

# 12. POGLAVLJE

## Web-servisi

### U ovom poglavlju:

- Standardi vezani uz web-servise
- Stvaranje web-servisa
- Opcije web-metoda
- Isprobavanje web-servisa u pregledniku
- Korištenje web-servisa u .NET aplikacijama
- Ručno i automatsko generiranje proxy-klasa

**D**osad ste naučili izrađivati *desktop* aplikacije i web-aplikacije, a sad slijedi prava poslastica – web-servisi. Krenimo odmah s objašnjenjem: na web-servise se može gledati kao na svojevrzne potprograme smještene na Webu. Dakle, radi se o programima smještenima na nekom web-poslužitelju (baš kao i obične web-aplikacije) koje možete koristiti putem standardnog protokola HTTP.

Njima tako možete pristupati putem Weba, koristiti ih, zadavati im upite, a oni će vam na isti način vraćati odgovore. No web-servisi nisu izmišljeni da bi s njima direktno komunicirao korisnik, već neki drugi program. Kao što je rečeno, na njih se može gledati kao na potprograme koje će druge aplikacije pozivati i koristiti u svom radu. Zahvaljujući činjenici

### III. DIO: DIJELOVI .NET-A

da se komunikacija s web-servisima, kao što im i ime kaže, odvija preko Weba, njih mogu koristiti baš svi.

Dajmo odmah jedan praktičan primjer: banka objavi svoj web-servis koji ima jednostavan zadatak – vraćati tečajnu listu za određeni dan. Web-servis može primiti jedan parametar, i to datum za koji se traži tečajna lista. No sad tu tečajnu listu može koristiti bilo koja aplikacija s pristupom na Internet – jednostavno se poveže s web-servisom, zada mu upit i dobije odgovor s tečajnom listom koju tad može prikazati u svojoj aplikaciji.



**Aplikacije Microsoft Officea 2003 u sebi također imaju ugrađenu podršku za vanjske web-servise, pa tako direktno iz Worda ili Excela možete pristupati dostupnim web-servisima. U Hrvatskoj je već načinjeno nekoliko takvih servisa, pa tako možete pretraživati tečajne liste, vozni red vlakova, pozivne brojeve gradova i država itd.**

U ovom poglavlju pokazat ćemo vam koliko je lako izrađivati web-servise i koliko je lako koristiti ih i pozivati iz vlastitih aplikacija. Dakle, pokazat ćemo dvije strane priče – prvo ćemo se postaviti u kožu autora i programera web-servisa, a zatim u situaciju programera aplikacije koja želi iskoristiti neki web-servis.

Web-servisi ne trebaju biti javni odnosno ne trebate ih dati svima na korištenje. Najčešće ćete izraditi vlastite web-servise koje ćete koristiti samo iz vlastite aplikacije, bila ona web-aplikacija, prozorska aplikacija ili pak aplikacija za mobilne uređaje. Pritom će glavna uloga web-servisa biti da objedinjavaju neku programsku logiku odnosno predstavljaju već spomenuti *potprogram* koji možete pozivati iz bilo kojeg dijela svoje aplikacije.

## Standardi

No dobro, u čemu je tajna web-servisa? Vrlo jednostavno – tajna je u njihovoj standardiziranosti. Naime, web-servisi nisu uopće ovisni o platformi. Recimo to najjednostavnijim rječnikom: web-servis možete izgraditi na IBM-ovoj platformi i zatim ga koristiti iz .NET aplikacija; web-servis možete izgraditi na .NET-u, a zatim ga koristiti iz Oracle aplikacija; web-servis možete izgraditi uz pomoć Oracleove tehnologije, a zatim ga koristiti iz aplikacija izrađenih u IBM-ovim alatima.

Recimo to ispravno – web-servisi nisu nečija tehnologija, već dogovoreni standard koji omogućava svima da ih koriste i time omogućavaju izradu pravih distribuiranih sustava. Pogledamo li malo šire, veoma rijetko ćemo pronaći tehnologije koje se koriste na svim platformama, što čini web-servise doista uspješnima.

## Nigdar ni bilo...

**N**ekoć davno, u doba zmajeva i prinčeva, nije bilo web-servisa, no oduvijek postoji ideja i želja za takozvanim distribuiranim aplikacijama. Kao što im i ime kaže, radi se o aplikacijama čija je funkcionalnost *podijeljena* na više lokacija i više programa. Jedan od mnogih koraka u tom smjeru bio je i Distributed COM (DCOM), čiji je zadatak bio da funkcionalnost programa bude distribuirana kroz mrežu, no opet dostupna svim zainteresiranim programima. DCOM je bio izgrađen na temeljima *remote procedure call* (RPC) arhitekture, što je za posljedicu imalo i niz nedostataka.

Kao prvo, DCOM i RPC su bili mnogo prikladniji za intranetska okruženja, a ne za Internet. *Portovi* kojima komuniciraju DCOM i RPC najčešće nisu otvoreni na korporacijskim mrežama, već su zaštićeni vatrozidovima. Iako to nije bio problem sve do popularizacije Interneta, danas je to ipak preveliko okruženje i svaka distribuirana tehnologija mora bez problema raditi preko Interneta (budimo potpuno precizni – za DCOM je postojao COM Internet Services

koji je za komunikaciju koristio HTTP *port* 80 i tako izbjegavao vatrozide, no to ipak nije bilo rješenje svih problema).

Drugi njihov nedostatak bio je utjecaj na same aplikacije. Njihovo korištenje bilo je jako teško i komplicirano te je najčešće zahtijevalo promjenu arhitekture aplikacija. Kako je DCOM nadogradnja COM (Component Object Model) tehnologije, koja je, kao što smo u prvim poglavljima objasnili, zastarjela pojavom .NET-a zbog svojih nedostataka, i DCOM je patio od tih istih nedostataka – ne baš jednostavnog korištenja, problemima s registriranjem ispravnih verzija komponenti i slično.

Na samom kraju, najveći nedostatak iz današnje perspektive bilo je to da DCOM nije bio neovisan o platformi, već je podrazumijevao Microsoftove Windowse. Web-servisi, naravno, adresiraju svaki od navedenih nedostataka i nameću se kao rješenje budućnosti (a i današnjice) za distribuirane sustave.

Kao što smo rekli – web-servise možete izrađivati na nizu različitih platformi, a cilj ovog poglavlja je pokazati vam koliko je to jednostavno na Microsoft .NET platformi i u razvojnom okruženju Visual Studija .NET.



Razlog tolike standardnosti web-servisa i njihove opće prihvaćenosti leži u dvjema tehnologijama na kojima se oni temelje. Jedna je već spomenuta – HTTP protokol koji je temelj Weba, a druga je obrađena u prethodnom poglavlju – XML.

### III. DIO: DIJELOVI .NET-A

XML i HTTP spojeni zajedno rezultirali su novim standardom, Simple Object Access Protocol (SOAP). Za njega je zadužen W3C (World Wide Web Consortium), neovisno tijelo zaduženo za razvoj Interneta i donošenje novih standarda. SOAP koristi XML za slanje poruka preko HTTP-a, a zapravo se radi o posebnoj XML gramatici, kao što ćete vidjeti kasnije.

Uz SOAP ruku pod ruku ide i još nekoliko standarda – najvažniji je svakako WSDL ili Web Services Description Language. Njega su razvili Microsoft, IBM i drugi proizvođači ujedineni oko inicijative web-servisa, a radi se prvenstveno o XML-shemi koja opisuje neki web-servis, sve njegove metode i njihove parametre. Zahvaljujući toj shemi moguće je komunicirati s web-servisom, jer je poznato koju funkcionalnost sadržava te kako se ta funkcionalnost može iskoristiti i pozvati iz druge aplikacije.

Tu je i UDDI – Universal Description, Discovery and Integration, otvoreni *framework* za opisivanje web-servisa i njihovu međusobnu integraciju i povezivanje. UDDI je također rezultat IBM-ova i Microsoftova prijedloga, a više informacija o njemu možete naći na <http://www.uddi.org/>.

**Slika 12-1:**  
**Web-site**  
**[www.uddi.org](http://www.uddi.org)**  
**sadržava detaljne**  
**informacije o UDDI-u.**



Uz sve opisane standarde, pri izradi web-servisa u .NET-u susrest ćete se i *.disco* datotekama odnosno *Discoveryem*, Microsoftovim protokolom za XML dokumente koji sadrže linkove na druge resurse koji opisuju web-servis.

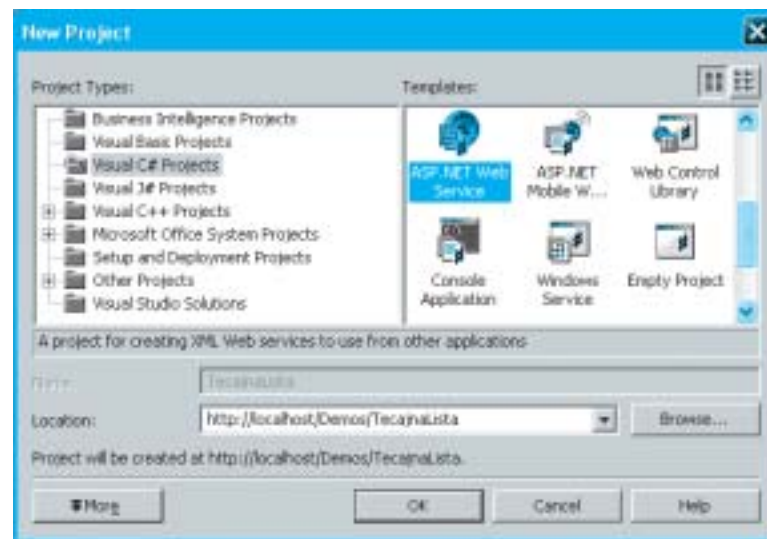
Iako SOAP postoji još od 1999. godine, dosad nije bilo puno programera koji su ga znali iskoristiti za stvaranje SOAP aplikacija. Dolaskom .NET-a i integrirane podrške za izradu web-servisa u Visual Studiju, za razvoj web-servisa koji, kao što je rečeno, koriste SOAP protokol, gotovo da i nećete morati poznavati tajne SOAP-a. To je, naravno, djelomično istina, jer ćete vi, kao savjestan programer željan novih znanja i iskustava, rado proučavati SOAP poruke kako se budete bavili web-servisima i znanje o SOAP-u ćete nesvjesno usvojiti u svakodnevnom radu.

