41

Přístup k Internetu

V kapitolách 37 až 39 jste se dozvěděli, jak můžete v jazyce C# vytvářet velmi výkonné, účinné a dynamické webové stránky pomocí technologie ASP.NET. Klienty stránek ASP.NET představují obvykle uživatelé Internet Exploreru nebo jiných webových prohlížečů, jako jsou Opera nebo Firefox. Nicméně i ve vlastních aplikacích můžete využít možnosti prohlížení webu nebo programového získávání informací z webového serveru. Pokud jde o druhou možnost, je obvykle lepší implementovat webovou službu. Jestliže však přistupujete k veřejnému internetovému serveru, nemusíte mít možnost ovlivňovat jeho implementaci.

V této kapitole se zaměříte na využití základních tříd platformy .NET pro práci se síťovými protokoly, především pak s protokoly HTTP a TCP, které budete používat při klientském přístupu do různých sítí nebo k Internetu. Konkrétně se budete zabývat následujícími tématy:

- Stahování souborů ze sítě WWW
- Využití ovládacího prvku Webový prohlížeč (Web Browser) v formulářové aplikaci
- Manipulace s adresami IP a vyhledávání názvů DNS
- Programování soketů s třídami TCP, UDP a třídami soketů

V této kapitole se budeme věnovat některým nízkoúrovňovým způsobům, jak se k těmto protokolům dostat prostřednictvím platformy .NET Framework. Předvedeme si také další způsoby komunikace, které s pomocí těchto technologií pracují s položkami, například technologii Windows Communication Foundation (WCF), jíž se budeme věnovat v následující kapitole. Při vytváření síťových aplikací jsou nejdůležitější jmenné prostory `System.Net` a `System.Net.Sockets`. Jmenný prostor `System.Net` je věnován operacím vyšší úrovně, například stahování nebo odesílání souborů a vytváření webových požadavků pomocí protokolu HTTP nebo jiných protokolů, zatímco jmenný prostor `System.Net.Sockets` obsahuje třídy pro vykonávání operací nižší úrovně. Užitečnost těchto tříd oceníte v okamžiku, kdy budete chtít pracovat přímo se sokety nebo s protokoly, jako je TCP/IP. Metody definované v těchto třídách napodobují velmi věrně funkce Windows socket (Winsock) z API, odvozené od soketového rozhraní původně vyvinutého na univerzitě v Berkeley. Zjistíte také, že některé z objektů, s nimiž v kapitole pracujeme, se nachází ve jmenném prostoru System.IO.

Probíraná teorie a postupy programování sítí jsou v této kapitole praktiky doplněné vzorovými příklady. Nejedná se zde o průvodce světem počítačových sítí, ale o úvod do síťové komunikace prostřednictvím platformy .NET Framework.

Dále se podíváme na využití nového ovládacího prvku Webový prohlížeč (Web Browser) v prostředí formulářů Windows a také na to, jak vám tento ovládací prvek může usnadnit vykonávání některých úkolů spojených s přístupem k Internetu.

Začnete nejjednodušším případem odeslání požadavku na server a uložení informací vrácených v odpovědi. (Zdrojové kódy příkladů této kapitoly si můžete, stejně jako u ostatních kapitol, stáhnout z webu nakladatelství Computer Press na adrese `www.cpress.cz`.)

Třída WebClient

Jestliže si chcete pouze vyžádat soubor z určité adresy URI, je nejjednodušší použít třídu .NET `System.Net.WebCIient`. Je to třída velmi vysoké úrovně navržená pro realizaci základních operací prostřednictvím jednoho nebo dvou příkazů. Platforma .NET Framework v současné době podporuje adresy URI s identifikátory protokolů `http:`, `https:` a `file:`.

> Zřejmě byste měli vědět, že nové technické specifikace již nepoužívají pojem URL (Uniform Resource Locator) a dávají přednost termínu URI (Uniform Resource Identifier). Adresa URI má v podstatě stejný význam jako URL, ale je obecnější, protože nenaznačuje využití některého z důvěrně známých protokolů, jako je HTTP nebo FTP.

Stahování souborů

Pro stahování souborů pomocí instance třídy `WebCIient` jsou určeny dvě metody. Kterou z těchto metod si vyberete, závisí na tom, jak chcete obsah souboru zpracovat. Chcete-li soubor pouze uložit na disk, použijte metodu `DownloadFile()`. Tato metoda přebírá dva parametry: adresu URI souboru a místo uložení požadovaného souboru (cestu a název souboru):

```
WebCIient Client = new WebClient();
Client.DownloadFile("http://www.reuters.com/",   "ReutersHomepage.htm");
```

Ve svých aplikacích však budete zřejmě častěji zpracovávat data získaná z webového serveru, což vám umožní metoda `OpenRead()`. Tato metoda vrací odkaz na objekt typu `Stream`, který můžete použít pro načtení dat do paměti:

```
WebCIient Client = new WebClient();
Stream strm = Client.OpenReacK "http://www. reuters .com/");
```

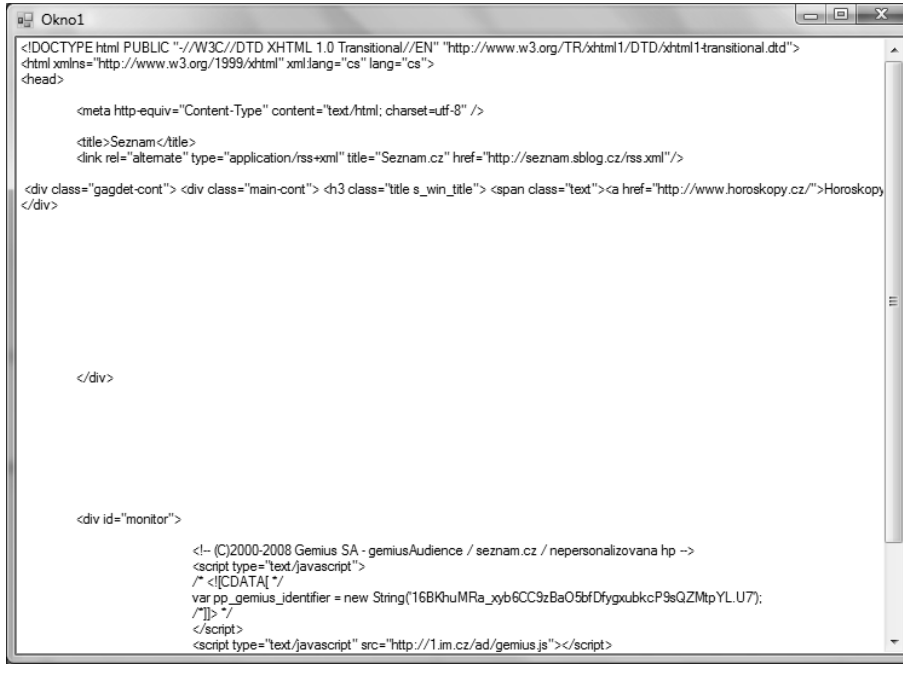
Příklad jednoduchého webového klienta

V prvním příkladu si vyzkoušíte využití metody `WebClient.OpenRead()` a obsah stažené stránky zobrazíte v ovládacím prvku typu `ListBox`. Začnete vytvořením nového projektu standardní formulářové aplikace v jazyce C#, na jejíž formulář vložíte ovládací prvek ListBox nazvaný `listBoxl` s vlastností `Dock` nastavenou na hodnotu `DockStyle.Fill`. Na začátek zdrojového souboru musíte prostřednictvím direktiv `using` vložit odkazy na jmenné prostory `System.Net` a `System.IO` Potom provedete v konstruktoru hlavního formuláře následující úpravy:

```
public Form1()
{
  InitializeComponent();
  WebClient Client = new WebClient();
  Stream strm = Client.OpenRead("http://www.seznam.cz");
  StreamReader sr = new StreamReader(strm);
  string line;
  while ( (line=sr.ReadLine()) != null )
  {
    listBox1.Items.Add(line);
  }
  strm.Close();
}
```

V tomto příkladu připojíte instanci třídy `StreamReader` definované ve jmenném prostoru `System.IO` k síťovému proudu. Pak můžete data z proudu získávat v textovém tvaru voláním metod vyšší úrovně, jako je `ReadLine()`. Jedná se o názornou ukázku výhodného zobecnění datových přenosů prostřednictvím proudu, o kterém jste se dozvěděli v kapitole 25, „Práce se soubory a systémovým registrem".

