работоспособността на метода.

За метода Validate() например, средата предоставя начин за тестване така че при въвеждане номер на кредитна карта и дата на валидност системата стартира експонираната в Web услуга функция Validate() и връща резултат в XML структурирано съобщение. За нашият случай резултатът е True или False, означаващ валидност или невалидност на кредитната карта.

Ето примерен asmx файл за публикуване на услуга в сървър (hello.asmx):

```
@ WebService Language="C#" Class="Hello" %>
using System;
using System.Web.Services;
public class Hello : WebService
                { [WebMethod] pub<%lic String SayHello( String Name )
{   return ( "Hello " + Name ); }      }
```

За да може клиент да ползва услугата: създаваме proxy class DLL. (подобно на COM). В .NET Framework SDK, използваме tool WebServiceUtil.exe за генериране на proxy class (Hello.cs), който компилираме до DLL.

Код в клиент

Експонираният в WEB гореописан клас (hello.asmx) е достъпен през SOAP messages. Например,ето ASP.NET страница, ползваща XML Web услуга (*hello_client.aspx):*

```
<%@ Import Namespace="nsHello" %>
 <html>
 <script language="VB" runat="Server">
  Sub Greeting( Src as Object, E as EventArgs )
     Dim h as New Hello
       lblGreeting = h.SayHello( edtName.Value )
  End Sub

  </script>
 <body>
  <form runat="SERVER">
     Your name:
   <ASP:INPUT ID="edtName" TYPE="Text" RUNAT="Server"/><br/>
     <ASP:BUTTON TEXT="Say Hello!" onclick="Greeting"/>
  </form>
<h1 ID="lblHello" RUNAT="Server"/>
</body>        </html>
```

Hello Web Service proxy class DLL –ът пакетира повикванията в SOAP съобщение и ги подава по мрежата. Hello Web Service парсва SOAP съобщението и вика *SayHello()*. Резултатът се пакетира вSOAP response message и се подава на proxy class DLL в клиента.

Промяна на началното приложение с оглед използване на отдалечена функция от Web услуги

Последната задача е да променим създаденото в началото приложение така че Web формата да генерира повикване не на локална , а на отдалечена услуга за валидизиране на потребителската карта.

За целта следва да се стартира wizard:

"Add Web Reference"

В текстово поле за адрес да се въведе URL адреса на сървъра, който хоства Web услугата. Както беше споменато търсенето в рамките на определен сървър се извършва с помощта на информацията във файл с разширение .wsdisco (освен URL адреса на хоста се указва и името на автоматично генерирания файл .wsdisco на проекат). Информация от въпросния файл позволява автоматично генериране на прокси, така че повикване на функция от Web услугата да се разглежда като че ли това е метод от нормален .NET клас.

Остава да променим в кода повикванията и да ги насочим към въпросния клас. За нашия пример вместо локалната Validate() функция ще викаме едноименната, но разположена в отдалечена Web услуга. Първо е необходимо да се създаде инстанция на proxy-класа, обхващащ Web услугата. Ако за нашия случай това е клас с име CreditCard, кодът създаващ инстанцията ще изглежда както е показано:

```
public void PlaceOrder_Click (object sender, System.EventArgs e)
{
   localhost.CreditCard cc = new localhost.CreditCard();
…
```

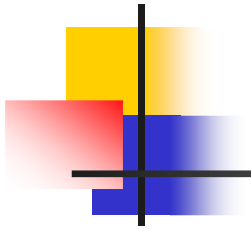
Повикването на “отдалечения” метод Validate() ще става така:

```
if(cc.Validate(TextBox1.Text, Calendar1.SelectedDate))
   {   Status.Text = "Благодарим за заявката.";       }
   else
   {   Status.Text = "Кредитната карта е невалидна.";  }
}
```

Клиентът комуникира с WEB услугата в сървъра по един от следните начини:

HTTP GET команда;
HTTP POST команда;
SOAP протокол.

Първите 2 метода са добре познати на всеки който е писал код с комуникация с WEB сървър. GET командата (която добавяше данни към низа с URL, изпращтан към сървъра) и POST командата (която изпращаше данните в тялото на HTTP съобщението), позволяваха комуникация с WEB сървъра от произволен език.
Третият метод използва протокола SOAP, който позволява обмен на по-голям обем и по-сложно структурирана инфорамция в сравнение с HTTP командите



SOAP

В основа на Web услугите е Simple Object Access Protocol (SOAP), който предоставя стандартизиран начин за пакетиране на съобщенията. SOAP доби популярност в последните години, тъй като облекчава начина на повикване на RPC процедури от клиент, когато те са разположени на отдалечен сървър. Вероятно основна причина за популярността е приемането на протокола от основните софтуерни играчи като Microsoft, IBM, Sun Microsystems, SAP, Ariba. По-долу изброяваме предимства на SOAP:

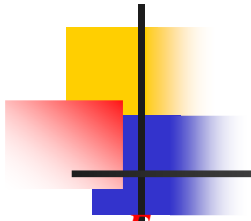
1. *Не е привързан към определен език*. SOAP не предоставя API среда, така че неговата функционална интерпертация се реализира на конкретния език и платформа.
2. *Не е привързан към определен транспортен протокол*. Действително спецификацията на SOAP описва как следва да бъде привързан към HTTP, но дотолкова доколкото SOAP съобщението не е нищо повече от един XML документ, той може да бъде транспортиран с помощта на всеки протокол, способен да предава текстова инфорамция.
3. *Не изисква поддръжка от разпределена инфраструктура* – това, което беше отличителен белег в COM технологията.
4. *Позволява междуплатформен пренос*. Така например, приложение, работещо в PC, може да комуникира със сървърно приложение, работещо на Mainframe, да подава и получава XML структурирана информация през HTTP.

Анатомия на SOAP съобщението

```
<?xml version= “1.0”?>

<soap:Envelope xmlns:soap=”http://schemas.xmlsoap.org/soap/envelope/”>

<soap:Header>
        <!—Специфична за заглавната част информация се поставя тук -->
        <To>Иван</To>
        <From>Петров</From>
</soap:Header>
<soap:Body>
        <!—Самото съобщение -->
</soap:Body>
```

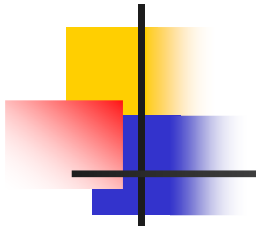


Елементът header може да съдържа и информация, позволяваща:

1. получателят да проверява (да извършва автентикиция) самоличността на изпращащия съобщението.
2. повишаване сигурността на съобщението. Тя е изработена на база съдържанието на съобщението от изпращащия и кодирана в header-а за проверка от получател.
3. рутиране: ако съобщението трябва да се изпраща към много дестинации, те са изброени в header-а.
4. реализиране на заплащане. Ако получател на съобщението предоставя услуги срещу заплащания, той вгражда информация за това в header-а.

Елемент body

Валидно **SOAP** съобщение притежава само един елемент **body**. Той съдържа неговата същност. Няма ограничение за начина на кодиране на информация в тялото. Разбира се, тя следва да води до валиден, резултатен **XML** документ.



Пример за SOAP RPC съобщение

Едно от основните предназначения на SOAP е да създаде стандартизиран начин за дистанционно извикване на процедури през Интернет при спазване на XML и HTTP. SOAP спецификацията, така както е стандартизирана през 2001 г. не налага начин на кодиране на RPC съобщенията. Разработчикът е свободен да създаде и своя кодировка на RPC повикванията.