Eto, sad kad ste uvjereni da su web-servisi *way to go*, ne samo zbog svoje opće prihvaćenosti i nevezanosti isključivo uz Microsoft, vrijeme je da se bacimo na pravi posao – izradu web-servisa. Bez straha, izrada web-servisa u Visual Studiju .NET je doista lagana, a ako ste svladali prethodna poglavlja i udomačili se u programiranju .NET aplikacija, za vas će ostatak ovog poglavlja biti mačji kašalj.

## Izrada web-servisa

Nakon svega što ste vidjeli u ovoj knjizi, vjerojatno vas neće posebno iznenaditi spomenuta lakoća njihove izrade u Visual Studiju .NET, jer ćete takvo što ipak očekivati. Dakle, njihova izrada se ni po čemu ne razlikuje od već opisane izrade prozorskih ili web-aplikacija.

Stoga se odmah upustimo u izradu web-servisa: u Visual Studiju odaberite *File – New – Project* te zatim pod *Visual C# Projects* pronađite *ASP.NET Web Services*. To je pravi trenutak da objasnimo zašto se tako zovu odnosno zašto riječ ASP.NET u njihovu nazivu. Naime, web-servis će biti smješten u nekom virtualnom direktoriju na Internet Information Services (IIS) poslužitelju, što je identično standardnim ASP.NET web-aplikacijama jer će se njima pristupati preko Web-a.



**Slika 12-2:**  
**Stvaranje novog**  
**web-servisa**

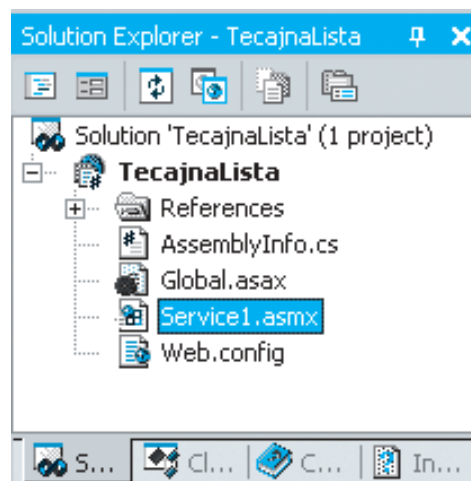
### III. DIO: DIJELOVI .NET-A

Stoga u polje *Location* upisujete lokaciju na poslužitelju gdje će web-servis biti smješten, a to ujedno i predstavlja njegovo ime. U nastavku ćemo se baviti izradom jednostavnog servisa, pojednostavljene tečajne liste, stoga smo ga nazvali “TecajnaLista”.

Nakon što kliknete OK, Visual Studio .NET će za vas automatski stvoriti virtualni direktorij i namjestiti sve potrebne datoteke.



**Slika 12-3:**  
**Početne datoteke**  
**novog web-servisa**



Primijetite da među datotekama u direktoriju web-servisa postoji i *web.config* datoteka. Nju ste već prije upoznali u poglavlju o ASP.NET-u, a samo ukratko podsjetimo – služi za osnovnu konfiguraciju postavki web-aplikacije, a njeno postojanje uz web-servis dodatno objašnjava zašto ih u Visual Studiju .NET zovu “ASP.NET Web Services”. Naravno, kao i kod ASP.NET aplikacija, *web.config* datoteka ima istu namjenu, te je stoga nećemo dodatno objašnjavati.

Web-servisi su u .NET-u predstavljeni kao datoteke s ekstenzijom *.asmx*. Pogledajte među datotekama na slici 12-3 i vidjet ćete jednu imena *Service1.asmx*. Svaka datoteka s *.asmx* ekstenzijom je jedan web-servis, a on može sadržavati više funkcionalnosti. ASMX datoteke, baš kao i ASPX za web-stranice ili ASCX za web-kontrole, kontrolira IIS i izvršava njihov kôd.

Za početak, preimenujmo u *Solution Exploreru* datoteku *Service1.asmx* – označite je, pritisnite F2 i upišite novo ime, primjerice “*Valute.asmx*” pa je, ako nije već otvorena, dvaput kliknite i otvorite.

## 12. POGLAVLJE: WEB-SERVISI

Za web-servise je ključno razumjeti da oni nemaju sučelje i da s njima ne radite kao s običnim ASP.NET stranicama. Tako će i datoteka *Valute.aspx* sadržavati sljedeći kôd:

```
<%@ WebService Language="c#" Codebehind="Valute.aspx.cs"
Class="TecajnaLista.Service1" %>
```

Nju ćete standardno otvoriti u grafičkom načinu rada i tamo će vas dočekati poruka koja kaže da, ako želite dodati komponente web-servisu, trebat ćete ih povući iz *Server Explorera* ili *Toolboxa* i iskoristiti *Properties* prozor za njihovo namještanje, no isto tako, ako želite stvoriti nove metode, trebat ćete se prebaciti u prikaz kôda.

Prevedeno, kako web-servisi nemaju svoje sučelje kojem biste dodali kontrole poput tekstualnih polja za unos ili padajućih izbornika, jedino što možete raditi jest pisati kôd. No isto tako, možete iz *Toolboxa* dodavati komponente za spajanje na baze podataka na isti način, što je i objašnjeno u 9. poglavlju.

Što se tiče sadržaja ASMX datoteke, kao što je i vidljivo u prethodnom ispisu, tu se nalazi samo uputa o web-servisu, a kompletan kôd će biti sadržan u datoteci s pozadinskim kôdom s ekstenzijom *.asmx.cs*. Atribut *Class* određuje ime klase web-servisa. Pri prvom pokretanju i pristupanju web-servisu, mehanizam ASP.NET-a će pretražiti direktorij *bin* u potrazi za *assemblyjem* koji sadrži navedenu klasu.

## Pisanje kôda

Kao što smo rekli, odlučili smo se za izradu jednostavne tečajne liste. U pravom svijetu vi ćete te podatke dohvaćati iz baze ili neke XML datoteke, no zbog jednostavnosti i da bi vam fokus ostao na najzanimljivijoj stvari – samoj izradi web-servisa, odlučili smo se za lakši pristup.

**Ako baš želite izrađivati web-servis tečajne liste, Hrvatska narodna banka svaki dan objavljuje tečajnu listu u posebnom formatiranom zapisu prilagođenom za automatsku obradu. Za više informacija posjetite <http://www.hnb.hr/tecajn/htecajn.htm>. Te datoteke možete automatski preuzimati i iz njih programski izvlačiti odgovarajuće podatke.**



Prebacite se u prikaz kôda web-servisa – kliknite na link “click here to switch to code view” u prikazu ASMX datoteke ili desnim gumbom na *Valute.aspx* i odaberite *View Code*. U kodu ćete tako pronaći komentiranu sljedeću metodu:

```
[WebMethod]
public string HelloWorld()
```

### III. DIO: DIJELOVI .NET-A

```
{
    return "Hello World";
}
```

Njena struktura sasvim dobro objašnjava dijelove neke metode web-servisa. Da biste nekoj funkcionalnosti, tj. metodi web-servisa mogli pristupati putem Weba, ona mora biti označena s [WebMethod], a prava pristupa joj moraju biti postavljena na *public*. Takva se metoda može pozivati kao metoda web-servisa, a u gornjem primjeru ona samo vraća poruku "Hello World".



**Naravno, kao i kod programiranja drugih tipova aplikacija, unutar web-servisa možete imati velik broj metoda koje se interno koriste, no nisu označene s [WebMethod] pa im nije moguće pristupiti izvana preko Weba. U njih možete staviti dijelove kôda koji se često koriste iz različitih metoda web-servisa.**

No mi ćemo ipak napisati svoju verziju jednostavne web-metode. Napisat ćemo metodu koja će iznos u nekoj drugoj valuti pretvarati u kune. Primat će dva parametra – jedan će označavati iznos, a drugi valutu. Primjerice, pozovemo li metodu s parametrima (100, "EUR"), rezultat će biti 750.11 (naravno, kuna).

Evo kako bi i izgledao kôd takve metode (još jednom napomena – zbog jednostavnosti smo *hardkodirali* odnose tečajeva, no u stvarnoj situaciji vi biste te podatke vukli iz baze podataka ili nekog drugog izvora, poput prije spomenute datoteke koju objavljuje HNB).

```
[WebMethod]
public double PretvoriValutu(double Kolicina, string Valuta)
{
    double faktor, Kuna;
    switch (Valuta)
    {
        case "EUR": faktor = 7.501118; break;
        case "USD": faktor = 6.115374; break;
        case "CHF": faktor = 4.787540; break;
        case "GBP": faktor = 11.094687; break;
        default: faktor = 0; break;
    }
    Kuna = Math.Round(faktor * Kolicina, 2);
    return Kuna;
}
```



Kôd je uistinu jednostavan – metoda prima dva parametra te u ovisnosti o drugom parametru izračunava faktor. Potom množi proslijeđeni iznos (varijabla *Kolicina*) s faktorom i zaokružuje rezultat na dvije decimale. Taj se rezultat (varijabla *Kuna*) vraća kao rezultat metode.

## Imenski prostor web-servisa

**P**ri izradi web-servisa preporučuje se dodijeliti mu odgovarajući *namespace*. To može biti vaša internetska adresa na kojoj će biti smješten web-servis, ime tvrtke ili bilo što drugo što bi ga moglo identificirati. *Namespace*ovi web-servisa tako često podsjećaju na URL-ove, no oni ne moraju ukazivati na stvarne resurse na Internetu.