Výsledný vzhled spuštěného příkladu si můžete prohlédnout na obrázku 41.1:



Obrázek 41.1

Třída `WebClient` obsahuje také metodu `OpenWrite()`, která vrací proud, do něhož lze data zapisovat a tím je odesílat na příslušnou adresu URI, přičemž můžete určit metodu odesílání dat na vzdá-

lený počítač; implicitní metodou je POST. V následujícím kódu předpokládáte, že na místním počítači existuje adresář nazvaný `accept` s povoleným přístupem pro zápis. Tento kód vytvoří v uvedeném adresáři nový soubor `newfile.txt` obsahující text Ahoj lidi!:

```
WebClient webClient = new WebClient();
Stream stream = webClient.OpenWrite("http://localhost/accept/newfile.txt", "PUT");
StreamWriter streamWriter = new StreamWriter(stream);
streamWriter.WriteLine("Ahoj lidi!");
streamWriter.Close();
```

Odesílání souborů

Třída `WebClient` dále nabízí metody `UploadFile()` a `UploadData()`. Tyto metody použijete, když potřebujete odeslat formulář HTML nebo celý soubor. Metoda `UploadFile()` odešle soubor určený místním názvem na zadané místo, zatímco metoda `UploadData()` odešle na zadanou adresu URI binární data dodaná jako pole bajtů (třída `WebClient` obsahuje také metodu `DownloadData()`, která načte pole bajtů z adresy URI).

```
WebClient client = new WebClient();
client.UploadFile("http://www.ourwebsite.com/NewFile.htm",
  "C:\\WebSiteFiles\\NewFile.htm");
byte[] image;
// Kód pro inicializaci obrázku, aby obsahoval všechna binární
// data souboru jpg.
client.UploadData("http://www.ourwebsite.com/NewFile.jpg", image);
```

Třídy WebRequest a WebResponse

Přestože lze třídu `WebClient` používat velmi snadno, jsou její možnosti omezené. Konkrétně ji nelze použít pro přenos autentizačních pověření – což je při přenášení dat na webové servery poměrně vážný problém, protože jen málo serverů přijímá soubory bez předchozího ověření uživatele. Do záhlaví požadavků můžete vkládat různé informace a na druhé straně procházet záhlaví odezev, ale pouze velmi obecně – neexistuje totiž konkrétní podpora určitého protokolu. Důvodem je, že třída `WebClient` je navržena velmi univerzálně pro odesílání požadavků a příjem odezev prostřednictvím všech protokolů (například HTTP nebo FTP), ale neumí si poradit s charakteristickými vlastnostmi jednotlivých protokolů, jako jsou například soubory cookie v protokolu HTTP. Výhody těchto vlastností byste mohli využít pomocí tříd odvozených od základních tříd `WebRequest` a `WebResponse` ze jmenného prostoru `System.Net`.

Nejprve se podívejme, jak lze pomocí zmiňovaných tříd stáhnout webovou stránku. Jedná se o obměnu předchozího příkladu pomocí tříd `WebRequest a WebResponse`. V následujícím příkladu se seznámíte s hierarchií těchto tříd a dozvíte se, jak můžete v této hierarchii využít rozšiřující vlastnosti protokolu HTTP.

Níže uvedený kód ukazuje úpravy, které musíte udělat v příkladu `BasicWebClient`, abyste mohli využít tříd `WebRequest` a `WebResponse`:

```
public Form1()
{
```

```
  InitializeComponent();
  WebRequest wrq = WebRequest.Create("http://www.reuters.com");
  WebResponse wrs = wrq.GetResponse();
  Stream strm = wrs.GetResponseStream();
  StreamReader sr = new StreamReader(strm);
  string line;
  while ( (line = sr.ReadLine()) != null)
  {
    listBox1.Items.Add(line);
  }
  strm.Close();
}
```

Nejprve vytvoříte objekt představující webový požadavek, přičemž nepoužijete konstruktor, ale zavoláte statickou metodu `WebRequest.Create()`. Třída `WebRequest` je součástí hierarchie tříd podpory různých protokolů, o níž se více podrobností dozvíte později v této kapitole. Pro získání odkazu na správný objekt požadovaného typu je zde mechanismus implementovaný v metodě `WebRequest.Create()`, která vytvoří objekt vhodný pro zadaný protokol.

Třída `WebRequest` zastupuje požadavek na zaslání informací z určité adresy URL Tuto adresu předáte v parametru metody `Create()`. Objekt typu `WebResponse` reprezentuje data získaná ze serveru. Voláním metody `WebRequest.Create()` vlastně odešlete požadavek na webový server a vytvoříte objekt typu `WebResponse`, který použijete při zkoumání získaných dat. Podobně jako u instance třídy `WebClient` získáte i v tomto případě proud představující data, ale tentokrát voláním metody `WebResponse.GetResponseStream()`.

Další možnosti tříd WebRequest a WebResponse

V této podkapitole se dozvíte o několika dalších oblastech podporovaných třídami `WebRequest`, `WebResponse` a ostatními příbuznými třídami.

Informace v záhlaví HTTP

Velmi důležitou vlastností protokolu HTTP je možnost předávání velkého objemu informací v záhlavích tohoto protokolu s proudy požadavků i odezev. Tyto informace mohou obsahovat například soubory cookie nebo podrobnosti o webovém prohlížeči posílajícím příslušný požadavek (USER AGENT). Platforma .NET Framework, jak se správně domníváte, poskytuje plnou podporu přístupu k nejvýznamnějším datům. Třídy `WebRequest` a `WebResponse` podporují čtení informací ze záhlaví. Nicméně přístup k dodatečným informacím přenášeným pomocí protokolu HTTP nabízejí dvě od nich odvozené třídy `HttpWebRequest` a `HttpWebResponse`. Jak uvidíte později, příkaz pro vytvoření instance typu `WebRequest` s adresou URI protokolu HTTP skončí vytvořením instance typu `HttpWebRequest`. Protože třída `HttpWebRequest` je odvozena od třídy `WebRequest`, můžete použít novou instanci, i když je požadován objekt typu `WebRequest`. Navíc lze odkaz na získanou instanci přetypovat na odkaz typu `HttpWebRequest` a přistupovat k vlastnostem zaměřeným na protokol HTTP. Obdobně vrací volání metody `GetResponse()` při práci s protokolem HTTP odkaz na instanci typu `HttpWebResponse` jako odkaz na typ `WebResponse`. I v tomto případě lze výslednou instanci přetypovat a využít vlastnosti specifické pro protokol HTTP.

Obsah několika vlastností vyhrazených pro stejnojmenná záhlaví protokolu HTTP můžete prozkoumat pomocí následujícího kódu vloženého před volání metody `GetResponse()`:

```
WebRequest wrq = WebRequest.Create("http://www.seznam.cz");
HttpWebRequest hwrq = (HttpWebRequest)wrq;
listBox1.Items.Add("Request Timeout (ms) = " + wrq.Timeout);
listBox1.Items.Add("Request Keep Alive = " + hwrq.KeepAlive);
listBox1.Items.Add("Request AllowAutoRedirect = " + hwrq.AllowAutoRedirect);
```

Vlastnost `Timeout` je uváděna v milisekundách a implicitně obsahuje hodnotu 100 000. Změnou hodnoty této vlastnosti můžete řídit dobu čekání objektu typu `WebRequest` na odezvu před vyvoláním výjimky typu `WebException`. Příčiny vyvolání případné výjimky naleznete ve vlastnosti `WebException`, jež obsahuje hodnotu výčtového typu `WebExceptionStatus`. Tento výčet obsahuje hodnotu určující vypršení časového limitu, přerušená spojení, chyby protokolu a další.

Vlastnost `KeepAlive` je specifickým rozšířením protokolu HTTP, takže ji můžete používat pouze prostřednictvím odkazu typu `HttpWebRequest`. Vlastnost `KeepAlive` umožňuje použít jedno připojení pro několik požadavků a ušetřit tak čas, který byste jinak potřebovali na zavření a opětovné otevření připojení pro odeslání dalších požadavků. Vlastnost obsahuje implicitně hodnotu `true`.

Vlastnost `AllowAutoRedirect` je rovněž součástí třídy `HttpWebRequest` a určuje, zda má webový požadavek automaticky sledovat přesměrování vrácené webovým serverem. I tato vlastnost obsahuje implicitně hodnotu `true`. Chcete-li omezit počet přesměrování, nastavte vlastnost `MaximumAutomaticRedirections` třídy `HttpWebRequest` na požadovanou hodnotu.

Zatímco třídy požadavku a odezvy zpřístupňují většinu důležitých záhlaví v podobě specifických vlastností, vlastnost `Headers` umožňuje přístup k celé kolekci záhlaví. Zobrazení výpisu všech záhlaví v ovládacím prvku `ListBox` provedete přidáním následujícího kódu za volání metody `GetResponse()`:

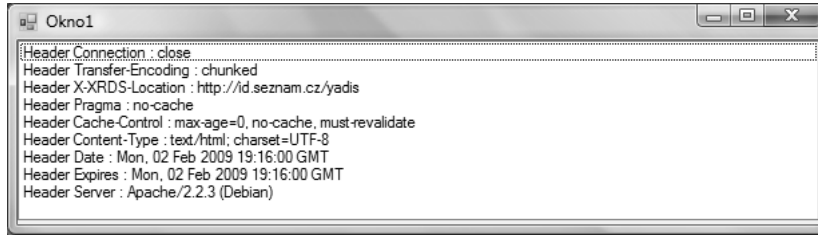
```
WebRequest wrq = WebRequest.Create("http://www.seznam.cz");
WebResponse wrs = wrq.GetResponse();
WebHeaderCollection whc = wrs.Headers;
for(int i = 0; i < whc.Count; i++)
{
  listBox1.Items.Add(string.Format("Header {0}: {1}", whc.GetKey(i), whc[i]));
}
```

Kód tohoto příkladu vytvoří seznam záhlaví uvedený na obrázku 41.2.