По-долу е показан стандартен метод за кодиране на SOAP съобщение в RPC стил. За да се облекчи процесът на заявка и отговор, изискван от RPC, е необходимо структуриране на две SOAP съобщения – едно за заявка и едно за отговор.

Ето една заявка, в която C# функция, събираща две целочислени стойности, се кодира като XML SOAP съобщение:

Ф-ия:

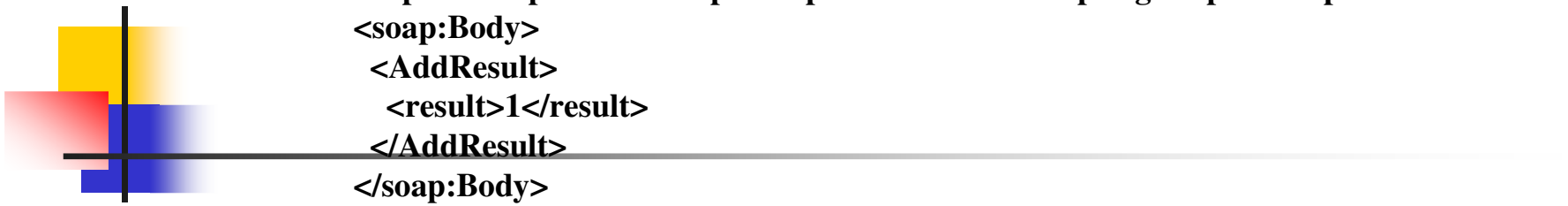
```
public int Add(int x, int y)
{
   return x + y;
}
```

XML съобщението – заявка

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Add>
 <x>1</x>
      <y>2</y>
    </Add>
  </soap:Body>
</soap:Envelope>
```

XML съобщение – отговор

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <AddResult>
      <result>1</result>
    </AddResult>
  </soap:Body>
</soap:Envelope>
```



SOAP кодировки

Стандартът дефинира начин, по който данни от различен тип се сериализират в SOAP съобщение. SOAP спецификацията указва начина на кодиране на:

- Прости типове като низове, цели числа, час, дата и т.н.
- Съставни типове като структури и масиви, сечения на масиви и дисперсни подмножества на масиви;
- Предаване на указатели.

Типове

Simple Types integers, strings, floating-point числа – директно се кодират като SOAP елементи

Compound Types

C structure:

```
struct tag_Person
{ string name;          double age;} Person;
```

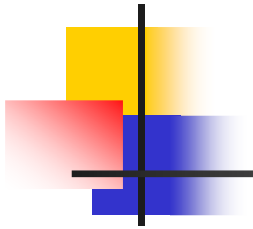
Ето съответстващ SOAP код (compound_type.xml):

```
<Person xmlns="someURI">
 <name>Harry</name>
  <age>37</age>
 </Person>
```

Масиви

SOAP-ENC:arrayType атрибут се използва за отличаване. Ето (array_type.xml)масив от структури:

```
<PersonList SOAP-ENC:arrayType="Person[2]">
  <Person>
   <name>Tom</name>
    <age>27</age>
   </Person>
   <Person>
   <name>Dick</name>
   <age>30</age>
  </Person>     </PersonList>
```

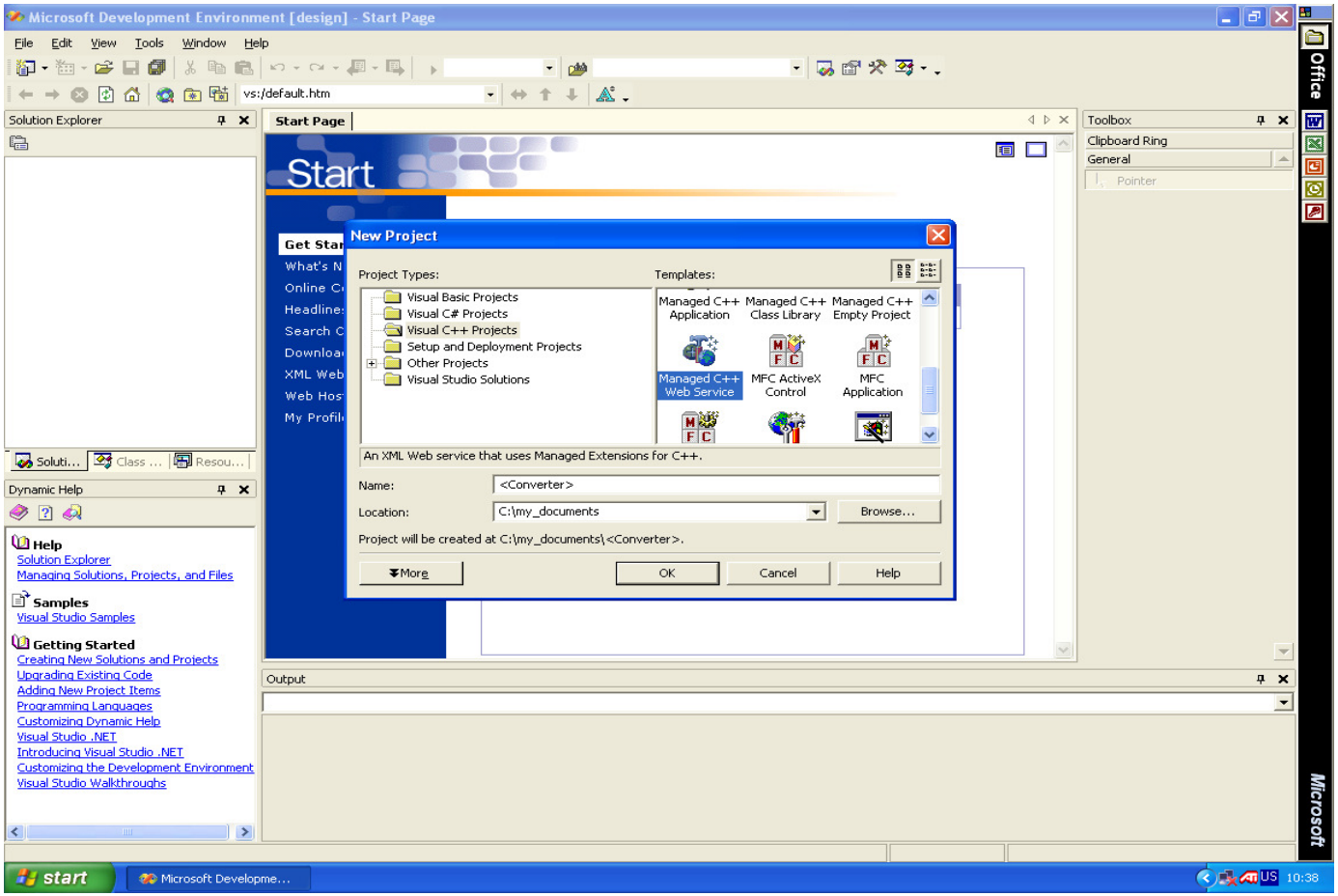


предимства на SOAP за привързване с HTTP :