Ukoliko svom web-servisu ne dodijelite *namespace*, dobit će *defaultni*, "http://tempuri.org/". On je prikladan za web-servise koji se još razvijaju i nisu objavljene njihove finalne verzije, no čim završite s izradom web-servisa, preporučuje se namještanje *namespacea*. Taj *namespace* će biti korišten u svim SOAP porukama i služi za lakše identificiranje vašeg servisa.

*Namespace* se postavlja klasi web-servisa. Prebacite li se u kôd, na početku ćete vidjeti naziv klase (imat će ime "Service1", ako ste sve radili kao što je opisano u knjizi), a iznad te linije trebate dodati *namespace*, primjerice:

```
[WebService(Namespace="http://NET/Poglavlje12/TecajnaLista")]
public class Service1 :
    System.Web.Services.WebService
{
    // itd.
```

*Namespace* se dodaje unutar *WebService* direktive. U našem smo primjeru dodali *namespace* koji nam pomaže u snalaženju – podsjeća na URL, no on to nije. Naravno, vi *namespaceu* možete dati bilo koji oblik i bilo koju vrijednost, to je potpuno na vama.

Klasa *Service1* koja u sebi sadržava svu funkcionalnost web-servisa, nasljeđuje *System.Web.Services.WebService*. U klasi *WebService* se tako nalaze članovi potrebni za ispravan rad web-servisa. U njoj je i podrška za rad sa *sessionima* te objekti *Server* i *User*, baš kao i u web-aplikacijama.

## Opcije web-metoda

Kao što ste vidjeli, da bismo neku metodu učinili dijelom javne i svima dostupne funkcionalnosti web-servisa, označili smo je s [WebMethod]. Sama direktiva ima šest opcija pomoću kojih se dodatno može upravljati načinom rada servisa.

### III. DIO: DIJELOVI .NET-A



Pri čitanju dokumentacije vezane uz web-servise, susrest ćete se s nekoliko sličnih akronima: URI, URL i URN. URI ili Uniform Resource Identifier je niz koji jedinstveno identificira neki resurs na mreži. Postoje dva tipa URI-ja: Uniform Resource Locator (URL) i Uniform Resource Name (URN). URL je zadan prefiksom koji određuje protokol, adresom servera ili IP adresom, portom i putem do resursa (radi se o standardnoj adresi, primjerice <http://www.server.com/aplikacija/>). URN je pak običan opisni niz koji podsjeća na adresu, primjerice NET12:/tečajna\_lista. Pri obradi URN-a, aplikacija može znati da NET12 odgovara 12. poglavlju knjige o .NET-u te da se u njemu nalazi opis tečajne liste. Dakle, radi se o opisu – URN ne ukazuje na stvarni resurs na Internetu.

**BufferResponse.** Njena *defaultna* postavka je *true* jer je omogućeno *bufferiranje* odgovora web-servisa odnosno njegovo slanje *u komadu*. Ukoliko ga pak postavite na *false*, odgovor web-servisa će se slati u dijelovima velikim 16 KB. Naravno, preporučljivo je ostaviti *defaultnu* opciju i ne mijenjati ništa, jer se slanjem u jednom dijelu smanjuje opterećenost servera i ubrzava prijenos podataka.

```
[WebMethod(BufferResponse=false)]
```

**CacheDuration.** Odgovori odnosno rezultati poziva web-servisa mogu se *cacheirati*, baš kao i rezultati ASP.NET skripti. Vrijednost ovog atributa određuje vrijeme *cacheiranja* odgovora u sekundama. *Defaultno* vrijeme je 0, odnosno ništa se ne *cacheira*. No ukoliko imate određen set parametara koji mogu biti prosljeđeni web-servisu i koji uvijek rezultiraju istim rezultatima, isplati se uključiti *cacheiranje*.

Evo i primjera kako namjestiti da se rezultati metode *cacheiraju* i čuvaju 5 minuta. Ukoliko se za to vrijeme dogodi poziv s istim parametrima, on neće izvršiti metodu web-servisa, već će odmah dobiti spremljeni odgovor.

```
[WebMethod(CacheDuration=300)>
```

**Description.** Unutar ovog parametra navodite opis web-servisa koji se pojavljuje na stranici za pomoć. Kako se radi o tekstualnom parametru, njegovu vrijednost navodite unutar navodnika.

```
[WebMethod(Description="Tečajna lista vraća podatke...")]
```

**EnableSession.** Omogućava korištenje *sessiona* te može imati vrijednost *true* ili *false*. Ukoliko omogućite *sessione*, možete im pristupiti preko *HttpContext.Current.Session* ili preko *WebService.Session*. Način korištenja *sessiona* isti je kao i u web-aplikacijama.

## 12. POGLAVLJE: WEB-SERVISI

```
[WebMethod(EnableSession=true)]
```

**MessageName.** Ukoliko koristite preopterećene metode (prisjetimo se 6. poglavlja – preopterećene metode su one metode koje imaju isto ime, no drugačiji potpis, tj. različite parametre i tip), korištenjem *MessageName* možete namjestiti drugačije ime za svaku metodu. Po *defaultu* je vrijednost parametra *MessageName* jednaka imenu metode, no ukoliko imate više istoimenih metoda, možete im definirati *alias*e.

```
[WebMethod(MessageName="TecajKune")]
```

**TransactionOption.** Ovaj parametar omogućava sudjelovanje metode web-servisa kao korijenski objekt u transakciji. Za postavljanje njegove vrijednosti koristi se pobrojan niz *TransacionOption*, a možete iskoristiti dvije mogućnosti – ili web-servis ne može sudjelovati u transakciji (*Disabled*, *NotSupported* ili *Supported*) ili se stvara nova transakcija (*Required*, *RequiresNew*). *Defaultna* vrijednost je *TransactionOption.Disabled*. Da biste mogli koristiti transakcije, u web-servisu trebate uključiti referencu na *System.EnterpriseServices.dll* odnosno uključiti *System.Enterprise namespace..*

```
[WebMethod(TransactionOption=TransactionOption.RequiresNew)]
```

Kako je HTTP protokol *stateless* odnosno prethodni koraci i operacije se ne pamte, web-servisi ne mogu punopravno sudjelovati u postojećim transakcijama. Primjerice, u slučaju *rollbacka* odnosno otkazivanja transakcije provjeravanje što je sve transakcija promijenila i otkazivanje svih operacija bi bilo problematično. No ipak, web-metode mogu pokrenuti nove transakcije – tako možete pratiti rad i izvršene naredbe u web-metodi i na njenom kraju (ili ranije), ako želite, otkazati sve promjene.

Kad se unutar web-metode pojavi iznimka koja se ne obradi ispravno, transakcija se odmah prekida. Ukoliko pak nema nikakve iznimke, transakcija se na kraju metode automatski izvršava. Izvršavanje transakcije znači da se sve akcije obavljene unutar metode apliciraju i primjenjuju odnosno izvršavaju. Ukoliko se pak pozove *SetAbort* metoda, transakcija se prekida i odustaje od izvršavanja metode – niti jedna linija kôda metode se neće doista izvršiti (ako se već neka naredba izvršila, ona će se otkazati i sve će se vratiti na prethodno stanje). Naravno, da biste mogli koristiti *SetAbort* metodu (i *SetComplete*), možete im pristupiti preko *ContextUtil* klase koja je definirana u *System.Enterprise namespaceu*.



Iako većinu ovih parametara nećete mijenjati, neke od njih ipak možete iskoristiti za metode svojih web-servisa. Tako ćete često iskoristiti *cacheiranje* podataka te davanje opisa samim meto-

### III. DIO: DIJELOVI .NET-A

dama. Evo i kako biste spojili dva gornja parametra i definirali da se odgovor metode web-servisa cacheira narednih 5 minuta te namjestili njen opis.

```
[WebMethod(CacheDuration=300, Description="Metoda pretvara iznos u stranoj valuti u kunsku protuvrijednost")]
```

Atributi *WebMethod* direktive odvajaju se zarezima pa ih tako, ako želite, možete sve navesti.

## Isprobavanje web-servisa

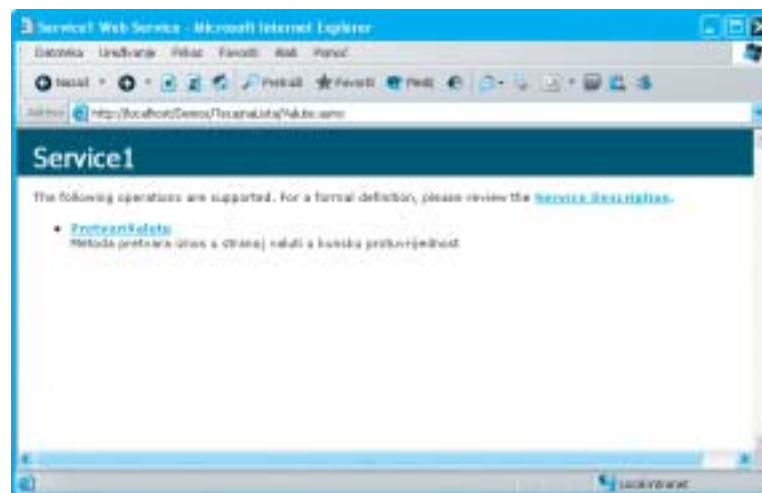
Nakon mukotrpne izrade web-servisa i dugog vremena prilagođavanja na njihov poseban način pisanja kôda (naravno, uočavate ironiju), vrijeme je za njihovo isprobavanje. Baš kao i pri izradi drugih tipova aplikacija, iskoristite opciju *Build* (izbornik *Build* – *Build Solution* ili pritisnite CTRL+SHIFT+B).