Autentizace

Třída `WebRequest` dále obsahuje vlastnost `Credentials`. Jsou-li pro úspěšné zpracování webového požadavku vyžadována pověření, můžete vytvořit instanci třídy `NetworkCredential` (rovněž ze jmenného prostoru `System.Net`), které předáte uživatelské jméno a heslo. Následující kód můžete vložit před volání metody `GetResponse()`.

```
NetworkCredential myCred = new NetworkCredential("myusername", "mypassword");
wrq.Credentials = myCred;
```



Obrázek 41.2

Práce s proxy

Ve sféře podnikání se mnoho firem musí potýkat s proxy serverem, který má na starosti všechny typy požadavků http a FTP. Proxy server, který ve firmě směřuje všechny požadavky a odpovědi, používá často nějakou formu zabezpečení (obyčejně je to jméno uživatele a heslo). V aplikacích, které používají objekty `WebClient` a `WebRequest`, budete možná muset po takových proxy serverech sáhnout. Podobně jako u předchozích objektů `NetworkCredential`, budete muset objekt `WebProxy` použít dříve, než uskutečníte vlastní volání dotazu.

```
WebProxy wp = new WebProxy("192.168.1.100", true);
wp.Credentials = new NetworkCredential("user1", "user1Password");
WebRequest wrq = WebRequest.Create("http://www.seznam.cz");
wrq.Proxy = wp;
WebResponse wrs = wrq.GetResponse();
```

Potřebujete-li navíc k osobním údajům uživatele nějak určit doménu, použijte při vytváření instance objektu typu `NetworkCredential` konstruktor s jinou signaturou:

```
WebProxy wp = new WebProxy("192.168.1.100", true);
wp.Credentials = new NetworkCredential("user1", "user1Password", "myDomain");
WebRequest wrq = WebRequest.Create("http://www.reuters.com");
wrq.Proxy = wp;
WebResponse wrs = wrq.GetResponse();
```

Asynchronní požadavky na stránky

Další z funkčností třídy `WebRequest` je schopnost posílat asynchronní požadavky na stránky. Jedná se o značnou výhodu, protože mezi odesláním požadavku a získáním odezvy může být dosti značná prodleva. Metody, jako jsou `WebClient.DownloadData()` a `WebRequest.GetResponse()`, nevracejí řízení volajícímu programu, dokud neobdrží celou odpověď. Zcela jistě byste nechtěli, aby vaše aplikace přestala reagovat jen kvůli dlouhé časové prodlevě. V takových případech je lepší použít metody `BeginGetResponse()` a `EndGetResponse()`. Metoda `BeginGetResponse()` pracuje asynchronně, a proto vrátí řízení volajícímu programu prakticky okamžitě. Běhová knihovna však na pozadí spustí pracovní podproces (vlákno, thread), který načítá odpověď ze serveru. Místo vrácení objektu typu `Web Response` vrací metoda `BeginGetResponse()` objekt obsahující implementaci rozhraní `IasyncResult`. Pomocí objektu s tímto rozhraním můžete čekat, až bude odpověď k dispozici, a pak zavolat metodu `EndGetResponse()` pro získání výsledků.

Metodě `BeginGetResponse()` můžete předat i delegát zpětného volání; některý představuje metodu vracející prázdný typ `void` a přijímající prostřednictvím argumentu odkaz typu `IAsyncResult`.

Běhová knihovna pak zavoláním delegátu oznámí dokončení úlohy pracovním podprocesem (vláknem). Z následujícího kódu vyplývá, že volání metody `EndGetResponse()` v metodě zpětného volání umožňuje získat odkaz na objekt typu `WebResponse`:

```
public Form1()
{
  InitializeComponent();
  WebRequest wrq = WebRequest.Create("http://www.reuters.com");
  wrq.BeginGetResponse(new AsyncCallback(OnResponse), wrq);
}
protected static void OnResponse(IAsyncResult ar)
{
  WebRequest wrq = (WebRequest)ar.AsyncState;
  WebResponse wrs = wrq.EndGetResponse(ar);
  // načtení odezvy ...
}
```

Všimněte si, že původní objekt `WebRequest` lze získat předáním příslušného odkazu metodě `BeginGetResponse()` ve druhém parametru, kterým je odkaz na objekt stavu. Při zpracování metody zpětného volání můžete získat stejný stavový objekt prostřednictvím vlastnosti `AsyncState` deklarované v rozhraní `IAsyncResult`.

Zobrazení výsledku ve tvaru stránky HTML

Uvedené příklady ukazují způsob využití základních tříd platformy .NET při snadném stahování a zpracovávání dat z Internetu. I když jste stažené soubory doposud zobrazovali pouze jako prostý text, častěji budete chtít prohlížet soubory HTML podobně jako Internet Explorer, kde vám vygenerovaný formát HTML umožňuje vidět, jak vlastně tento webový dokument vypadá. K dispozici bohužel není verze Internet Exploreru pro .NET, ale to neznamená, že tento úkol nelze snadno splnit. Před vydáním platformy .NET verze 2.0 a. 3.5 jste mohli využít odkaz na objekt COM, který zapouzdřoval Internet Explorer a prostřednictvím interoperability .NET jste mohli vložit do své aplikace funkčnost tohoto webového prohlížeče. Nyní můžete z formulářů v prostředí .NET Framework verze 2.0 používat nový vestavěný ovládací prvek `WebBrowser`.

Ovládací prvek `WebBrowser` zapouzdřuje objekt COM, čímž vám dále zjednodušuje práci. Kromě využití ovládacího prvku `WebBrowser` je další možností volat ze svého kódu instance Internet Exploreru.

Jestliže nechcete využít ovládací prvek `WebBrowser`, můžete programově spustit Internet Explorer a přejít na požadovanou webovou stránku prostřednictvím třídy `Process` ze jmenného prostoru `System.Diagnostics`:

```
Process myProcess = new Process();
myProcess.StartInfo.FileName = "iexplore.exe";
myProcess.StartInfo.Arguments = "http://www.cpress.cz";
myProcess.Start();
```

Tento kód však spustí Internet Explorer v samostatném okně. Vaše aplikace pak už nemá s novým oknem žádné spojení, takže nemůže tento prohlížeč nijak řídit.

Na druhé straně použitím ovládacího prvku WebBrowser získáte možnost zobrazovat a řídit prohlížeč, který se tak stává součástí vaší aplikace. Nový ovládací prvek WebBrowser je velmi propracovaný a nabízí množství vlastností, metod a událostí.

Jednoduché prohlížení webu z vašich aplikací

Pro zjednodušení začněte vytvořením okenní aplikace s jedním ovládacím prvkem `TextBox` a jedním ovládacím prvkem `WebControl`. Tato aplikace umožní uživateli vložit do textového pole adresu URL a po stisknutí tlačítka Enter provede ovládací prvek `WebControl` veškerou práci na získání webové stránky a zobrazení výsledného dokumentu.

V návrháři formulářů Visual Studia 2008 by měla vaše aplikace vypadat jako na obrázku 41.3

Obrázek 41.3

Jakmile uživatel zadá adresu URL a stiskne klávesu Enter, aplikace zachytí událost stisknutí této klávesy a nasměruje na požadovanou stránku ovládací prvek `WebBrowser`, který stránku po jejím získání zobrazí.

Kód aplikace by měl vypadat následovně:

```
using System;
using System.Windows.Forms;
namespace CSharpInternet
{
  partial class Form1: Form
  {
    public Form1()
    {
      InitializeComponent();
    }
    private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
    {
      if (e.KeyChar == (char)13)
```

```
        {
          webBrowser1.Navigate(textBox1.Text);
        }
      }
    }
}
```

V tomto příkladu zachytí obslužná metoda události `textBox1_KeyPress()` každé stisknutí klávesy uživatelem v textovém poli, a pokud je vloženým znakem návrat vozíku (stlačení klávesy Enter vkládá znak `(char) 13`), nasměrujete ovládací prvek `WebBrowser` prostřednictvím jeho metody `Navigate()` na zadanou adresu URL, kterou získáte jako řetězec z vlastnosti `textBox1.Text`. Výsledek si můžete prohlédnout na obrázku 41.4.

Obrázek 41.4

Spouštění instancí Internet Exploreru

Pokaždé nemusíte vkládat prohlížeč do svých aplikací, jak jste viděli v předchozí podkapitole; někdy budete chtít uživateli pouze umožnit zobrazení webové stránky v obvyklém prohlížeči (například po klepnutí na odkaz ve vaší aplikaci). Jako příklad této úlohy vytvořte okenní aplikaci s ovládacím prvkem `LinkLabel` na hlavním formuláři. Tomuto ovládacímu prvku můžete například přiřadit text „Navštivte webové stránky naší společnosti!“.

Nyní můžete pomocí následujícího kódu otevřít web vaší společnosti v nezávislém prohlížeči, na rozdíl od jeho přímého zobrazení ve formuláři vaší aplikace:

```
private void linkLabel1_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
  WebBrowser wb = new WebBrowser();
```

```
  wb.Navigate("http://www.cpress.cz", true);
}
```

Když uživatel klepne na ovládací prvek `LinkLabel`, vytvoříte novou instanci typu `WebBrowser`. Potom zavoláte metodu `Navigate()` třídy `WebBrowser` s umístěním webové stránky a booleovskou hodnotou, která určuje, zda bude požadovaná stránka otevřena uvnitř okenní aplikace (hodnota `false`) nebo v nezávislém prohlížeči (hodnota `true`). Implicitní hodnotou je `false`. Výše uvedený kód tedy po klepnutí uživatele na odkaz v okenní aplikaci zajistí vytvoření instance prohlížeče a okamžité otevření webu společnosti Computer Press (na adrese `www.cpress.cz`).