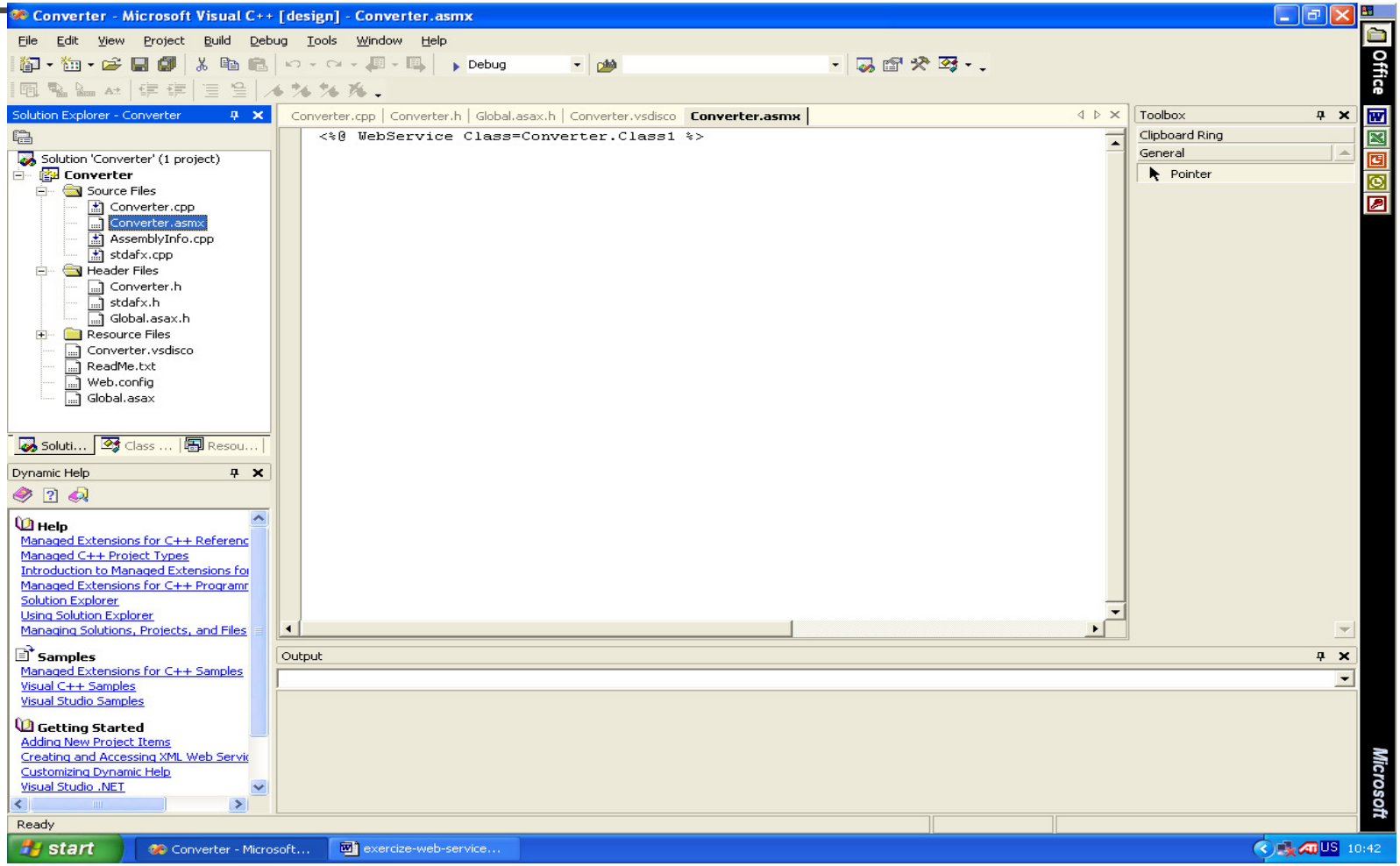
1. ***Предаването не се ограничава от firewall защитите***. Това не се отнясяше за DCOM технологиите. Повечето firewall поддържат порт 80 и отворен HTTP трафик.
2. ***Наличие на стабилна поддържаща инфраструктура***. Много технологии са развити за поддръжка на HTTP базирани приложения. Това се използва от SOAP.
3. ***HTTP е структурно съвместим с RPC повикванията***, тъй като всяка заявка и в едното и в другото е съпътствана от съобщение отговор.
4. ***Протоколът е отворен.*** Практически всички мрежово-ориентирани технологии поддържат HTTP.
5. ***Структурно сходство***. Подобно на HTTP, протоколът се състои от две секции – тяло (body) и заглавия (header). Това позволява оптимално разполагане на SOAP съобщението в HTTP.

Малко практика – екранни форми за web-разработка

Създава се проект в развойната среда. (Managed C++ Web Service project). Именоваме проекта напр.Converter



2. Разглеждайки породените в проекта файлове, забелязвате файл с разширение .asmx. В този файл се съдържа информация , необходима на Web сървъра за определяне в кой клас е реализацията на Web услугата.



Съдържанието на файла е следното:

```
<%@ WebService Class=Converter.Class1 %>
```

Следователно Web сървъра знае, че Web услугата Converter е реализирана в Converter.Calss1 (т.е клас Class1 в пространството на имена Converter). Class1 не е описателно име и затова го подменете например с TemperatureConverter така:

```
<%@ WebService Class=Converter.TemperatureConverter %>
```

Нека разгледаме и породения заглавен файл Converter.h. В него може да видим вече дефинирано горното пространство от имена. Тук също следва да променим Class1 с TemperatureConverter. Останалата част на .h файла е обяснена по-долу.

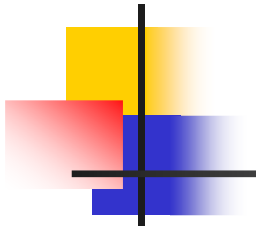
```
// Converter.h

#pragma once
#using <System.Web.Services.dll>

using namespace System;
using namespace System::Web;
using namespace System::Web::Services;

namespace Converter
{
          [WebServiceAttribute(
                        Namespace="http://VCSBS/WebServices/",
                        Description="Web Service used for converting temperatures")]
     public __gc class TemperatureConverter : public
               WebService
     {
    public:
               [WebMethod(Description=
                        "Converts temperatures from Fahrenheit to Celsius")]
               double ConvertF2C(double dFahrenheit);
               [WebMethod(Description=
                        "Converts temperatures from Celsius to Fahrenheit")]
               double ConvertC2F(double dCelsius);
       };      }
```

Проектът съдържа и файл .vsdisco, използван за публикуване. Той се ползва за разпознаване на Web услугата от отдалечен клиент.