Kako ste na početku stvaranja web-servisa odabrali lokaciju na kojoj će se on nalaziti, na lokalnom serveru *localhost* u nekom virtualnom direktoriju, možete otvoriti preglednik i ručno upisati adresu. Ukoliko ste radili sve kao i u primjeru, web-servis će se nalaziti na adresi <http://localhost/Demos/TecajnaLista/Valute.aspx>. Primijetite da za adresu web-servisa upisujete točnu adresu ASMX skripte jer se u njoj nalazi web-servis.



Naravno, ne trebate sami upisivati adresu. U Visual Studiju u izborniku *Debug* odaberite opciju *Start Without Debugging* (CTRL+F5) ili *Start* (F5) i otvorit će se Internet Explorer te automatski učitati web-servis.

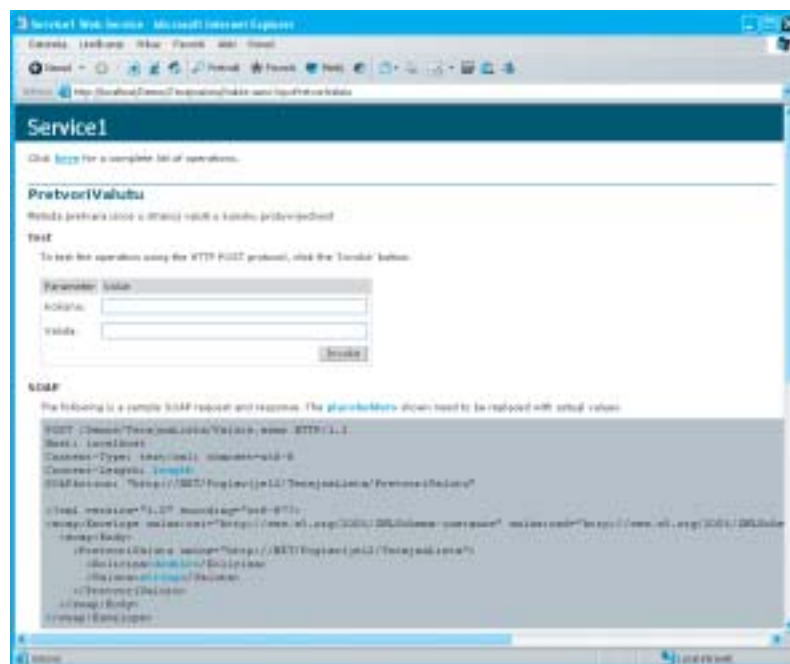
**Slika 12-4:**  
**Prikaz web-servisa u pregledniku**



## 12. POGLAVLJE: WEB-SERVISI

Nakon pokretanja i učitavanja web-servisa, dočekat će vas sučelje poput onog na slici 12-4. Radi se o automatski generiranom sučelju koje prikazuje popis svih metoda web-servisa. Kako je u našem primjeru postojala samo jedna metoda označena s *WebMethod*, tako je samo ona ponuđena u sučelju (primijetite ispod nje opis koji smo naveli kao *Description* parametar u *WebMethod* direkтиви), no da smo ih imali više, sve bi bile prikazane.

Klikom na tu metodu otvorit će vam se sučelje za komunikaciju s njom, kao na slici 12-5. Tu će se pojaviti tekstualna polja za unos svih parametara. Imena parametara su automatski izvučena iz kôda metode i naziva parametara koji joj se prosljeđuju.



**Slika 12-5:**  
**Prikaz sučelja za**  
**isprobavanje metode**  
**web-servisa**

Uočite i adresu te metode koja neodoljivo podsjeća na ASP.NET skripte – kao parametar *op* proslijeđeno je ime metode koju želimo pozivati.

`http://localhost/Demos/TecajnaLista/Valute.aspx?op=PretvoriValutu`

Upišite u tekstualna polja parametre za isprobavanje metode. Primjerice, možete za parametar *Kolicina* upisati “500”, a za parametar *Valuta* “USD”. Pritisnite gumb *Invoke* i otvorit će se novi prozor s rezultatima poziva metode prikazan na slici 12-6.

### III. DIO: DIJELOVI .NET-A

Na stranici neke metode web-servisa prikazane su i SOAP-poruke koje služe za komunikaciju. Iako nije obavezno, bacite pogled na njih i proučite njihov sadržaj. SOAP je ipak preveliko područje da bismo ga obradili u ovoj knjizi, no o njemu postoji čitav niz zasebnih knjiga, koje vam preporučujemo. Zasad samo informativno pogledajte kako su te poruke građene i kako se obavlja komunikacija s web-servisom.



## Složenac

I ako smo u našem primjeru pokazivali vraćanje jednostavnih tipova podataka zahvaljujući XML serijalizaciji objekata i SOAP-u koji može prezentirati kompleksne tipove podataka zbog internog korištenja XML shema, web-servis može vraćati objekte poput *DataSet*ova, klasa i struktura. Primjerice, mogli smo napraviti web-servis koji bi vraćao *DataSet* – evo njegove skraćene verzije:

```
[WebMethod]
public DataSet
PretvoriValutu(double Kolicina,
string Valuta)
{
    DataSet ds = new DataSet();
    // punjenje DataSeta iz baze
    // podataka ili u samom kodu
    return ds;
}
```

Dakle, vi biste iz aplikacije koja koristi web-servis mogli pozvati gornju metodu i automatski biste u svojoj aplikaciji dobili rezultat u obliku objekta *DataSet*. Interno bi se konverzija odvijala dvaput – prvi put bi se *DataSet* iz web-metode pretvorio u XML u SOAP-poruci koja bi se poslala aplikaciji, zatim bi se u aplikaciji dobivena SOAP-poruka natrag pretvorila u objekt *DataSet* koji biste mogli normalno koristiti u svo-

joj aplikaciji. Vi pritom ne trebate ništa znati o tim konverzijama i na sve možete gledati s više razine – web-metoda vraća *DataSet* koji vi primete i normalno koristite u svojoj aplikaciji (u nastavku poglavlja bit će prikazano kako iz aplikacije pozivati web-servise i raditi s njihovim rezultatima).

Evo i kako izgleda objekt *DataSet* pretvoren u XML unutar SOAP-poruke – radi se o *DataSetu* s jednim objektom *DataTable* koji ima dva stupca (Valuta i Faktor) te dva zapisa s odgovarajućim vrijednostima. Uočite da je na njegovom početku prvo definirana XML shema podataka, a tek poslije stvarni podaci.

```
<?xml version="1.0"
encoding="utf-8" ?>
<DataSet
xmlns="http://NET/Poglavlje12/Te
cajnaLista">
  <xs:schema id="NewDataSet"
xmlns=""
xmlns:xs="http://www.w3.org/2001
/XMLSchema"
xmlns:msdata="urn:schemas-
microsoft-com:xml-msdata">
    <xs:element
name="NewDataSet"
msdata:IsDataSet="true"
msdata:Locale="hr-HR">
```

## 12. POGLAVLJE: WEB-SERVISI

```

<xs:complexType>
  <xs:choice maxOccurs="unbounded">
    <xs:element name="Tečajna">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Valuta" type="xs:string"
            minOccurs="0" />
          <xs:element name="Faktor" type="xs:double"
            minOccurs="0" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>
<diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
  xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
  <NewDataSet xmlns="">
    <Tečajna diffgr:id="Tečajna1" msdata:rowOrder="0"
      diffgr:hasChanges="inserted">
      <Valuta>EUR</Valuta>
      <Faktor>7.501118</Faktor>
    </Tečajna>
    <Tečajna diffgr:id="Tečajna2" msdata:rowOrder="1"
      diffgr:hasChanges="inserted">
      <Valuta>USD</Valuta>
      <Faktor>6.115374</Faktor>
    </Tečajna>
  </NewDataSet>
</diffgr:diffgram>
</DataSet>

```



**Slika 12-6:**  
**Rezultat poziva**  
**metode web-servisa i**  
**pretvaranja 500 USD**  
**u kune**

### III. DIO: DIJELOVI .NET-A



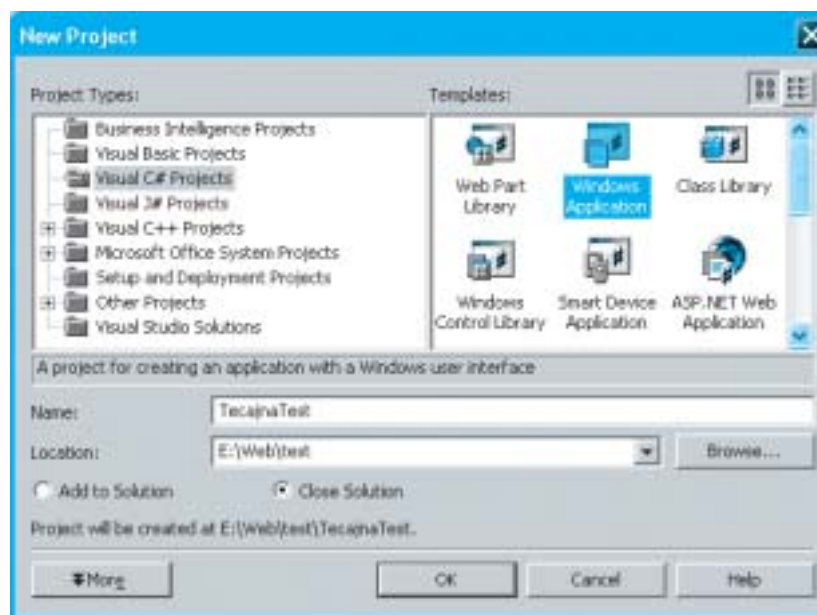
Web-sučelje koje je maloprije prikazano služi isključivo isprobavanju rada web-servisa – testiranju i proučavanju SOAP poruka koje se šalju te samog rada i funkcionalnosti dostupnih metoda. Prvenstvena namjena web-servisa je da budu korišteni iz drugih programa, dakle ne direktno od stvarnih korisnika – stoga će naš sljedeći korak u radu s web-servisima biti njihovo korištenje iz drugih programa pisanih u .NET-u.