Přidání dalších možností prohlížeče IE do vašich aplikací

Při práci s předchozím příkladem, ve kterém jste použili ovládací prvek `WebBrowser` přímo ve formulářové aplikaci, si můžete všimnout, že po klepnutí na odkaz na stránce se text ovládacího prvku `TextBox` nezmění tak, aby zde byla vždy aktuální adresa URL. Tento nedostatek lze napravit zachycením událostí přicházejících z ovládacího prvku `WebBrowser` a vložením odpovídajících obslužných metod.

Aktualizace titulku formulářového okna titulkem stránky HTML je snadno proveditelná. Musíte pouze použít událost `Navigated` a aktualizovat vlastnost `Text` formuláře:

```
private void webBrowser1_Navigated(object sender, EventArgs e)
{
  this.Text = webBrowser1.DocumentTitle.ToString();
}
```

V tomto příkladě se ovládací prvek `WebBrowser` přesune na jinou stránku, spustí se událost `Navigated` a v důsledku toho se změní titulek formuláře na titulek zobrazené stránky. V některých instancích budete při práci s webovými stránkami, i když nezadáte konkrétní adresu, přesměrováni na zcela jinou webovou adresu. To zřejmě budete chtít v textovém poli formuláře (poli adresy) zobrazit. Uděláte to tak, že na zobrazované stránce zadáte do textového pole formuláře celou webovou adresu. K tomu lze použít událost `Navigated` ovládacího prvuk `WebBrowser`

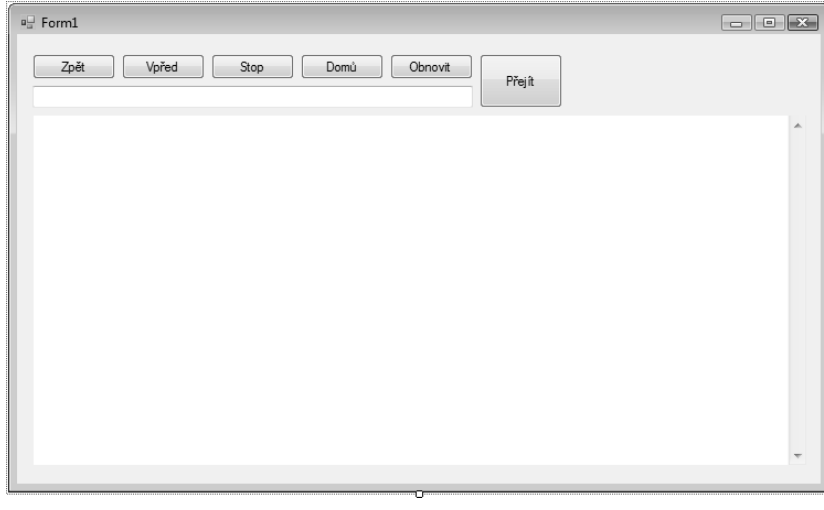
```
private void webBrowser1_Navigated(object sender, WebBrowserNavigatedEventArgs e)
{
  textBox1.Text = webBrowser1.Url.ToString();
  this.Text = webBrowser1.DocumentTitle.ToString();
}
```

Ovládací prvek `WebBrowser` vyvolá událost `Navigated`, jestliže přešel na nový dokument a začal s jeho stahováním. V takovém případě jednoduše aktualizujete vlastnost `Text` ovládacího prvku `textBoxl` adresou URL stahované stránky. To znamená, že když se během procesu stahování změní adresa URL (například díky přesměrování), objeví se v textovém poli nová adresa URL. Jestliže do příkladu přidáte výše uvedenou obslužnou metodu a otevřete web společnosti Wrox (`http://www.wrox.com`), uvidíte okamžitou změnu adresy URL na `http://www.wrox.com/Wiley CDA/`. Z tohoto chování je také zřejmé, že když uživatel klepne na odkaz ve stránce HTML, zobrazí se v textovém poli adresa URL nově požadované stránky.

Jestliže nyní spustíte aplikaci s výše uvedenými změnami, zjistíte, že titulek formuláře a pole adresy pracují stejně jako v aplikaci Microsoft Internet Explorer, viz obrázek 41.5.

Obrázek 41.5

Dalším krokem je vytvoření panelu nástrojů po vzoru aplikace Internet Explorer, který by uživateli umožňoval další řízení ovládacího prvku `WebBrowser`. To znamená vložení tlačítek Zpět, Vpřed, Stop, Obnovit a Domů.



Obrázek 41.6

Místo využití ovládacího prvku panelu nástrojů `ToolBar` vložíte do horní části formuláře, kde nyní máte pole adresy, několik tlačítek. Rozmístění všech pěti tlačítek si můžete prohlédnout na obrázku 41.6.

Text tlačítek změníte tak, aby vystihoval jejich funkci. Můžete samozřejmě jít ještě dál a použít „vypůjčené" obrázky tlačítek aplikace Internet Explorer sejmuté zachycením obrazovky. Identifikátory tlačítek upravte na `buttonBack`, `buttonForward`, `buttonStop`, `buttonRefresh` a `buttonHome`. Tlačítku Přejít přiřaďte název `buttonSubmit`.

Po spuštění aplikace byste měli znepřístupnit tlačítka `buttonBack`, `buttonForward` a `buttonStop`, protože bez zobrazené stránky v ovládacím prvku `WebBrowser` nemají význam. Později budete zpřístupňovat a znepřístupňovat tlačítka Zpět a Vpřed v závislosti na tom, kde se bude uživatel nacházet v zásobníku stránek. Při stahování stránky také zpřístupníte tlačítko Zastavit a po dokončení stahování ho zase znepřístupníte. Na stránce se bude nacházet také tlačítko Odeslat, které umožní odeslat požadavek na adresu URL.

Nejprve však přiřadíte požadovanou funkčnost tlačítkům. Třída `WebBrowser` obsahuje všechny metody, které k tomu potřebujete, takže tento úkol bude velmi jednoduchý:

```
using System;
using System.Windows.Forms;
namespace CSharpInternet
{
  partial class Form1: Form
  {
    public Form1()
    {
      InitializeComponent();
    }
    private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
    {
      if (e.KeyChar == (char)13)
      {
        webBrowser1.Navigate(textBox1.Text);
      }
    }
    private void webBrowser1_Navigated(object sender,
       WebBrowserNavigatedEventArgs e)
    {
      textBox1.Text = webBrowser1.Url.ToString();
      this.Text = webBrowser1.DocumentTitle.ToString();
    }
    private void Form1_Load(object sender, EventArgs e)
    {
      buttonBack.Enabled = false;
      buttonForward.Enabled = false;
      buttonStop.Enabled = false;
    }
    private void buttonBack_Click(object sender, EventArgs e)
```

```
{
  webBrowser1.GoBack();
  textBox1.Text = webBrowser1.Url.ToString();
}
private void buttonForward_Click(object sender, EventArgs e)
{
  webBrowser1.GoForward();
  textBox1.Text = webBrowser1.Url.ToString();
}
private void buttonStop_Click(object sender, EventArgs e)
{
  webBrowser1.Stop();
}
private void buttonHome_Click(object sender, EventArgs e)
{
  webBrowser1.GoHome();
  textBox1.Text = webBrowser1.Url.ToString();
}
private void buttonRefresh_Click(object sender, EventArgs e)
{
  webBrowser1.Refresh();
}
private void buttonSubmit_Click(object sender, EventArgs e)
{
  webBrowser1.Navigate(textBox1.Text);
}
private void webBrowser1_Navigating(object sender,
   WebBrowserNavigatingEventArgs e)
{
  buttonStop.Enabled = true;
}
private void webBrowser1_DocumentCompleted(object sender,
   WebBrowserDocumentCompletedEventArgs e)
{
  buttonStop.Enabled = false;
  if (webBrowser1.CanGoBack)
  {
     buttonBack.Enabled = true;
  }
  else
  {
     buttonBack.Enabled = false;
  }
  if (webBrowser1.CanGoForward)
  {
     buttonForward.Enabled = true;
```

```
      }
      else
      {
         buttonForward.Enabled = false;
      }
    }
  }
}
```

V tomto příkladu probíhá mnoho aktivit, protože uživatel má při používání aplikace spoustu možností. Každé události klepnutí na tlačítko jste přiřadili volání odpovídající metody třídy `WebBrowser`. Například po klepnutí na tlačítko Zpět jednoduše použijete metodu `GoBack()` ovládacího prvku `WebBrowser`. Pro tlačítko Vpřed máte k dispozici metodu `GoForward()`, pro tlačítko Stop využijete metodu `Stop()`, tlačítku Obnovit přidělíte metodu `Refresh()` a tlačítku Domů metodu `GoHome()`. Není tedy příliš složité vytvořit panel nástrojů s podobnou funkčností jako má Microsoft Internet Explorer.

Při prvním zavedení formuláře znepřístupní obslužná metoda `Forml_Load()` události `Load` příslušná tlačítka. V tomto formuláři může uživatel zadat do textového pole požadovanou adresu URL a klepnutím na tlačítko Přejít zobrazit odpovídající stránku.