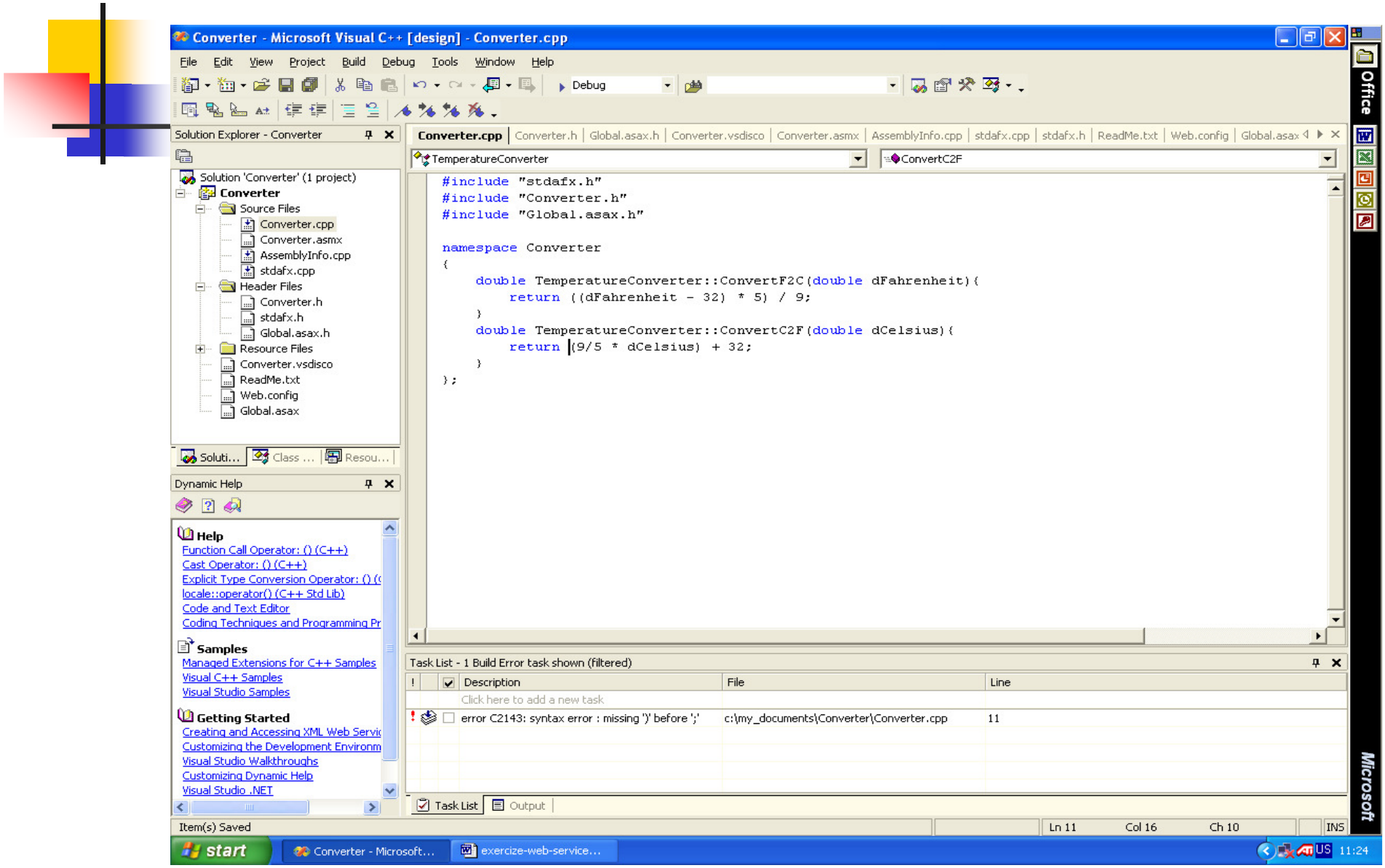


Част от кода е заграден в []. Това са т.нар. атрибути, възприети в .NET технологията. Атрибутите са пояснения, добавящи информация към данните в тази развойна среда. Данни, от какъвто и да е тип, с прикачена към тях допълнителна информация се наричат метаданни. Очевидно, информацията е от такъв характер, че да не може да се зададе със средствата на езика за програмиране. Каква може да е тази информация? Например, правилно функциониращ COM обект (това е всъщност тип) изисква допълнителна информация пазена в регистъра на Windows (това не е добро решение- тази информация се пази отделено от обекта и може да бъде повредена от код, извън него). Друга подобна информация касае версиите на компонент, например.
В .NET средата, компилаторът прикачва метаданните (с техните атрибути) към кода и при зареждане или изпълнение се ползва информацията, заложена в тях.

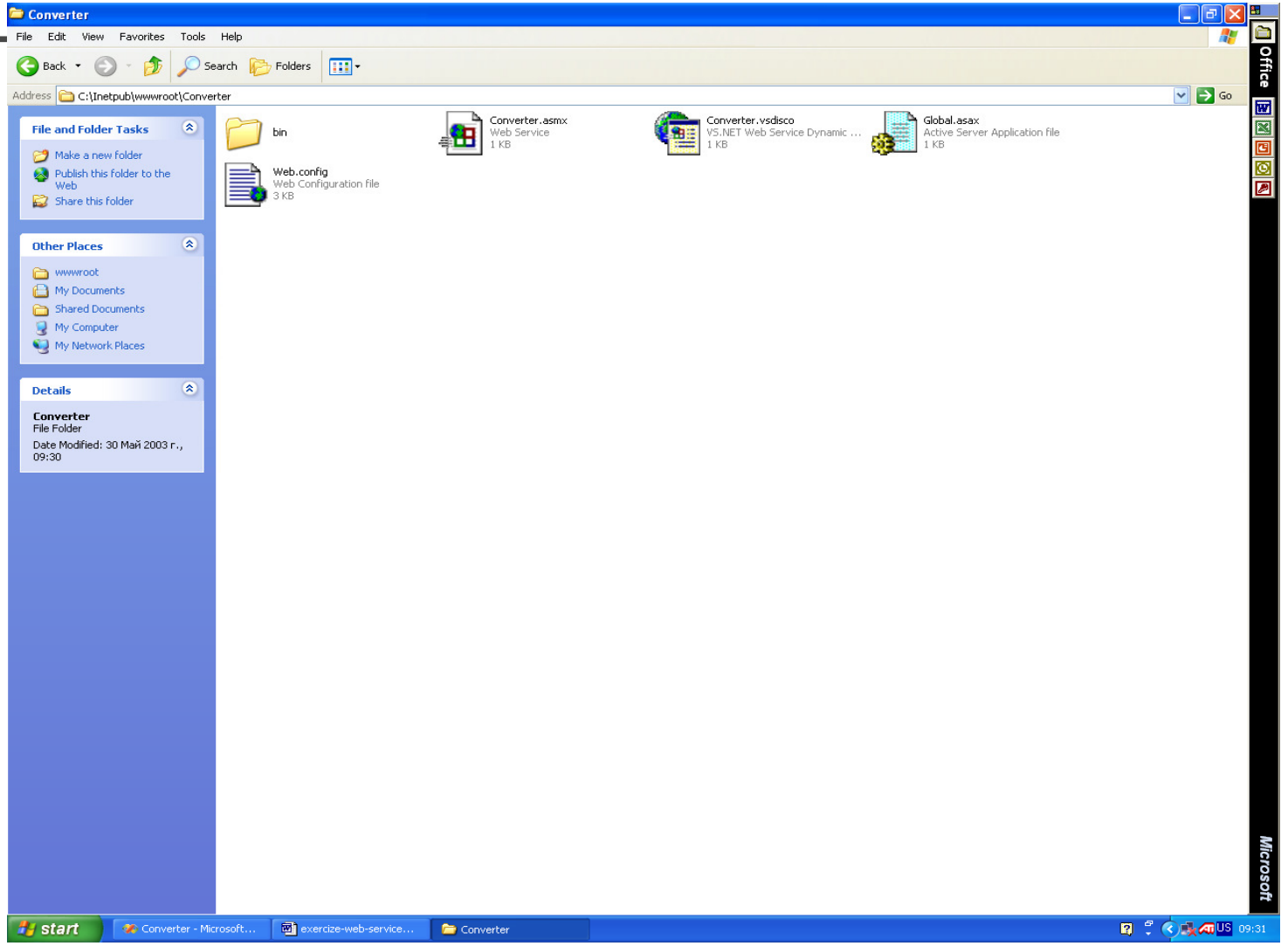
В Converter.cpp добавяме наш код за двете функции, които ще експонираме: ConvertC2F() и ConvertF2C().

```
#include "stdafx.h"
#include "Converter.h"
#include "Global.asax.h"

namespace Converter
{
    double TemperatureConverter::ConvertF2C(double dFahrenheit){
            return ((dFahrenheit - 32) * 5) / 9;
    }
    double TemperatureConverter::ConvertC2F(double dCelsius){
            return (9/5 * dCelsius) + 32;
    }
};
```