## Korištenje web-servisa

Iako je za korištenje web-servisa iz aplikacija nužna povećana količina kôda, vi je, kao što ste već vjerojatno i navikli, nećete morati pisati. To će za vas obaviti Visual Studio .NET. U narednom primjeru prikazat ćemo korištenje web-servisa iz prozorske aplikacije za Windowse, no rad s web-servisima se nimalo ne razlikuje u slučaju da ih odlučite upotrebljavati iz web-aplikacija (ili drugih tipova aplikacija, pa čak i iz drugih web-servisa).

Stoga započnimo izradu male aplikacije koja će služiti kao pretvarač valuta. Kako smo se u početku pri izradi web-servisa ograničili samo na jednu metodu, tako će i naša aplikacija imati malu funkcionalnost pretvaranja iz stranih valuta u kune. Stvorite stoga novi projekt kao na slici 12-7.

**Slika 12-7:**  
**Stvaranje nove**  
**prozorske**  
**aplikacije u kojoj**  
**ćemo koristiti**  
**web-servis**





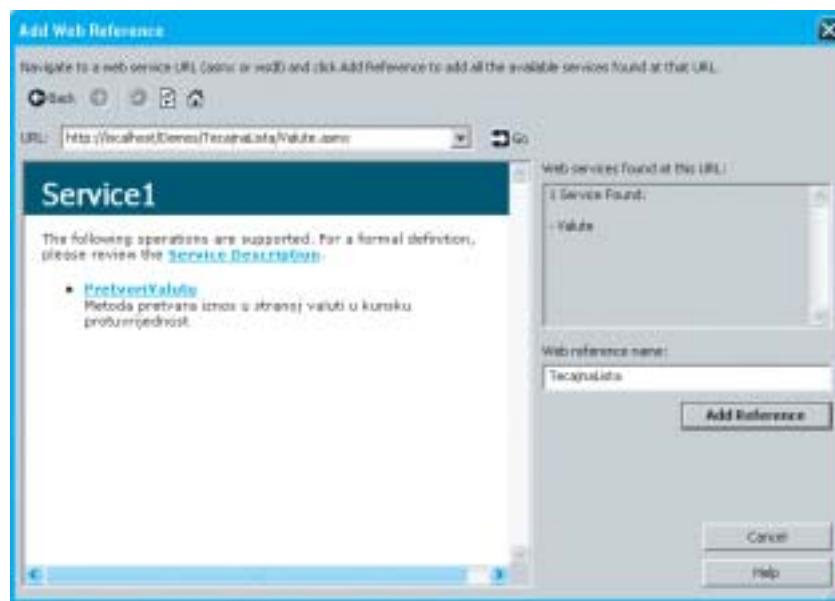
Važno je primijetiti da će sva funkcionalnost aplikacije biti sadržana u već napraavljenom web-servisu, a sama prozorska aplikacija služit će isključivo kao sučelje. To je i cilj korištenja web-servisa – da sadržavaju neku programsku funkcionalnost koju je moguće pozivati iz različitih tipova programa putem weba. Ako biste željeli sad napraviti web-aplikaciju za pretvaranje valuta, napravili biste samo sučelje i pozivali web-servis za dohvat rezultata.

## Dodavanje web-reference

Da bismo iz aplikacije mogli koristiti web-servise, trebamo napraviti tzv. *proxy*-klasu koja će komunicirati sa samim servisom i njegovim metodama. Kasnije u poglavlju ćemo pokazati kako se to radi ručno pomoću alata *wSDL.exe*, no za početak ćemo iskoristiti pomoć koju nam nudi Visual Studio .NET.

Stoga se prebacite u prozor *Solution Explorer* netom stvorene prozorske aplikacije i kliknite desnom tipkom miša na stavku *References* te odaberite opciju *Add Web Reference* i pojavit će vam se prozor kao na slici 12-8.

U prozoru *Add Web Reference* ne trebate ručno upisivati adresu web-servisa ukoliko je on smješten na istom računalu – jednostavno možete kliknuti na link *Browse to – Web services on the local machine* i dobit ćete popis svih web-servisa na računalu.



**Slika 12-8:**  
**Dodavanje**  
**postojećeg**  
**web-servisa kao**  
**reference u**  
**aplikaciju**

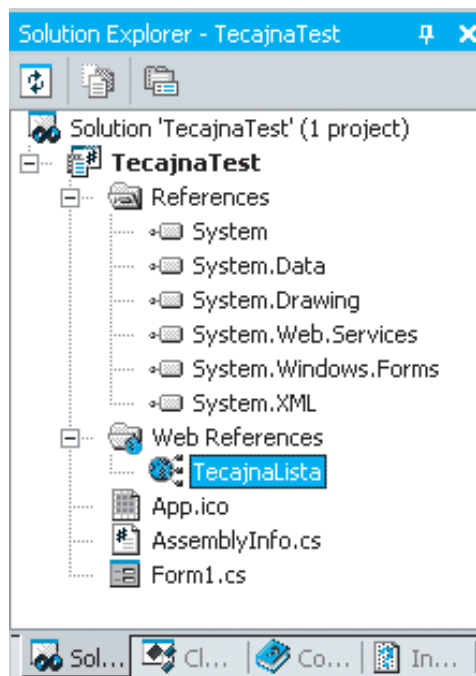
### III. DIO: DIJELOVI .NET-A

U polje za URL upišite adresu web-servisa – ukoliko se on nalazi na nekom drugom računalu na Internetu, upišite njegovu punu adresu, ime servera i put do servisa, a ako se nalazi na lokalnom računalu (kao u našem primjeru), upišite adresu koja počinje s *http://localhost/*.

Obavezno promijenite ime reference u tekstualnom polju označenom s “Web reference name”. Naime, tamo će po *defaultu* stajati ime “localhost”, no to nije neko odgovarajuće ime za web-servis koje želite koristiti iz svoje aplikacije. Primjerice, možete mu dati ime “TecajnaLista” jer ćete se tako lakše snalaziti.

Nakon što ste dodali web-referencu, u *Solution Exploreru* možete pronaći web-servis po stavkom *Web References*. Klikom na njega možete mu u prozoru *Properties* mijenjati osnovne postavke.

**Slika 12-9:**  
Referenca na web-servis u  
*Solution Exploreru*



U stvarnim ćete situacijama web-servise koristiti za pružanje neke funkcionalnosti preko Interneta. Ukoliko pak želite tu funkcionalnost omogućiti samo aplikacijama na istom računalu, korištenje web-servisa nije baš efikasno jer je ipak pozivanje web-servisa sporije nego instanciranje klase s kôdom na istom računalu.

## Proxy-klasa

Nakon dodavanja reference na web-servis, s njim Za to je najzaslužnija *proxy*-klasa koja je automatski stvorena stvaranjem reference na web-servis. Pogledate li u direktorij u kojem je smještena vaša aplikacija koja koristi web-servis, pronaći ćete direktorij imena "Web References". U njemu će se pak nalaziti još jedan direktorij imena dodane reference (u našem slučaju "TecajnaLista").

U njemu se, među ostalima, nalazi WSDL datoteka koja opisuje web-servis te za ovaj slučaj mnogo važnija C# datoteka (s ekstenzijom .cs) koja sadrži kôd *proxy*-klase. Ta *proxy*-klasa omogućava korištenje web-servisa iz vaše aplikacije. Slijedi ogledni sadržaj datoteke s *proxy*-klasom imena "Reference.cs".

```
//-----// <autogenerated>
// This code was generated by a tool.
// Runtime Version: 1.1.4322.573
//
// Changes to this file may cause incorrect behavior and will be lost if
// the code is regenerated.
// </autogenerated>
//-----

//
// This source code was auto-generated by Microsoft.VSDesigner
// Version 1.1.4322.573.

namespace TecajnaTest.TecajnaLista {
    using System.Diagnostics;
    using System.Xml.Serialization;
    using System;
    using System.Web.Services.Protocols;
    using System.ComponentModel;
    using System.Web.Services;

    /// <remarks/>
    [System.Diagnostics.DebuggerStepThroughAttribute()]
    [System.ComponentModel.DesignerCategoryAttribute("code")]

    [System.Web.Services.WebServiceBindingAttribute(Name="Service1Soap",
        Namespace="http://NET/Poglavlje12/TecajnaLista")]
    public class Service1 : System.Web.Services.Protocols.SoapHttpClientProtocol {
```

### III. DIO: DIJELOVI .NET-A

```

    /// <remarks/>
    public Service1() {
        this.Url = "http://localhost/Demos/TecajnaLista/Valute.asmx";
    }

    /// <remarks/>

[System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://NET/Poglavlje12/TecajnaLista/PretvoriValutu", RequestNamespace="http://NET/Poglavlje12/TecajnaLista", ResponseNamespace="http://NET/Poglavlje12/TecajnaLista", Use=System.Web.Services.Description.SoapBindingUse.Literal, ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
    public System.Double PretvoriValutu(System.Double Kolicina, string Valuta) {
        object[] results = this.Invoke("PretvoriValutu", new object[] {
            Kolicina,
            Valuta});
        return ((System.Double)(results[0]));
    }

    /// <remarks/>
    public System.IAsyncResult BeginPretvoriValutu(System.Double Kolicina, string Valuta, System.AsyncCallback callback, object asyncState) {
        return this.BeginInvoke("PretvoriValutu", new object[] {
            Kolicina,
            Valuta}, callback, asyncState);
    }

    /// <remarks/>
    public System.Double EndPretvoriValutu(System.IAsyncResult asyncResult) {
        object[] results = this.EndInvoke(asyncResult);
        return ((System.Double)(results[0]));
    }
}

```

Dakle, radi se o klasi pisanoj u C# koja sadržava kôd za komunikaciju s web-servisom. Tu se nalaze sve metode za pozivanje pravih metoda web-servisa. Bacite li pogled na metodu *PretvoriValutu*, primijetiti ćete da se pri njenom korištenju poziva metoda *Invoke* koja vraća polje objekata, a prvi element tog polja se pretvara u *Double* tip (koji i vraća sama metoda) te vraća kao rezultat.