Zpřístupňování a znepřístupňování tlačítek vyřešíte prostřednictvím několika událostí. Jak jste se již dříve dozvěděli, na začátku stahování zpřístupníte tlačítko Zastavit, k čemuž využijete událost `Navigating`:

```
private void webBrowser1_Navigating(object sender, WebBrowserNavigatingEventArgs e)
{
  buttonStop.Enabled = true;
}
```

Tlačítko Zastavit opět znepřístupníte po dokončení stahování dokumentu:

```
private void webBrowser1_DocumentCompleted(object sender,
   WebBrowserDocumentCompletedEventArgs e)
{
  buttonStop.Enabled = false;
}
```

Zpřístupnění a znepřístupnění tlačítek Zpět a Vpřed závisí na možnosti pohybu vpřed a zpět v zásobníku stránek. Tuto funkčnost naprogramujete prostřednictvím událostí `CanGoForwardChanged` a `CanGoBackChanged`:

```
private void webBrowser1_CanGoBackChanged(object sender, EventArgs e)
{
  if (webBrowser1.CanGoBack == true)
  {
     buttonBack.Enabled = true;
  }
  else
  {
     buttonBack.Enabled = false;
  }
}
```

```
private void webBrowser1_CanGoForwardChanged(object sender, EventArgs e)
{
  if (webBrowser1.CanGoForward == true)
  {
     buttonForward.Enabled = true;
  }
  else
  {
     buttonForward.Enabled = false;
  }
}
```

Nyní spusťte aplikaci, navštivte různé webové stránky a klepněte na pár odkazů. Při procházení webových stránek by vám také měla pomoci tlačítka panelu nástrojů. Výslednou aplikaci si můžete prohlédnout na obrázku 41.7.

Obrázek 41.7

Tisk pomocí ovládacího prvku WebBrowser

Uživatelé mohou pomocí ovládacího prvku `WebBrowser` nejen zobrazovat stránky a dokumenty, ale také je odesílat k vytištěné na tiskárnu. Dokument zobrazený v tomto ovládacím prvku vytisknete následujícím příkazem:

```
webBrowser1.Print();
```

Chcete-li stránku nebo dokument vytisknout, nemusíte ho, stejně jako v příchozích případech, ani prohlížet. Pomocí třídy `WebBrowser` můžete například provedením následujícího kódu stáhnout a vytisknout dokument HTML, aniž byste ho nejprve zobrazili:

```
WebBrowser wb = new WebBrowser();
wb.Navigate("http://www.cpress.cz");
wb.Print();
```

Výpis kódu požadované stránky

Na začátku této kapitoly jste zobrazovali kód požadované stránky pomocí tříd `WebRequest` a `Stream`. K tomuto účelu jste použili následující kód:

```
public Form1()
{
  InitializeComponent();
  System.Net.WebClient Client = new WebClient();
  Stream strm = Client.OpenRead("http://www.reuters.com");
  StreamReader sr = new StreamReader(strm);
  string line;
  while ( (line=sr.ReadLine()) != null )
  {
    listBox1.Items.Add(line);
  }
  strm.Close();
}
```

Stejný výsledek snadno získáte i pomocí ovládacího prvku `WebBrowser`. Můžete například pozměnit předchozí aplikaci – prohlížeč – přidáním jednoho řádku do obslužné metody `webBrowser1_DocumentCompleted()`:

```
private void webBrowser1_DocumentCompleted(object sender,
   WebBrowserDocumentCompletedEventArgs e)
{
  buttonStop.Enabled = false;
  textBox2.Text = webBrowser1.DocumentText.ToString();
}
```

Do vlastní aplikace přidejte pod ovládací prvek `WebBrowser` další textové pole. U vyžádané webové stránky nyní uživateli zobrazíte nejen vizuální podobu, ale také její kód v ovládacím prvku `TextBox`. Kód webové stránky získáte pomocí vlastnosti `DocumentText` ovládacího prvku `WebBrowser`. Tato vlastnost obsahuje celou stránku ve tvaru řetězce. Další možností je získat obsah stránky

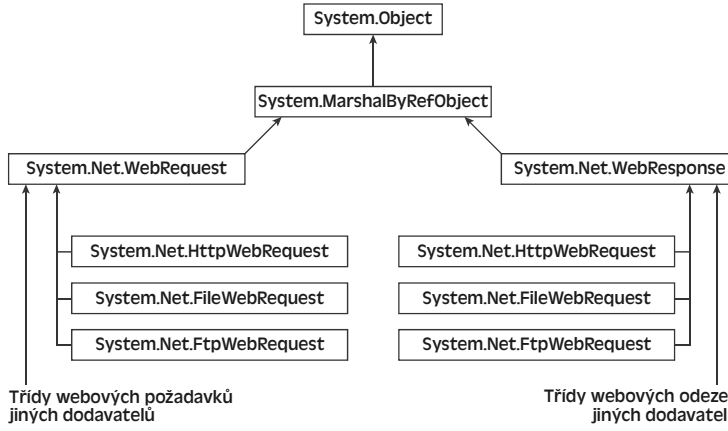
z vlastnosti `DocumentStream` jako proud typu `Stream`. Výsledek přidání druhého textového pole pro zobrazení obsahu stránky ve formátu řetězce (typ `string`) si můžete prohlédnout na obrázku 41.8.

Obrázek 41.8

Hierarchie tříd WebRequest a WebResponse

V této podkapitole se blíže podíváte na architekturu tříd `WebRequest` a `WebResponse`. Hierarchie zmiňovaných tříd je znázorněna na obrázku 41.9.

Tato hierarchie obsahuje více tříd než jen dvě, které jste použili v kódu. Kromě toho byste měli vědět, že třídy `WebRequest` a `WebResponse` jsou abstraktní, takže nelze vytvářet jejich instance. Tyto základní třídy nabízejí obecné funkce pro ošetření webových požadavků a odpovědí nezávisle na protokolu použitém pro danou operaci. Požadavky jsou odesílány prostřednictvím různých protokolů (HTTP, FTP, SMTP atd.) a příslušný požadavek vždy vyřídí instance odvozené třídy napsané pro daný protokol. Společnost Microsoft toto schéma označuje termínem vyměnitelné protokoly (pluggable protocols). V jedné z předchozích ukázek jste definovali proměnné jako odkazy na základní třídy, ovšem metoda `WebRequest.Create()` ve skutečnosti vracela instanci třídy `httpWebRequest`, zatímco metoda `GetResponse()` vracela instanci třídy `HttpWebResponse`. Mechanismus vytváření tříd založený na volání metod zakrývá v klientském kódu množství podrobností a podporuje rozmanité množství protokolů při využití stejného základního kódu.



Obrázek 41.9

Potřeba objektu schopného pracovat s protokolem HTTP je zřejmá z adresy URI předané metodě `WebRequest.Create()`. Tato metoda zjistí identifikátor protokolu zadaný v adrese URI a vrátí objekt příslušné třídy. Díky tomu se ve svém kódu nemusíte zabývat odvozenými třídami a problematikou použitého protokolu. Potřebujete-li přístup k funkčnosti charakteristické pro určitý protokol, můžete použít vlastnosti a metody odvozené třídy. V takovém případě přetypujete odkaz typu `WebRequest` nebo `WebResponse` na odkaz požadovaného typu.

Pomocí popisované architektury můžete odesílat požadavky prostřednictvím kteréhokoli z běžně používaných protokolů. Společnost Microsoft však v současné době poskytuje odvozené třídy pouze pro protokoly HTTP, HTTPS, FTP a FILE. Protokol FTP je nejnovější možností, kterou zavádí platforma .NET Framework verze 2.0. Chcete-li používat jiné protokoly, například SMTP, musíte se obrátit na organizaci Windows Communication Foundation a vrátit se k funkcím rozhraní Windows API, nebo použít objekt typu `SmtpClient`.

Pomocné třídy

V této podkapitole se seznámíte s několika užitečnými třídami, které podstatně zjednodušují programování pro web a práci s adresami URI a IP.

Třídy URI

Třídy `Uri` a `UriBuilder` ze jmenného prostoru `System` (nikoli `System.Net`) zastupují adresu URI. Třída `UriBuilder` podporuje vytváření adres URI z jednotlivých řetězcových složek, zatímco třída `Uri` umožňuje adresy URI zpracovávat, spojovat a porovnávat.

Konstruktor třídy `Uri` vyžaduje řetězec s úplnou adresou URI.

```
Uri MSPage = new
Uri("http://www.Microsoft.com/SomeFolder/SomeFile.htm?Order=true");
```

Tato třída obsahuje mnoho vlastností určených jen ke čtení, takže její objekty nelze po vytvoření upravovat:

```
string Query = MSPage.Query;                    // Order=true;
string AbsolutePath = MSPage.AbsolutePath;      // SomeFolder/SomeFile.htm
string Scheme = MSPage.Scheme;                  // http
int Port = MSPage.Port;                         // 80  (implicitní pro http)
string Host = MSPage.Host;                      // www.Microsoft.com
bool IsDefaul tPort = MSPage. IsDefaul tPort;   // true, port 80 je totiž implicitní
```

Naproti tomu třída `UriBuilder` implementuje podstatně méně vlastností. Pouze tolik, aby umožnila sestavení úplné adresy URI. Vlastnosti této třídy jsou určeny pro čtení i zápis.