Converter - Microsoft Visual C++ [design] - Converter.cpp
File Edit View Project Build Debug Tools Window Help
Debug
Solution Explorer - Converter
Solution 'Converter' (1 project)
Converter
Source Files
Converter.cpp
Converter.asmx
AssemblyInfo.cpp
stdafx.cpp
Header Files
Converter.h
stdafx.h
Global.asax.h
Resource Files
Converter.vsdisco
ReadMe.txt
Web.config
Global.asax
Dynamic Help
Help
Function Call Operator: () (C++)
Cast Operator: () (C++)
Explicit Type Conversion Operator: () (
locale::operator() (C++ Std Lib)
Code and Text Editor
Coding Techniques and Programming Pr
Samples
Managed Extensions for C++ Samples
Visual C++ Samples
Visual Studio Samples
Getting Started
Creating and Accessing XML Web Servic
Customizing the Development Environm
Visual Studio Walkthroughs
Customizing Dynamic Help
Visual Studio .NET
Converter.cpp
Converter.h
Global.asax.h
Converter.vsdisco
Converter.asmx
AssemblyInfo.cpp
stdafx.cpp
stdafx.h
ReadMe.txt
Web.config
Global.asax
TemperatureConverter
ConvertC2F
#include "stdafx.h"
#include "Converter.h"
#include "Global.asax.h"
namespace Converter
{
double TemperatureConverter::ConvertF2C(double dFahrenheit){
return ((dFahrenheit - 32) * 5) / 9;
}
double TemperatureConverter::ConvertC2F(double dCelsius){
return (9/5 * dCelsius) + 32;
}
};
Task List - 1 Build Error task shown (filtered)
Description
File
Line
Click here to add a new task
error C2143: syntax error : missing ')' before ';'
c:\my_documents\Converter\Converter.cpp
11
Task List
Output
Item(s) Saved
Ln 11
Col 16
Ch 10
INS
start
Converter - Microsoft...
exercize-web-service...
US
11:24
Office
Microsoft

. Изграждаме (Build) проекта. Освен че извършва компилация, развойната среда вгражда Web услугата в Web сървър (е, разбира се, такъв следва да е инсталиран). Можем да проверим дали услугата е правилно инсталирана като погледнем кореновата директория на сървъра – Inetpub\wwwroot. Следва да видите поддиректория с името на проекта



Използване на Web услуга

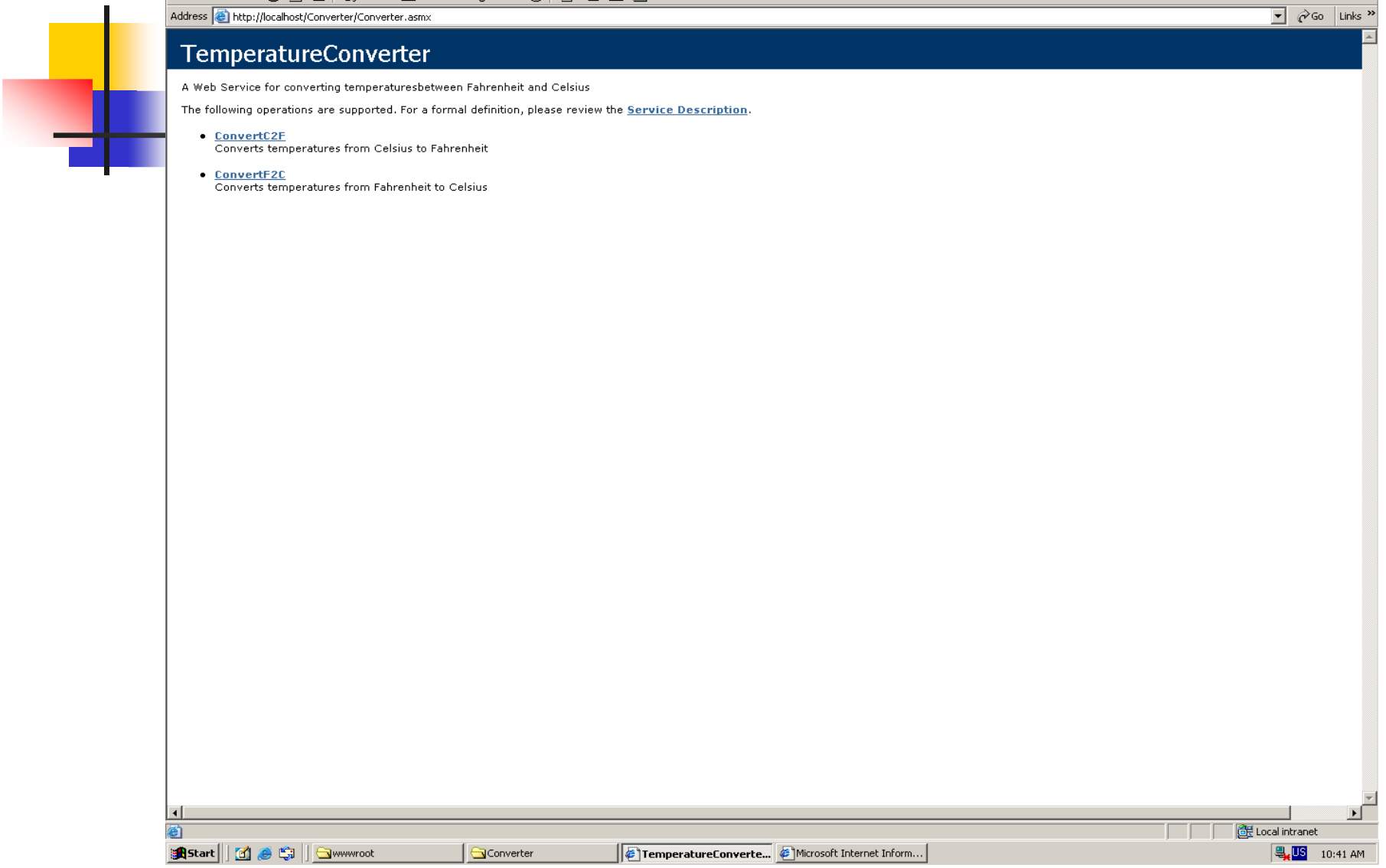
1. Използване на Web услуга от браузър

Можете да ползвате Internet browser за извикване на методите, експонирани в сървър като Web услуга.Те ще се изпълнят в него и за вас са все едно отдалечени процедури (Remote Procedure).

- **Стартирате Internet Explorer**
- **Въвеждате URL адреса, където се намира файла .asmx за вашата Web услуга. Ако искаме да ползваме току що създадената услуга Converter, инсталирана на локалния компютър, то следва да въведем като URL:**