Prethodni kôd se automatski generira te ga nije preporučljivo mijenjati jer će se pri sljedećem generiranju *proxy*-klase izgubiti sve promjene.

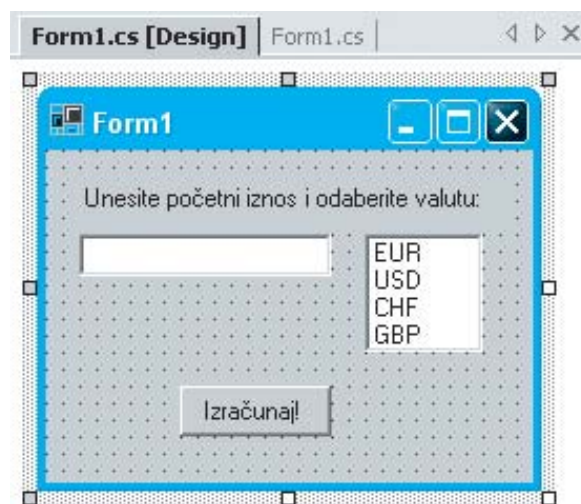


## Komunikacija s web-servisom

Da biste mogli komunicirati s web-servisom, koristit ćete *proxy*-klasu preko imena web-reference, što ćete najbolje vidjeti na primjeru. Kako smo na početku stvorili prozorsku aplikaciju, dodajmo joj par kontrola koje će nam poslužiti za komunikaciju s web-servisom.

Dodajte jedno tekstualno polje koje će služiti za unos količine valute (imena *textBox1*) te jednu kontrolu *ListBox* koja će ponuditi na izbor sve dostupne valute (imena *listBox1*). Označite potom *ListBox* i u prozoru *Properties* pod *Data – Items* unesite kolekciju elemenata (preporučljivo je da upišete samo one s kojima web-servis zna raditi – “EUR”, “USD”, “CHF”, “GBP”).

Trebat će vam i jedan gumb koji će pokretati akciju i vraćati rezultat. Ostavite mu ponuđeno ime *button1* i smjestite ga na radnu površinu. Konačni rezultat prozora može izgledati poput prikazanog na slici 12-10.



**Slika 12-10:**  
**Razmještaj kontrola u jednostavnoj aplikaciji za komunikaciju s web-servisom**

Još samo trebate dodati kôd koji će komunicirati s web-servisom. Da biste započeli komunikaciju s web-servisom trebat ćete samo stvoriti novi objekt maloprije objašnjene *proxy*-klase, primjerice:

### III. DIO: DIJELOVI .NET-A

```
TecajnaLista.Service1 t1 = new TecajnaLista.Service1();
```

Dakle, instancirali smo objekt tipa *TecajnaLista.Service1*, što odgovara našem web-servisu – njemu smo dali ime reference *TecajnaLista*, a unutar samog web-servisa postoji klasa *Service1* u kojoj se nalaze metode za rad. Pogledajte i uvjerit ćete se da je *proxy*-klasa baš to – klasa imena *Service1* i *namespacea TecajnaLista*.

Želite li u svom kodu komunicirati s web-servisom, IntelliSense će vam ponuditi izbor svih metoda web-servisa i drugih članova za rad, kao na slici 12-11.

#### Slika 12-11:

**IntelliSense vam nudi izbor metoda za komunikaciju s web-servisom.**



Pouzdaajući se u IntelliSense doista je lagano napisati kôd koji će pozvati metodu web-servisa i vratiti rezultat. Kao što vidite, sve se svodi na pozivanje neke metode iz *proxy*-klase, prosljeđivanje odgovarajućih parametara i dobivanje rezultata kao odgovora. Potpuno isto kao da radite s bilo kojom drugom klasom!

Dvaput kliknite na gumb na formularu i možete započeti s upisivanjem kôda koji će se izvršiti pri kliku na gumb. U tom trenutku želimo web-servisu proslijediti korisnikov izbor iz aplikacije (upisanu količinu valute i odabranu valutu), te na kraju ispisati rezultat.

```
private void button1_Click(object sender, System.EventArgs e)
{
    string Valuta;
    double Kolicina;

    try
    {
```

## 12. POGLAVLJE: WEB-SERVISI

```

        Valuta = listBox1.SelectedItem.ToString();
        Kolicina = Convert.ToDouble(textBox1.Text);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Neispravni podaci");
        return;
    }

    TecajnaLista.Service1 tl = new TecajnaLista.Service1();
    double Rezultat = tl.PretvoriValutu(Kolicina, Valuta);

    MessageBox.Show(Rezultat.ToString() + " kn", "Rezultat");
}

```

U prethodnom kodu koristili smo vrlo slab i općenit mehanizam za upravljanje iznimkama – njime smo htjeli spriječiti situaciju u kojoj se upisana količina valute ne može pretvoriti u broj (primjerice, korisnik je upisao znakovni niz) te situaciju u kojoj korisnik nije iz ponuđene liste odabrao valutu. Ispravno bi bilo hvatati svaki tip iznimke i ispisati odgovarajuću poruku ili pak provjeravati ulazne podatke *if*-naredbama.

U svakom slučaju, osigurajte se da u klijentskoj aplikaciji provjerite sve ulazne podatke prije njihova slanja web-servisu. Ukoliko to ne napravite, uvijek možete prepustiti web-servisu provjeravanje ulaznih podataka i vraćanje informacije o grešci, no u tom slučaju nepotrebno trošite vrijeme na komunikaciju s web-servisom. Sve provjere nad podacima obavezno napravite unutar same aplikacije, a što manje posla ostavite web-servisu.



**Slika 12-1:**  
**Aplikacija za komunikaciju s**  
**web-servisom u akciji**

### III. DIO: DIJELOVI .NET-A

Metoda koja se poziva na klik gumba krajnje je jednostavna – veći njen dio zauzima kôd za obradu podataka iz formulara, a tek dvije naredbe služe za komunikaciju s web-servisom. Prvo se instancira novi objekt *tl* tipa *TecajnaLista.Service1*, a zatim se taj isti objekt koristi za pozivanje metode *PretvoriValutu* s dva parametra. Rezultat se vraća i sprema u varijablu *Rezultat* koja je tipa *double*.

Na samom kraju ispisujemo poruku s rezultatom u malom prozoru, kao na slici 12-13.



**Slika 12-13:**  
**Rezultat poziva web-servisa ispisan naredbom**  
**MessageBox.Show**



## Asinkrono pozivanje web-servisa

Vratite li se natrag u kôd *proxy*-klase, vidjet ćete da osim metode *PretvoriValutu* postoje još dvije metode sličnog naziva, *BeginPretvoriValutu* i *EndPretvoriValutu*. One služe za tzv. asinkrono pozivanje web-servisa, dok metoda *PretvoriValutu* služi za sinkrono pozivanje.



U prethodnom primjeru smo metodu web-servisa pozivali sinkrono. Da pojasnimo, *sinkrono* znači da smo pozvali metodu, a aplikacija se zatim pauzirala i čekala na odgovor web-servisa. Tek kad je on stigao, nastavilo se s izvršavanjem sljedeće naredbe. Asinkroni poziv omogućava suprotno – pozovemo li metodu asinkronim načinom, naša aplikacija će moći nastaviti s radom, a kad stigne odgovor web-servisa, izvršit će se određena akcija. Time štedimo vrijeme klijentske aplikacije koja poziva web-servis – ukoliko je njegovo izvršavanje dugo-trajno (bilo zbog kompliciranog zadatka koji mora obaviti ili zagušene internetske veze), aplikacija ne mora čekati već može nastaviti s izvršavanjem.

Iako je naš web-servis brz jer je njegov zadatak lagan (ne otvara bazu podataka, ne obavlja složene izračune), a i nalazi se na lokalnom stroju, što dodatno ubrzava komunikaciju, na njegovu primjeru ćemo pokazati asinkrono pozivanje.

Pojednostavljena verzija asinkronog poziva uključuje korištenje metoda *Begin* i *End* (njihov sufiks odgovara nazivu metode web-servisa). Tako ćemo pozvati *BeginPretvoriValutu* te time proslijediti



## 12. POGLAVLJE: WEB-SERVISI

parametre web-servisu i zadati mu posao, no u aplikaciji ćemo nastaviti s izvršavanjem drugog kôda i kasnije tek završiti komunikaciju s web-servisom pozivanjem metode *EndPretvoriValutu*.

Sljedeći izmijenjena verzija dijela prethodno prikazanog kôda:

```
TecajnaLista.Service1 tl = new TecajnaLista.Service1();
IAsyncResult rez = tl.BeginPretvoriValutu(Kolicina, Valuta, null, null);

// nastavljamo s izvršavanjem aplikacije

double Rezultat = tl.EndPretvoriValutu(rez);
```

Dakle, prvo pozivamo *BeginPretvoriValutu* koja, uz parametre metode *PretvoriValutu*, prima još dva parametra koje ćemo koristiti kasnije, a zasad ih postavljamo na vrijednosti *null*. Time zadajemo zadatak web-servisu i ostavljamo ga da obradi parametre i dostavi odgovor i rezultat.

Za asinkronu komunikaciju s web-servisom koristi se sučelje *IAsyncResult* koje služi za spremanje rezultata asinkronog poziva. To sučelje nije vezano isključivo uz rad s web-servisima, već asinkrono možete raditi i s drugim resursima u .NET-u. Pri pozivu naredbe *EndPretvoriValutu* prosljeđuje se taj objekt za asinkronu komunikaciju.

Ukoliko web-servis ne dostavi rezultat do trenutka kad se poziva *EndPretvoriValutu*, blokira se daljnje izvršavanje aplikacije sve dok ne stigne odgovor. Dakle, ista situacija kao kad se koristi sinkrono pozivanje web-servisa, samo što između *Begin* i *End* naredbe imate priliku izvršiti dio kôda u klijentskoj aplikaciji.