Konstruktoru třídy `UriBuilder` lze předat jednotlivé složky adresy URI jako samostatné hodnoty:

```
Uri MSPage =
  new UriBuilder("http", "www.Microsoft.com", 80, "SomeFolder/SomeFile.htm");
```

Adresu však můžete vytvořit i přiřazením jednotlivých hodnot příslušným vlastnostem:

```
UriBuilder MSPage = new UriBuilder();
MSPage.Scheme = "http";
MSPage.Host = "www.Microsoft.com";
MSPage.Port = 80;
MSPage.Path = "SomeFolder/SomeFile.htm";
```

Po dokončení inicializace objektu třídy `UriBuilder` můžete získat objekt typu `Uri` prostřednictvím vlastnosti `Uri`:

```
Uri CompletedUri = MSPage.Uri;
```

Adresy IP a jména DNS

V síti Internet lze rozpoznávat servery a klienty podle adres IP nebo jmen DNS (jméno hostitele, hostname). Název DNS je srozumitelný název, který můžete zadat do okna webového prohlížeče (například `www.cpress.cz` nebo `www.microsoft.com`). Adresy IP využívají počítače pro vzájemnou identifikaci. Tyto identifikátory ve formě adres IP zajišťují doručování webových požadavků a odpovědí na správná místa. Každý počítač přitom může mít více než jednu adresu IP.

Dnes mají IP adresy obyčejně 32bitové hodnoty. Příkladem takové 32bitové adresy je 192.168.1.100. Tento formát IP adres označujeme jako Internetový protokol verze 4. Protože je však dnes k Internetu připojeno značné množství různých počítačů a dalších zařízení, byl vyvinut nový adresný systém – Internetový protokol verze 6. IPv6 nabízí 64bitové adresy IP. IPv6 může potenciálně zvládnout maximálně $3x10^{28}$ unikátních adres. Zjistíte, že platforma .NET umožní vašim aplikacím pracovat jak s protokolem IPv4, tak i s protokolem IPv6.

Chcete-li použít název DNS, musíte nejprve odeslat síťový požadavek na překlad názvu DNS na adresu IP, což je úloha pro servery DNS.

Server DNS udržuje tabulku vzájemného přiřazení názvů DNS a adres IP pro všechny počítače, o nichž ví, že existují. Kromě toho uchovává adresy IP dalších serverů DNS, které mohou vyhledat název DNS, o němž dotyčný server nic neví. Váš místní počítač by měl znát vždy alespoň jeden server DNS. Správce sítě konfiguruje tuto informaci při instalaci počítače.

Před odesláním požadavku váš počítač nejprve požádá server DNS o překlad názvu DNS na příslušnou adresu IP. Jakmile má tuto informaci k dispozici, může adresovat požadavek a odeslat jej do sítě. Všechny popsané operace probíhají obvykle při prohlížení webu bez vědomí uživatelů.

Třídy .NET pro práci s adresami IP

Platforma .NET Framework nabízí mnoho tříd, které vám mohou pomoci při vyhledávání adres IP a informací o hostitelských počítačích.

Třída IPAddress

Třída `IPAddress` zastupuje adresu IP. Adresa samotná je dostupná prostřednictvím vlastnosti `Address` a lze ji metodou `ToString()` převést na desítkový formát s tečkami. Třída `IPAddress` implementuje rovněž statickou metodu `Parse()` pro obrácenou operaci – převod řetězce s adresou v desítkovém tečkovaném formátu zpět na objekt typu `IPAddress`.

```
IPAddress ipAddress = IPAddress.Parse("234.56.78.9");
byte[] address = ipAddress.GetAddressBytes();
string ipString = ipAddress.ToString();
```

V této ukázce přiřadíte proměnné `address` adresu IP ve tvaru pole celých čísel typu `byte` a proměnné `ipString` řetězec `"234.56.78.9"`.

Vedle toho obsahuje třída `IPAddress` několik konstantních statických složek, v nichž jsou uloženy speciální adresy. Například adresa `Loopback` umožňuje počítači posílat zprávy sám sobě, zatímco adresa `Broadcast` zprostředkovává všesměrové vysílání (multicasting) v místní síti:

```
// Následující řádek přiřadí proměnné loopback řetězec "127.0.0.1".
// Adresa loopback označuje místní počítač,
string loopback = IPAddress.Loopback.ToString();
// Následující řádek přiřadí proměnné broadcast  řetězec  "255.255.255.255".
// Tato adresa se používá  při rozesílání zprávy všem počítačům v místní síti.
string broadcast = IPAddress.Broadcast.ToString();
```

Třída IPHostEntry

Třída `IPHostEntry` zapouzdřuje informace související s konkrétním hostitelským počítačem. Tato třída zpřístupňuje název DNS dotyčného počítače prostřednictvím vlastnosti `HostName` (typu `string`) a vlastnost `AddressList` vrací pole objektů typu `IPAddress`. Třídu `IPHostEntry` použijete v následujícím příkladu DnsLookup.

Třída Dns

Třída `Dns` umí komunikovat s implicitním serverem DNS a jeho prostřednictvím získávat adresy IP. Tato třída obsahuje dvě velmi důležité statické metody. Statická metoda `Resolve()` se spojí se serverem DNS a získá podrobnosti o hostitelském počítači podle názvu DNS. Statická metoda `GetHostByAddress()` rovněž vrací podrobnosti o hostitelském počítači, ale tentokrát na základě adresy IP. Obě metody vracejí objekt typu `IPHostEntry`.

```
IPHostEntry wroxHost = Dns.Resolve("www.cpress.cz");
IPHostEntry wroxHostCopy = Dns.GetHostByAddress("208.215.179.178");
```

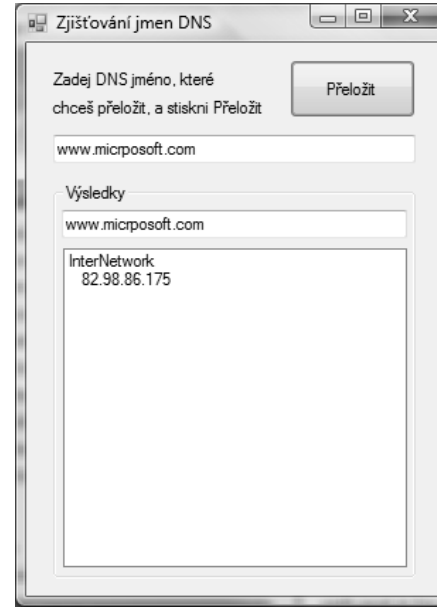
V tomto kódu budou oba objekty typu `IPHostEntry` obsahovat podrobnosti o serverech CPress.cz. Třída `Dns` se liší od tříd `IPAddress` a `IPHostEntry` tím, že umí komunikovat se servery a získávat od nich informace. Třídy `IPAddress` a `IPHostEntry` jsou v podstatě pouze jednoduché datové struktury s užitečnými vlastnostmi, které umožňují přístup k základním datům.

Příklad DnsLookup

Třídy určené pro práci s názvy DNS a adresami IP si můžete vyzkoušet v následujícím příkladu DnsLookup (viz obrázek 41.10).

Tato vzorová aplikace požádá uživatele o zadání DNS jména do hlavního textového pole. Když uživatel klepne na tlačítko Přeložit, zavoláte metodu `Dns.Resolve()`, čímž získáte odkaz typu `IPHost Entry` a zobrazíte název hledaného hostitele a jeho adresu IP. Všimněte si, že se jméno hledaného hostitele může lišit od zadaného jména. K tomu dojde například v situaci, kdy je jedno jméno DNS (`www.microsoft.com`) pouze zástupným jménem pro jiné jméno DNS (`lb1.www.ms.akadns. net`).

Aplikace DnsLookup je standardní okenní aplikací v C#. Ovládací prvky vložíte podle obrázku 35.12 a dáte jim jména `txtBoxInput`, `btnResolve`, `txtBoxHostName` a `listBoxIPs`. Potom pouze vložíte do třídy `Forml` následující obslužnou metodu `btnResolve_Click()` události klepnutí na tlačítko Přeložit (`btnResolve`):



Obrázek 41.10

```
void btnResolve_Click (object sender, EventArgs e)
{
  try
  {
    IPHostEntry iphost = Dns.GetHostEntry(txtBoxInput.Text);
    foreach (IPAddress ip in iphost.AddressList)
    {
      string ipaddress = ip.AddressFamily.ToString();
      listBoxIPs.Items.Add(ipaddress);
      listBoxIPs.Items.Add("  " + ip.ToString());
    }
    txtBoxHostName.Text = iphost.HostName;
  }
  catch(Exception ex)
  {
    MessageBox.Show("Požadavek se nepodařilo zpracovat, " +
      "protože nastal následující problém:\n" + ex.Message, "Vznikla výjimka");
  }
}
```

Všimněte si, že v kódu zachycujete vznik možných výjimek. K výjimce může dojít například v případě, že uživatel zadá neplatný název DNS, nebo při výpadku sítě.

Po obdržení instance typu `IPHostEntry` získáte pomocí její vlastnosti `AddressList` pole adres IP, které můžete procházet v cyklu `foreach`. U každého záznamu zobrazíte adresu IP jako celé číslo a pomocí metody `IPAddress.AddressFamily.ToString()` jako řetězec.