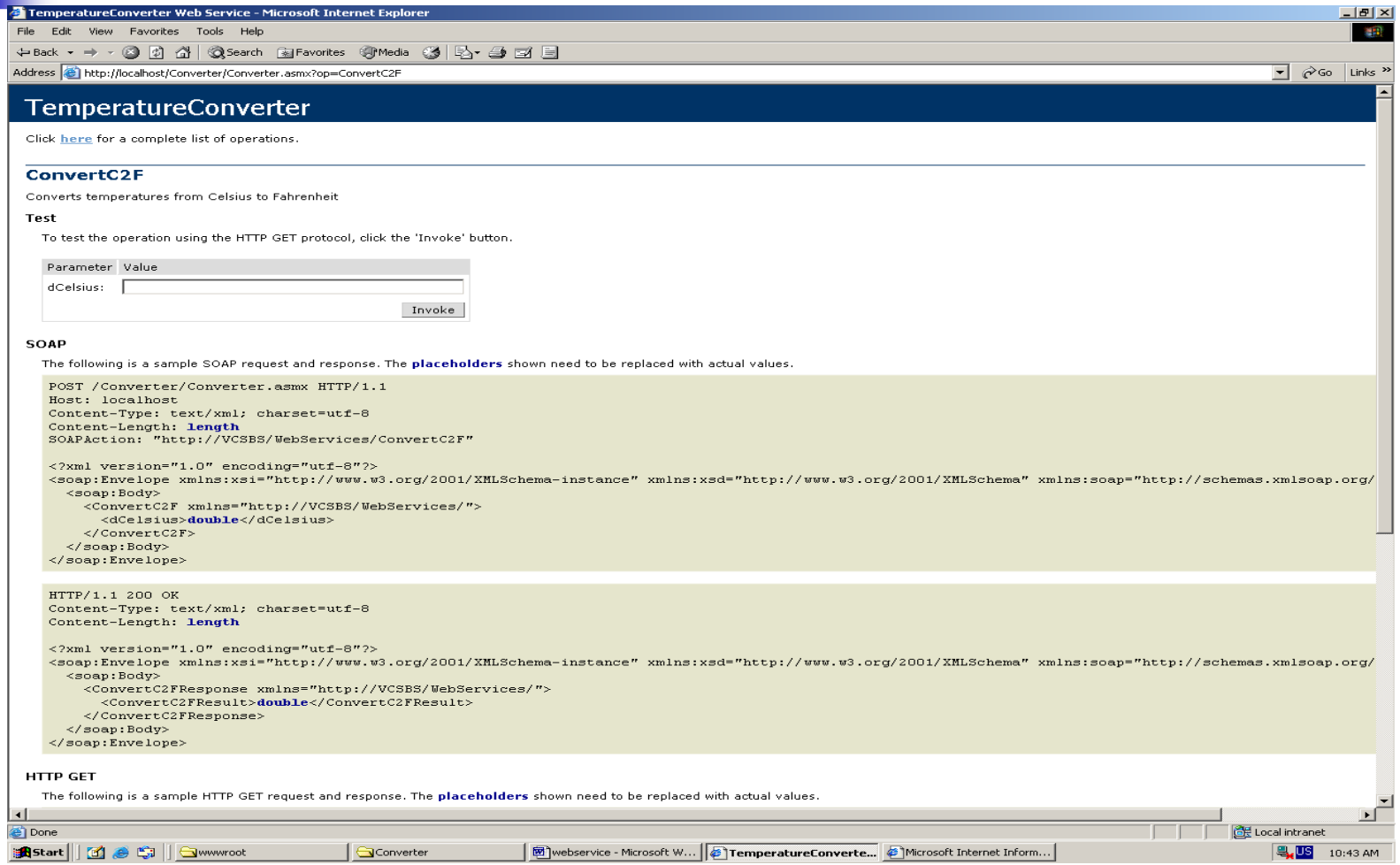
http://lokalhost/Converter/Converter.asmx

Web сървърът ще ви върне детайлна информация за тази услуга и за това какви методи тя експонира:


TemperatureConverter Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/Converter/Converter.asmx
Go Links
TemperatureConverter
A Web Service for converting temperaturesbetween Fahrenheit and Celsius
The following operations are supported. For a formal definition, please review the Service Description.
ConvertC2F
Converts temperatures from Celsius to Fahrenheit
ConvertF2C
Converts temperatures from Fahrenheit to Celsius
Local intranet
Start
wwwroot
Converter
TemperatureConverte...
Microsoft Internet Inform...
10:41 AM

Забелязвате, че описателната част(двете конструирани функции) е извлечена от атрибутните стойности на метаданните.

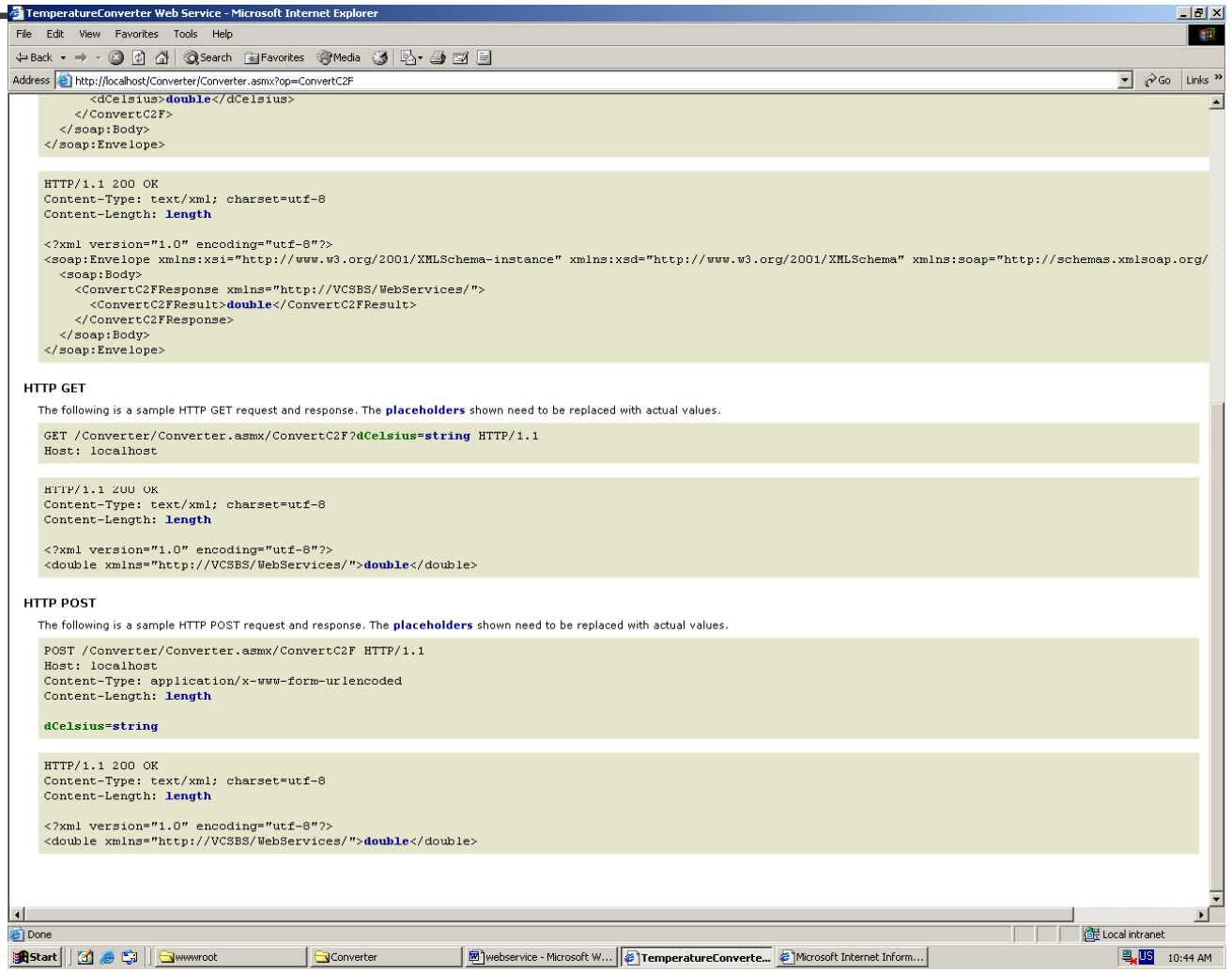
- **Избирате единия от двата експонирани метода. Това предизвиква пораждане на HTTP GET команда към сървъра. Сървърът създава вътрешен клиент на услугата и изпраща към него следната информация:**



Клиентът вижда HTML страница с описание на услугата, както и с начина на тестване .