Možete iskoristiti i *IsCompleted* svojstvo objekta za asinkronu komunikaciju za provjeru je li poziv web-servisu završen i je li stigao odgovor. Tako možete izbjeći prerano pozivanje naredbe *End* i time blokiranje daljnjeg izvršavanja aplikacije.

```
double Rezultat;
if (rez.IsCompleted)
{
    Rezultat = tl.EndPretvoriValutu(rez);
}
else
{
    // izvrši još kôda i pokušaj ponovno kasnije
}
```

Drugi način izvršavanja asinkronih upita web-servisu je korištenje *callback* delegata odnosno pokazivača na metodu koja će se izvršiti kad se vrati rezultat asinkronog poziva. To je ujedno i najpraktičniji način izvršavanja asinkronih poziva jer se svodi na sljedeći scenarij: vi zadate upit web-servisu i

### III. DIO: DIJELOVI .NET-A

pokazivač na metodu koju treba izvršiti u trenutku kad web-servis vrati odgovor, a vaša aplikacija ostaje potpuno slobodna za daljnje izvršavanje kôda.

Za to će vam trebati pokazivač na metodu koja će se pozvati po primitku rezultata web-servisa. Evo kako bi izgledao kôd s korištenjem pokazivača na *callback* metodu:

```
TecajnaLista.Service1 tl;

private void button1_Click(object sender, System.EventArgs e)
{
    // isti kôd kao i u originalnom primjeru

    tl = new TecajnaLista.Service1();

    AsyncCallback povratnaMetoda = new AsyncCallback(RezultatStigao);

    tl.BeginPretvoriValutu(Kolicina, Valuta, povratnaMetoda, null);
}

private void RezultatStigao(IAsyncResult rez)
{
    double Rezultat = tl.EndPretvoriValutu(rez);
    MessageBox.Show(Rezultat.ToString() + " kn", "Rezultat");
}
```



**U metodi koja se izvršava na klik gumba u aplikaciji je izostavljen početni dio kôda, koji je isti kao i u prethodnim primjerima (provjeravanje sadržaja kontrola i postavljanje vrijednosti varijabli *Kolicina* i *Valuta*).**

Dakle, stvaramo delegat imena *povratnaMetoda* tipa *AsyncCallback* (kao što i ime kaže, radi se o *callback* metodi za asinkrone pozive). Pri njegovu stvaranju kao parametar koristimo ime metode koja će se pozvati kad stigne odgovor od web-servisa, intuitivnog imena *RezultatStigao*.

Potom samo pozivamo metodu *BeginPretvoriValutu*, a kao treći parametar prosljeđujemo netom stvoreni *callback* delegat imena *povratnaMetoda*.

## 12. POGLAVLJE: WEB-SERVISI

Sama *callback* metoda za parametar mora obavezno primiti objekt tipa *AsyncResult*, jer će u nje-mu biti spremljene informacije o rezultatu asinkronog poziva. U metodi nastavljamo poznatim kôdom – pozivom metode *EndPretvoriValutu* i objektom rezultata asinkronog poziva kao parametrom dobivamo rezultat poziva web-metode koji prikazujemo s *MessageBox.Show*.

U gornjem primjeru poziv *EndPretvoriValutu* neće blokirati daljnji rad aplikacije, već će se izvršiti odmah jer se nalazi u metodi koja se poziva tek u trenutku kad stigne rezultat web-servisa.



Iako je naš servis dosta brz te možda nije odmah vidljiva isplativost korištenja asinkronih poziva, nju je svakako korisno implementirati pri pozivanju web-servisa koji obavljaju složene operacije i onih smještenih na udaljenim poslužiteljima, jer su realne mogućnosti zagušenja mreže, što ne možete kontrolirati ni predvidjeti.

## Komandno-linijski alat *wsdl.exe*

Ključni korak u izgradnji aplikacije koja komunicira s web-servisom je izrada *proxy*-klase. I dok će Visual Studio .NET to automatski napraviti za vas u trenutku kad dodajete web-referencu svom projektu, dobro je znati što se događa u pozadini tog postupka.



Alat *wsdl.exe* se nalazi u direktoriju "C:\Program Files\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin" (naravno, pod uvjetom da ste Visual Studio .NET 2003 instalirali na *defaultnu* lokaciju). No najčešće ćete željeti *wsdl.exe* pozivati iz nekog drugog direktorija, vjerojatno iz onog u kojem je smještena aplikacija koja želi komunicirati s web-servisom. Kako ćete to raditi iz komandno-linijskog prompta (*Start – All Programs – Accessories – Command Prompt*), komplicirano bi bilo svaki put pisati punu putanju do *wsdl.exe* datoteke, pa je preporučljivo njen direktorij dodati u *path*. Tako ćete iz bilo kojeg direktorija moći pozivati sve datoteke iz tog direktorija samo navodeći njihovo ime. U komandno-linijskom promptu upišite sljedeću naredbu:

```
path=%path%;C:\Program Files\Microsoft Visual Studio .NET
2003\SDK\v1.1\Bin
```

Sad ćete moći iz bilo kojeg direktorija pozvati *wsdl.exe* bez navođenja pune putanje do njegove lokacije na disku.

No postoji i jednostavniji način – otvorite posebnu verziju komandne linije koja se nalazi u *Start – All Programs – Visual Studio .NET 2003 – Visual Studio .NET Tools – Visual Studio .NET 2003 Command Prompt*. Radi se o podešenoj verziji komandne linije koja već ima podešen *path*, pa su vam svi alati dostupni samo preko njihova imena.

### III. DIO: DIJELOVI .NET-A

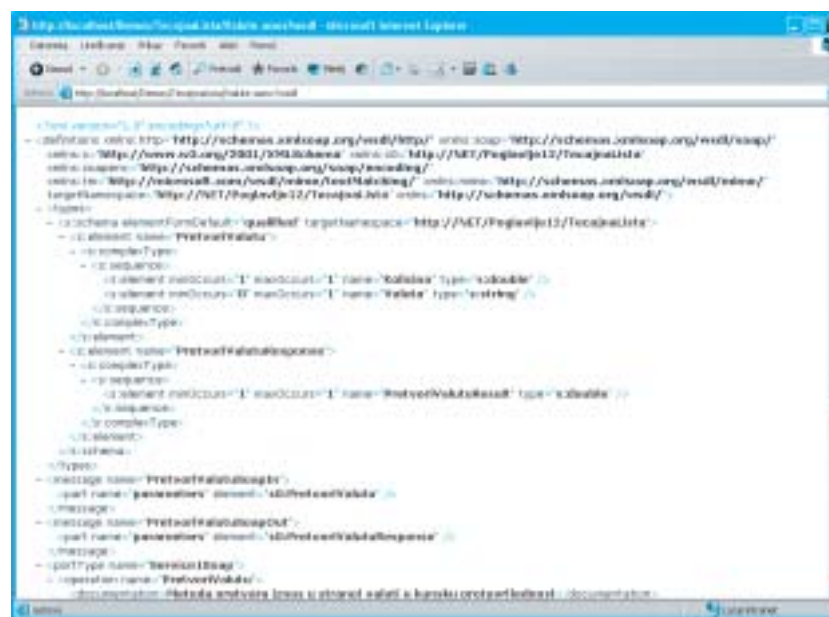
Naime, dok vi bezbrižno klikćete po Visual Studiju, u pozadini se poziva komandno-linijski alat `wsdl.exe` koji obavlja cijeli posao generiranja *proxy*-klase. U nastavku ćemo ručno upravljati alatom `wsdl.exe`, jer on nudi nekoliko zanimljivih opcija i predstavlja ključan korak u radu s web-servisima.

Korištenje `wsdl.exe` alata za stvaranje *proxy*-klase krajnje je jednostavno, izuzmemo li činjenicu da se radi u komandno-linijskom načinu rada, na što možda niste navikli. Želite li tako stvoriti *proxy*-klasu, za to će vam trebati WSDL opis web-servisa. Na svu sreću, uz svaki web-servis automatski se generira njegov WSDL opis tako da mu dodate sufiks “?wsdl”.

Primjerice, želite li stvoriti *proxy*-klasu za web-servis koji smo izradili u ovom poglavlju, mogli biste napisati:

```
wsdl http://localhost/Demos/TecajnaLista/Valute.aspx?wsdl
```

**Slika 12-14:**  
WSDL opis web-servisa možete vidjeti i u pregledniku upišete li adresu servisa i dodate li joj “?wsdl”



Dakle, proslijedili smo mu adresu web-servisa sa sufiksom “?wsdl”, što automatski daje njegov WSDL-opis. Evo kako se odvija komunikacija s alatom `wsdl.exe` (prijepis rezultata iz komandno-linijskog prompta).

```
E:\Web\test\TecajnaTest>wsdl http://localhost/Demos/TecajnaLista/Valute.aspx?wsdl
```

```
Microsoft (R) Web Services Description Language Utility
```

## 12. POGLAVLJE: WEB-SERVISI

```
[Microsoft (R) .NET Framework, Version 1.1.4322.573]
Copyright (C) Microsoft Corporation 1998-2002. All rights reserved.

Writing file 'E:\Web\test\TecajnaTest\Service1.cs'.

E:\Web\test\TecajnaTest>_
```

On je, dakle, stvorio CS datoteku istog imena kao i klasa web-servisa i zapisao ju je u direktorij iz kojeg smo ga pozvali.