Protokoly nižší úrovně

Tato podkapitola je věnovaná stručnému popisu tříd .NET používaných ke komunikaci na nižší úrovni. Síťová komunikace probíhá na několika různých úrovních. Třídy, kterými jste se dosud v této kapitole zabývali, pracují na nejvyšší úrovni, kde jsou zpracovávány konkrétní příkazy. Tuto problematiku zřejmě nejlépe pochopíte, jestliže se podíváte na přenosy souborů pomocí protokolu FTP. I když dnešní aplikace s grafickým uživatelským rozhraním skrývají mnoho podrobností protokolu FTP, není to tak dávno, co se běžně zadávaly příkazy FTP z příkazového řádku. V tomto prostředí jste mohli explicitně psát příkazy odesílané na server, jimiž jste požadovali stažení souboru, jeho přenos na server nebo výpis souborů v serverovém adresáři.

FTP není jediným protokolem vyšší úrovně založeným na textových příkazech. Na podobném principu pracují rovněž protokoly HTTP, SMTP, POP a další. I v těchto případech mnoho moderních nástrojů s grafickým uživatelským rozhraním skrývá před uživateli přenosové příkazy, takže s nimi už ani nepřijdete do styku. Jestliže například zadáte webovému prohlížeči adresu URL a pošlete tento webový požadavek na server, prohlížeč ve skutečnosti odešle na server příkaz GET (v podobě prostého textu). Tento příkaz slouží stejnému účelu jako příkaz `get` používaný protokolem FTP. Prohlížeč může odeslat rovněž příkaz `POST`, kterým říká, že k požadavku připojil další data.

Zmiňované protokoly však nepokrývají všechny požadavky na komunikaci mezi dvěma počítači. I kdyby oba počítače rozuměly například protokolu HTTP, stále by spolu nemohly komunikovat, pokud by se nedomluvily na přesném způsobu přenosu znaků: Jaký použijí binární formát? A jaké elektrické napětí bude vyjadřovat jedničky a nuly na nejnižší úrovni? Vzhledem k tomu, že je nutno konfigurovat mnoho položek a dohodnout se na mnoha dalších věcech, hovoří vývojáři a hardwaroví inženýři o zásobníku protokolů (protocol stack). Vytvoříte-li seznam všech protokolů a dalších mechanismů nezbytných pro komunikaci mezi dvěma počítači, dostanete zásobník protokolů s protokoly vyšší úrovně nahoře a protokoly nižší úrovně dole. Tento přístup položil základy modulárního a vrstveného pojetí protokolů s cílem dosáhnout co nejefektivnější komunikace.

Většina vývojářů naštěstí nemusí jít až na samé dno zmiňovaného zásobníku a pracovat na úrovni napětí. Píšete-li však kód, který vyžaduje efektivní komunikaci mezi dvěma počítači, není vůbec neobvyklé, že se dostanete na úroveň přenosů paketů obsahujících binární data. Zde vstupujete do oblasti protokolů, jako je TCP, a společnost Microsoft připravila několik tříd, které vám umožní pohodlnou práci s binárními daty i na této úrovni.

Třídy nižší úrovně

Třídy určené pro komunikaci na nižší úrovni najdete ve jmenném prostoru `System.Net.Sockets`. Tyto třídy umožňují přímé odeslání síťových požadavků TCP, nebo naslouchání síťovým požadavkům TCP na určitém portu. V následující tabulce si můžete prohlédnout popis hlavních tříd.

Třída	Popis
`Socket`	Třída nízké úrovně, která se zabývá správou spojení. Třídy jako `WebRequest`, `TcpClient` a `UdpClient` vnitřně pracují s touto třídou.
`NetworkStream`	Potomek třídy `Stream`. Zastupuje proud dat plynoucích ze sítě.
`SmtpClient`	Pomocí protokolu SMPT (Simple Mail Transfer Protocol) vám umožní odesílat zprávy (poštu).
`TcpClient`	Umožňuje vytvářet a používat spojení TCP.
`TcpListener`	Umožňuje naslouchat příchozím požadavkům na spojení TCP.
`UdpClient`	Umožňuje vytvářet spojení pro klienty UDP. (Protokol UDP je alternativou protokolu TCP, používá se však mnohem méně, většinou v místních sítích.)

Využití třídy SmtpClient

Třída `SmtpClient` vám umožní poslat poštovní zprávu prostřednictvím protokolu SMTP. My zde uvádíme jednoduchý příklad použití objektu typu `SmtpClient`:

```
SmtpClient sc = new SmtpClient("mail.mySmtpHost.com");
sc.Send("evjen@yahoo.com", "editor@wrox.com",
   "Nejnovější kapitola", "Zde se nachází nejnovější.");
```

Ve své nejjednodušší formě zpracováváte instanci typu SmtpClient. V našem příkladě obsahovala instalace i hostitele serveru SMTP, který se používá k posílání poštovních zpráv do sítě Internet. Téhož byste dosáhli i pomocí vlastnosti `Host`.

```
SmtpClient sc = new SmtpClient();
sc.Host = "mail.mySmtpHost.com";
sc.Send("evjen@yahoo.com", "editor@wrox.com",
   "Nejnovější kapitola", "Zde se nachází nejnovější.");
```

Jakmile umístíte objekt typu `SmtpClient`, bude již stačit pouze zavolat metodu `Send()` a poskytnout adresu odesilatele (From), adresáta (To) a předmět, který bude následován samotným tělem (Body) zprávy.

Často se setkáte se složitějšími zprávami, než jaké si zde popisujeme. Když budete s takovými složitějšími zprávami chtít pracovat, můžete metodě `Send()` předat objekt typu `MailMessage`.

```
SmtpClient sc = new SmtpClient();
sc.Host = "mail.mySmtpHost.com";
MailMessage mm = new MailMessage();
mm.Sender = new MailAddress("evjen@yahoo.com", "Bill Evjen");
mm.To.Add(new MailAddress("editor@wrox.com", "Katie Mohr"));
mm.To.Add(new MailAddress("marketing@wrox.com", "Wrox Marketing"));
mm.CC.Add(new MailAddress("publisher@wrox.com", "Joe Wikert"));
mm.Subject = "Poslední kapitola";
mm.Body = "<b>Zde můžete vložit dlouhou zprávu</b>";
mm.IsBodyHtml = true;
```

```
mm.Priority = MailPriority.High;
sc.Send(mm);
```

Když použijete objekt typu `MailMessage`, budete si moci vyladit způsob sestavování poštovní zprávy. Můžete posílat zprávy ve formátu HTML, do polí adresáta (To) a kopií (CC) lze přidat libovolný počet adres, budete také moci měnit priority zpráv, upravovat kódování zpráv a připojovat přílohy. V následujícím výtažku kódu vidíte, jak je možno definovat připojování příloh.

```
SmtpClient sc = new SmtpClient();
sc.Host = "mail.mySmtpHost.com";
MailMessage mm = new MailMessage();
mm.Sender = new MailAddress("evjen@yahoo.com", "Bill Evjen");
mm.To.Add(new MailAddress("editor@wrox.com", "Katie Mohr"));
mm.To.Add(new MailAddress("marketing@wrox.com", "Wrox Marketing"));
mm.CC.Add(new MailAddress("publisher@wrox.com", "Joe Wikert"));
mm.Subject = "Poslední kapitola";
mm.Body = "<b>Zde můžete vložit dlouhou zprávu</b>";
mm.IsBodyHtml = true;
mm.Priority = MailPriority.High;
Attachment att = new Attachment("myExcelResults.zip",
   MediaTypeNames.Application.Zip);
mm.Attachments.Add(att);
sc.Send(mm);
```

V takovémto případě bude před voláním metody `Send()`vytvořen objekt typu `Attachment` a pomocí metody `Add()` připojen k objektu typu `MailMessage`.

Využití tříd TCP

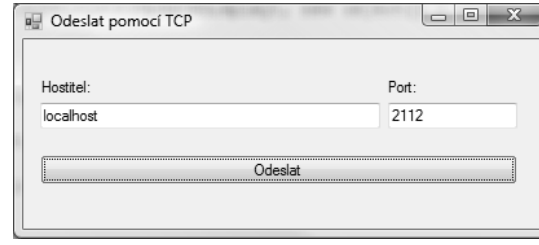
Třídy protokolu TCP (Transmission Control Protocol) nabízejí jednoduché metody pro připojování a odesílání dat mezi dvěma koncovými body. Koncový bod je kombinací adresy IP a čísla portu, přičemž existující protokoly mají čísla portů jasně definována. Například protokol HTTP používá port 80, zatímco protokol SMTP port 25. Úřad IANA (Internet Assigned Number Authority, `http://www.iana.org`) přiřazuje čísla portů obvyklým službám. Jestliže neimplementujete obvyklou službu, používejte čísla portů větší než 1 024.

Přenosy TCP tvoří hlavní část přenosů v Internetu. Protokol TCP je využíván velmi často, protože nabízí zaručené doručení obsahu, opravy chyb a ukládání dat do vyrovnávací paměti. Třída `TcpClient` zapouzdřuje připojení TCP a nabízí několik vlastností správy připojení, včetně ukládání dat do vyrovnávací paměti, definice velikosti vyrovnávací paměti nebo časových limitů. Pro čtení a zápis lze využít objekt typu `NetworkStream` vyžádaný metodou `GetStream()`.