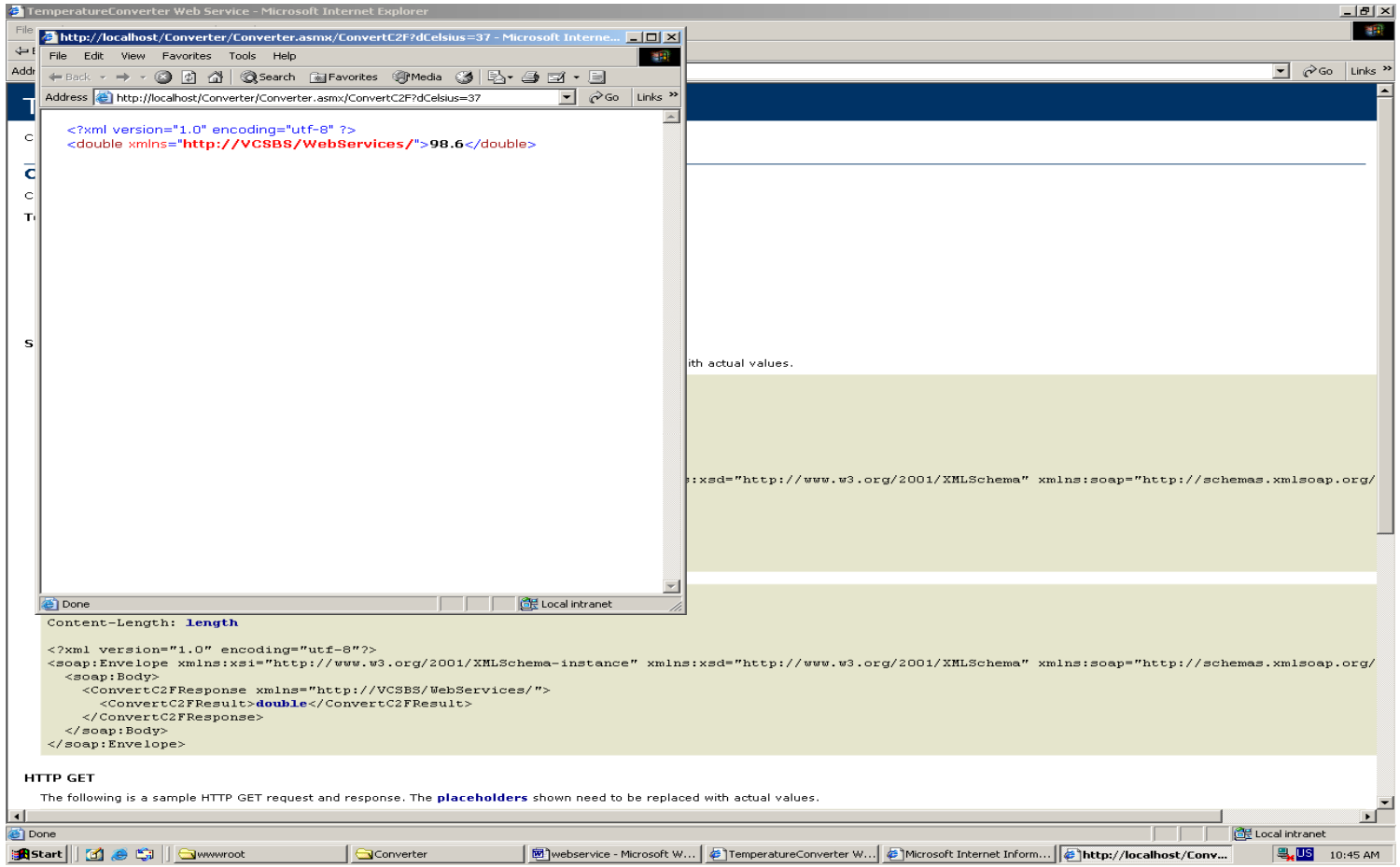
- **Тествате работоспособността на метода от Web услугата: въвеждате стойност и селектирате Invoke.**

Методът ще бъде изпълнен в сървъра и ще получите XML форматиран резултат



<?xml version=”1.0” encoding=”utf-8” ?>
<double xmlns=http://VCSBS/WebServices/>44</double>

Върнатият елемент е типизиран като double и е указано пространството на имената за услугата



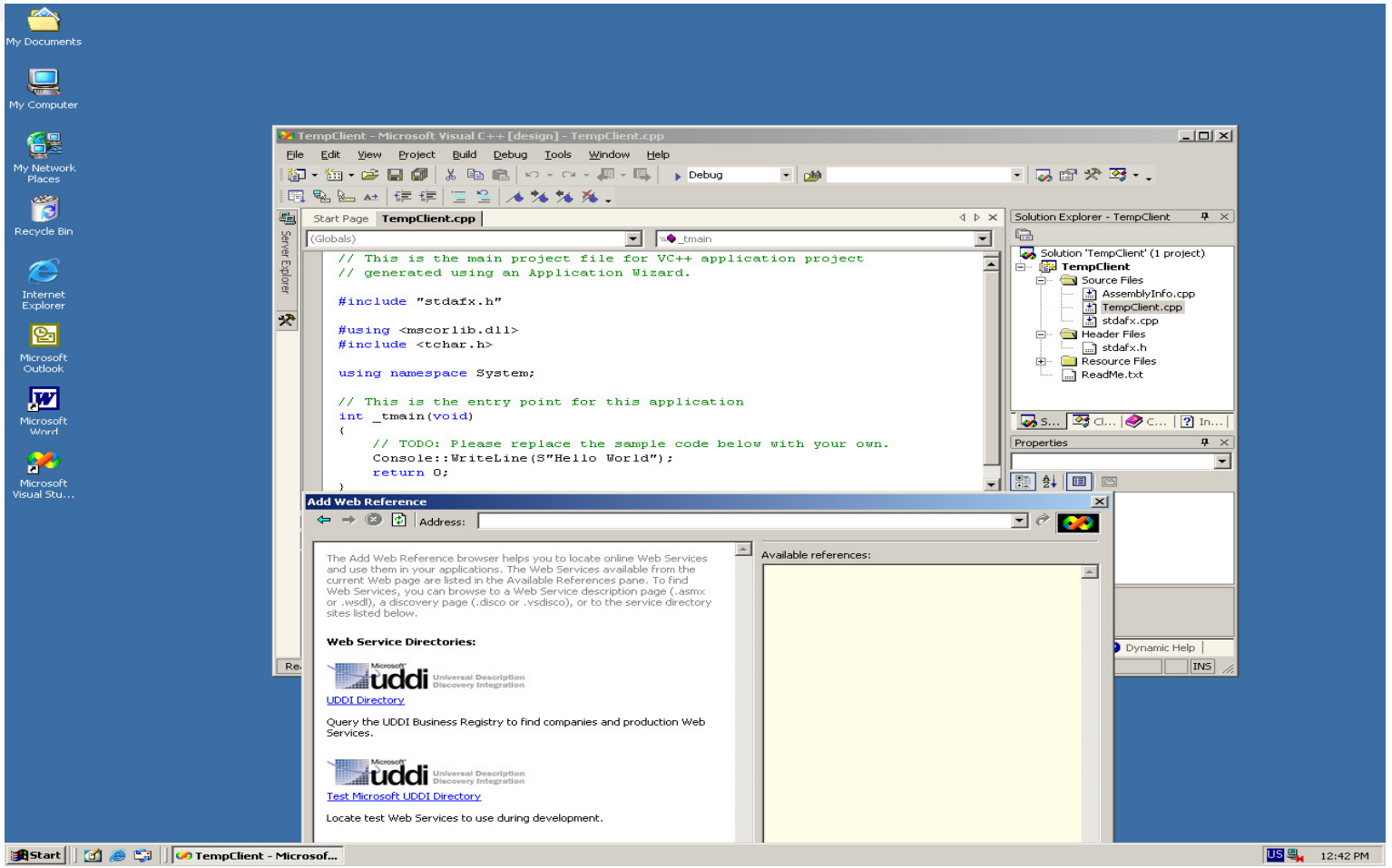
1. **Използване на Web услуга в клиентски код**

За да използвате Web услуга, първо трябва да сте я локализирали и да имате информация как да я ползвате.
Visual Studio.NET използва Web References за тези цели
След откриване на интересуващата ни услуга, Studio.NET използва услугата именована DISCO за извличане на WSDL файла, който описваше web услуга.
Добавяне на Web Reference към проекта използва информацията от WSDL за създаване на proxy клас, който подменя повикванията към дистанционно експонираните методи с повиквания към фиктивни, едноименни, но локални процедури.