**Pri korištenju alata `wsdl.exe` ne trebate web-servisu dodati sufiks “?wsdl”, već će se on, ukoliko to ne učinite, dodati automatski. Tako ste prethodni primjer mogli izvršiti naredbom:**

```
wsdl http://localhost/Demos/TecajnaLista/Valute.asmx
```

**No važno je znati da se za stvaranje *proxy*-klase koristi WSDL-opis web-servisa (zato se i alat za njeno generiranje zove `wsdl.exe`).**



Ukoliko pak želite napraviti *proxy*-klasu za web-servis koji nije pisan u .NET-u, možda vam neće biti dostupan njegov WSDL opis na isti način kao u gornjim primjerima (preko sufiksa “?wsdl”). U tom slučaju ćete vjerojatno imati običnu WSDL datoteku s opisom iz koje ćete generirati *proxy*-klasu. Tako alatu možete proslijediti običnu WSDL datoteku s opisom nekog web-servisa spremljenu na lokalnom računalu – rezultat je isti.

```
wsdl mojServis.wsdl
```

Kao što ste vidjeli, tako se stvara datoteka istog imena koje ima i klasa web-servisa. To možete promijeniti i generirati CS-datoteku drugog imena. Za to služi opcija `/out` koju zadajete `wsdl.exe` alatu.

```
wsdl /out:TecajnaProxy.cs http://localhost/Demos/TecajnaLista/Valute.asmx
```

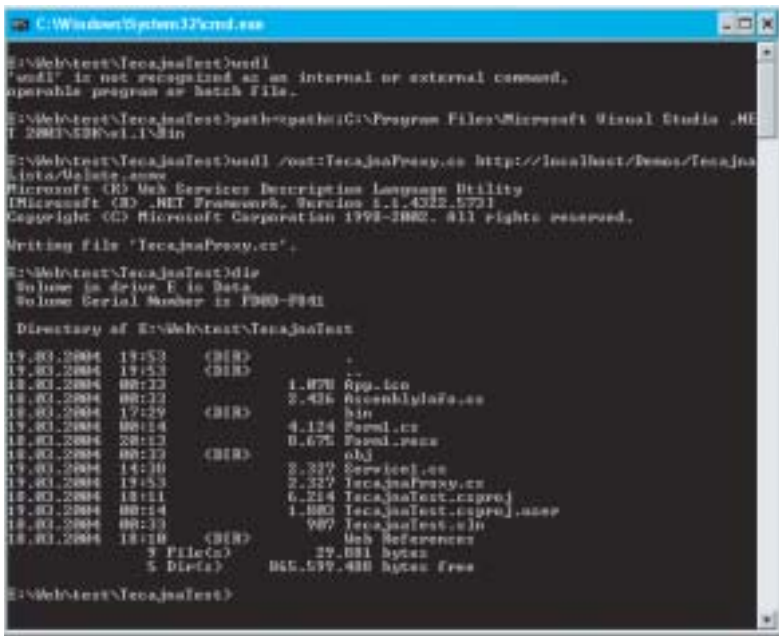
Kao što i u .NET-u imate mogućnost pisanja kôda u različitim jezicima, tako možete i *proxy*-klasu generirati u Visual Basic .NET jeziku, što vam može poslužiti kod proučavanja njenog sadržaja.

```
wsdl /language:vb http://localhost/Demos/TecajnaLista/Valute.asmx
```

Za to će vam, kao što vidite, poslužiti opcija `/language`, nakon koje odvojeno dvotočkom navodite jezik. Naravno, odabir jezika u kojem će biti napisana *proxy*-klasa nema nikakvog utjecaja na njen rad.

III. DIO: DIJELOVI .NET-A

**Slika 12-15:**  
**Primjer korištenja**  
**wSDL alata –**  
**primijetite postav-**  
**ljanje “patha” i**  
**stvaranje proxy-**  
**klase u datoteci s**  
**drugacijim imenom.**



Želite li da kôd generirane *proxy*-klase bude u posebnoj *namespaceu*, što može biti jako korisno ukoliko želite spriječiti podudaranja tipova definiranih u vašoj aplikaciji i onih u *proxy*-klasi, možete iskoristiti `/namespace` opciju.

`wsdl /namespace:WSTecajna http://localhost/Demos/TecajnaLista/Valute.asmx`

U nastavku slijedi tablica u kojoj će biti detaljnije opisane sve opcije `wsdl.exe` alata.

**Tablica 12-1:**  
**Opcije alata wsdl.exe**

Opcija	Opis
<code>/nologo</code>	miče “pozdravnu” poruku pri pokretanju, koja sadrži ime alata i verziju
<code>/language:&lt;jezik&gt;</code>	jezik koji će biti korišten za generiranje <i>proxy</i> -klase; možete izabrati između “CS”, “VB” ili “JS” ili ponuditi puno ime klase koja implementira <i>System.CodeDom.Compiler.CodeDomProvider</i> , čime imate mogućnost <i>proxy</i> -klasu generirati u bilo kojem .NET jeziku; <i>defaultna</i> postavka je “CS” odnosno C#; kratica za ovu opciju je <code>/l:&lt;jezik&gt;</code> .

## 12. POGLAVLJE: WEB-SERVISI

Opcija	Opis
/server	generira apstraktnu klasu za web-servis korištenjem ASP.NET-a
/namespace:<ime>	dozvoljava generiranje <i>proxy</i> -klase s posebnim <i>namespaceom</i> ; kratica za ovu opciju je /n:<ime>
/out:<datoteka>	ime datoteke pod kojom će se spremiti generirana klasa; ukoliko ne navedete ime, datoteka će dobiti ime iz klase web-servisa; kratica za ovu opciju je /o:<datoteka>
/protocol:<protokol>	korištenjem ove opcije možete prekoračiti <i>defaultni</i> SOAP protokol za komunikaciju s web-servisom i odabrati GET ili POST metode; dozvoljene opcije su "SOAP", "HttpGet" i "HttpPost"
/username:<korisničko_ime> /password:<zaporka> /domain:<domena>	postavke koje će se koristiti za spajanje na poslužitelj koji zahtijeva autentikaciju; kratice za ove opcije su /u:<korisničko_ime>, /p:<zaporka>, /d:<domena>.
/proxy:<url>	URL <i>proxy</i> servera za zadavanje HTTP zahtjeva; po <i>defaultu</i> će se koristiti sistemske postavke (Napomena: <i>proxy</i> server nema nikakve veze s <i>proxy</i> -klasom koja se generira, već je on dio postavki veze računala s Internetom.)
/proxyusername: <korisničko_ime> /proxypassword:<zaporka> /proxydomain:<domena>	postavke koje će se koristiti za spajanje preko <i>proxy</i> poslužitelja koji zahtijeva autentikaciju; kratice za ove opcije su /pu:<korisničko_ime>, /pp:<zaporka> i /pd:<domena>.
/appsettingurlkey:<ključ>	konfiguracijski ključ koji će se koristiti u kodu <i>proxy</i> -klase za čitanje postavki URL-a web-servisa; ako koristite konfiguracijske datoteke ne trebate imati URL web-servisa hardkodiran u <i>proxy</i> -klasi, već može biti smješten u konfiguracijskoj datoteci; kratica za ovu opciju je /urlkey:<ključ>.
/appsettingbaseurl: <bazni_URL>	bazni dio URL-a koji će se koristiti pri računanju adrese web-servisa (ona će biti relativna u odnosu na bazni URL definiran ovom opcijom); pri korištenju ove opcije obavezno treba koristiti i appsettingurlkey opciju; kratica za ovu opciju je /baseurl:<bazni_URL>.

*Proxy*-klasu generiranu wsdl.exe alatom možete kompajlirati u DLL datoteku i spremiti u direktorij *bin* unutar projekta u kojem želite komunicirati s web-servisom, a taj DLL možete referencirati korištenjem ključne riječi *using* u kodu aplikacije.

## III. DIO: DIJELOVI .NET-A

## SOAP ili ne-SOAP?

**P**roxy-klasa koju generirate pomoću `wsdl.exe` alata po *defaultu* će biti izvedena iz klase `System.Web.Services.Protocols.SoapHttpClientProtocol` koja omogućava pozivanje web-servisa korištenjem SOAP-a preko HTTP protokola, što je osnovna ideja i standardni način rada web-servisa. No vi ipak niste na to ograničeni – možete promijeniti protokol kojim će biti komunicirano s web-servisom korištenjem opcije `/protocol`. Sljedeći primjer tako stvara *proxy*-klasu izvedenu iz `HttpGetClientProtocol`, što znači da će se komunikacija s web-servisom odvijati preko metode GET protokola HTTP.

```
wsdl /protocol:httpget
http://localhost/Demos/TecajnaLi
sta/Valute.asmx
```

Na isti način ste mogli odlučiti koristiti metodu POST – kao protokol biste naveli “`httppost`”.

Zašto se uopće odlučiti na promjenu protokola za komunikaciju s web-servisom? U većini slučajeva, SOAP je najbolje rješenje. No SOAP se temelji na XML-u i sve poruke su “upakirane” u posebne *tagove*, tzv. SOAP omotnicu (engl. *envelope*). Komunicirate li s web-servisom prosleđujući i dobivajući natrag jednostavne tipove podataka (primjerice, brojeve), korištenjem GET ili POST metoda ti će pozivi biti malo efikasniji jer će se slati manje podataka preko mreže (neće biti okolnog XML kôda).



Ukoliko koristite Visual Studio .NET za rad s web-servisima, možete se potpuno pouzdati u njegovu *Add Web Reference* opciju za stvaranje *proxy*-klasa, jer se u pozadini ionako pokreće alat `wsdl.exe`. Ta opcija također može dodavati i UDDI web-servise, što vam dodatno olakšava rad. No ukoliko se nađete u situaciji da iz web-aplikacije smještene na nekom poslužitelju komunicirate s web-servisom koji se u međuvremenu malo promijenio, na tom poslužitelju najčešće nećete imati Visual Studio .NET i morat ćete se pouzdati u ručno generiranje novih *proxy*-klasa. Takav ćete način rada prakticirati i u slučajevima kad želite dodatno podesiti *proxy*-klase. No imajte na umu da ćete pri svakoj promjeni web-servisa morati ponovno generirati *proxy*-klasu – s druge strane, za takve situacije vam Visual Studio .NET pruža mnogo praktičniju opciju *Update Web Reference*.