Ето практическата последователност:

1. **Създавате нов проект (Managed C++ Application) с име например TempClient.**

В Project меню избирате Add Web Reference. Появява се диалогов екран Add Web Reference



[ожете да локализирате Web услуга по 3 начина:

1. ако знаете URL адрес, поставяте го в горната част;

- 2. ако услугата е регистрирана с UDDI service (Universal Description, Discovery and Integration – механизъм, разработен от Microsoft за експониране от страна на сървъра или локализиране от страна на клиента на Web услуга) то кликвате в бутона UDDI.

3. ако услугата е разположена на локалния сървър, избирате опция Web References On Local Server (това е за нашия пример). Листват се всички локално регистрирани услуги. Информацията се взима от DISCO файла на локалния сървър. От XML кода в левия панел се виждат референции към DISCO файлове на отделни услуги.

избираме от десния панел линк към файла Converter.vxdisco. Изобразява се съдържаната в него информация, отново в XML и в потребителски вид.

Разгледайте детайлната информация, която се изобразява при избор на някоя от възможните опции.

- **изберете бутон Add Reference за създаване на proxy. Това предизвиква генериране на нов файл (WebService.h) към проекта. В него са подвключени необходимите за поддръжка на Web услугата DLL файлове, включително и Converter.dll, съдържащ код на proxy.**

Д добавете в TempClient.cpp, в main():

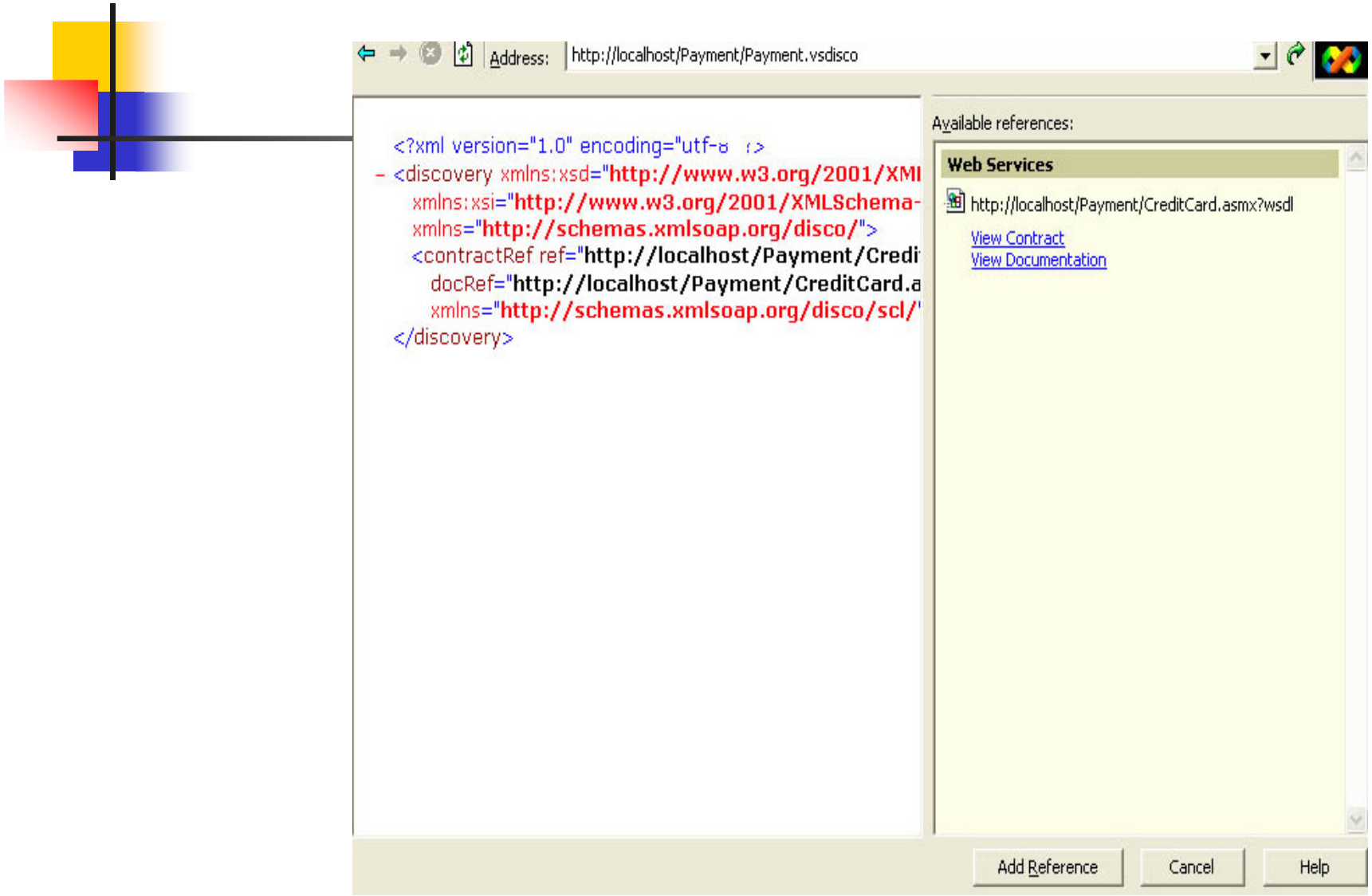
```
#iinclude “stdafx.h”
#include “WebService.h”
```

В всъщност кода на proxy, който е реферирaн от Converter.dll е генериран на C# и файла Converter.cs се е добавил към директорията на проекта и се подвключва автоматично в build процеса.

- **добавете код към main функцията за да реализирате повикване на експонираните от Web услугата външни функции:**

Proxy класът е добил име на клиентския проект → следователно е TemperatureConverter, а не Converter.
Виждаме, че повикването на експонираните функции е като че ли те са локални. Всъщност те могат да са навсякъде из Internet, могат да са писани на всякакъв език и могат да се ползват от всякакяв език.

Ето и някои от аналогичните прозорци за развития в лекцията пример:


Address:
http://localhost/Payment/Payment.vsdisco
<?xml version="1.0" encoding="utf-8" ?>
- <discovery xmlns:xsd="http://www.w3.org/2001/XMI
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
xmlns="http://schemas.xmlsoap.org/disco/">
<contractRef ref="http://localhost/Payment/Credi
docRef="http://localhost/Payment/CreditCard.a
xmlns="http://schemas.xmlsoap.org/disco/scl/"
</discovery>
Available references:
Web Services
http://localhost/Payment/CreditCard.asmx?wsdl
View Contract
View Documentation
Add Reference
Cancel
Help