Třída `TcpListener` naslouchá příchozím spojením TCP pomocí metody `Start()`. Po přijetí požadavku na připojení můžete použít metodu `AcceptSocket()`, která vrací soket pro komunikaci se vzdáleným počítačem, nebo metodu `AcceptTcpClient()` vracející pro komunikaci objekt vyšší úrovně typu `TcpClient`. Spolupráci tříd `TcpListener` a `TcpClient` nejsnáze pochopíte na příkladu.

Příklady TcpSend a TcpReceive

Abyste si vyzkoušeli, jak zmiňované třídy pracují, musíte vytvořit dvě aplikace. Na obrázku 41.11 si můžete prohlédnout první aplikaci TcpSend. Tato aplikace otevře spojení TCP se serverem a odešle mu vlastní zdrojový kód v jazyce C#.



Obrázek 41.11

Jedná se opět o standardní okenní aplikaci napsanou v jazyce C#. Formulář obsahuje dvě textová pole (`txtHost` a `txtPort`) pro zadání názvu DNS a čísla portu. Kromě textových polí obsahuje dále tlačítko (`btnSend`). Po klepnutí na toto tlačítko otevřete připojení. Nejprve však musíte v projektu zpřístupnit všechny nezbytné jmenné prostory:

```
using System;
using System.IO;
using System.Net.Sockets;
using System.Windows.Forms;
```

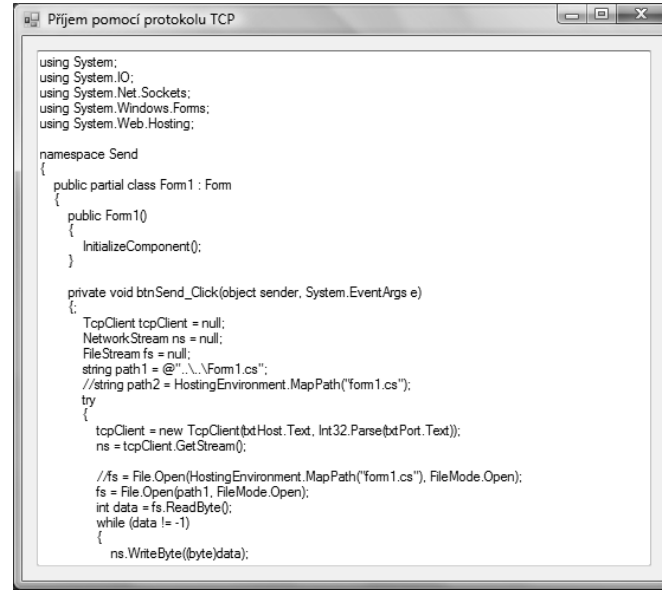
Následující kód obsahuje obslužnou metodu události klepnutí na tlačítko:

```
private void btnSend_Click(object sender, System.EventArgs e)
{
  TcpClient tcpClient = new TcpClient(txtHost.Text, Int32.Parse(txtPort.Text));
  NetworkStream ns = tcpClient.GetStream();
  FileStream fs = File.Open(Server.MapPath("form1.cs"), FileMode.Open);
  int data = fs.ReadByte();
  while(data!= -1)
  {
    ns.WriteByte((byte)data);
    data = fs.ReadByte();
  }
  fs.Close();
  ns.Close();
  tcpClient.Close();
}
```

V tomto příkladu vytvoříte instanci třídy `TcpClient` podle názvu DNS a čísla portu. Konstruktoru třídy `TcpClient` lze také předat instanci třídy `IPEnd Point`. Po získání instance třídy `NetworkStream` otevřete soubor zdrojového kódu a začnete načítat jeho bajty. Podobně jako v případě mnoha jiných binárních proudů musíte i nyní konec proudu ověřovat testováním návratové hodnoty metody `Read Byte()`. Hodnota -1 označuje konec proudu. Po načtení všech bajtů a po jejich odeslání do síťového proudu zavřete všechny otevřené soubory, připojení a proudy.

Na druhé straně připojení zobrazí aplikace `TcpReceive` po dokončení přenosu přijatý soubor (viz obrázek 41.12).

Tento formulář obsahuje jeden ovládací prvek `TextBox` nazvaný `txtDisplay`. V aplikaci `TcpReceive` budete naslouchat příchozím spojením pomocí třídy `TcpListener`. Abyste zabránili zatuhnutí rozhraní aplikace, budete čekat na spojení a číst z něj data v pracovním podprocesu na pozadí.



Obrázek 41.12

V projektu proto musíte rovněž použít jmenný prostor `System.Threading` a další jmenné prostory:

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Windows.Forms;
```

V konstruktoru formuláře spustíte pracovní vlákno (podproces, thread):

```
public Form1()
{
  InitializeComponent();
  Thread thread = new Thread(new ThreadStart(Listen));
  thread.Start();
}
```

Zbývá pak ještě tento důležitý kód:

```
public void Listen()
{
  IPAddress localAddr = IPAddress.Parse("127.0.0.1");
  Int32 port = 2112;
  TcpListener tcpListener = new TcpListener(localAddr, port);
  tcpListener.Start();
  TcpClient tcpClient = tcpListener.AcceptTcpClient();
  NetworkStream ns = tcpClient.GetStream();
  StreamReader sr = new StreamReader(ns);
  string result = sr.ReadToEnd();
  Invoke(new UpdateDisplayDelegate(UpdateDisplay), new object[] {result} );
  tcpClient.Close();
  tcpListener.Stop();
}
```

```
public void UpdateDisplay(string text)
{
  txtDisplay.Text= text;
}
protected delegate void UpdateDisplayDelegate(string text);
```

Podproces zavolá metodu `Listen()` a umožní spustit blokující volání `AcceptTcpClient()` bez zastavení uživatelského rozhraní. V kódu této aplikace jste napevno zadali adresu IP (127.0.0.1) a číslo portu (2112), takže i v aplikaci `TcpSend` musíte zadat stejné hodnoty.

Pro čtení otevřete nový proud pomocí objekt typu `TcpClient` vráceného metodou `AcceptTcpClient()`. Stejně jako v předchozím příkladu vytvoříte i zde objekt typu `StreamReader` pro snazší převod síťových dat na řetězec. Před uzavřením klienta a zastavením naslouchání aktualizujete textové pole zobrazené na formuláři. K textovému poli však nechcete přistupovat přímo z pracovního podprocesu spuštěného na pozadí, proto zavoláte metodu `Invoke ()` příslušného formuláře s delegátem a výsledek předáte jako první prvek v poli argumentů typu `object`. Metoda `Invoke()` zaručuje správné předání volání podprocesu, které vlastní systémové popisovače (handle) ovládacích prvků uživatelského rozhraní.

TCP versus UDP

Dalším protokolem, kterým se budeme zabývat v této podkapitole, je UDP (User Datagram Pro-tocol). Jedná se o jednoduchý protokol s nepříliš širokým rozsahem funkčností, ale zato s velmi malou režií. Vývojáři často používají tento protokol v aplikacích, kde je rychlost a výkon mnohem důležitější než spolehlivost – například při přenosu videa. Protokol TCP nabízí záruku správného doručení dat, korekci chyb a možnost opětovného přenosu v případech, kdy dojde ke ztrátě nebo porušení datového paketu. Protokol TCP ukládá odesílaná a přijímaná data do vyrovnávací paměti a také zaručuje příjem paketů ve správném pořadí, i když se během přenosu pomíchají. I přes dodatečnou režii je protokol TCP díky své vyšší spolehlivosti nejčastěji používaným protokolem na Internetu.

Třída UdpClient

Třída `UdpClient` nabízí v porovnání s třídou `TcpClient` menší a jednodušší rozhraní. Tato skutečnost odráží relativně jednoduchou povahu protokolu UDP. I když třídy TCP a UDP vnitřně pracují se sokety, třída `UdpCIient` neobsahuje žádnou metodu vracející síťový proud pro čtení a zápis. Místo toho přijímá členská funkce `Send()` pole bajtů, metoda `Receive()` je vrací. Protože je UDP nespojovým protokolem, můžete koncový bod komunikace určit až v argumentech metod `Send()` a `Receive()`, nikoli hned v konstruktoru nebo v metodě `Connect()`. Koncový bod lze také měnit při každém pozdějším volání metod `Send()` nebo `Receive()`.

V následující části kódu odešlete pomocí třídy `UdpClient` zprávu službě Echo, která je součástí služby Jednoduché služby TCP/IP. Server s takovou službou přijímá požadavky na připojení prostřednictvím protokolů TCP nebo UDP na portu číslo 7. Služba Echo jednoduše vrací zpět všechna přijatá data. Tato služba je velmi užitečná pro testování a diagnostiku, ale mnoho správců ji z bezpečnostních důvodů zakazuje:

```
using System;
using System.Text;
using System.Net;
```

```
using System.Net.Sockets;
namespace Wrox.ProCSharp.InternetAccess.UdpExample
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      UdpClient udpClient = new UdpClient();
      string sendMsg = "Ahoj, Echo";
      byte [] sendBytes = Encoding.ASCII.GetBytes(sendMsg);
      udpClient.Send(sendBytes, sendBytes.Length, "SomeEchoServer.net", 7);
      IPEndPoint endPoint = new IPEndPoint(0,0);
      byte [] rcvBytes = udpClient.Receive(ref endPoint);
      string rcvMessage = Encoding.ASCII.GetString(rcvBytes, 0, rcvBytes.Length);
      // měl by zobrazit text "Ahoj, Echo"
      Console.WriteLine(rcvMessage);
    }
  }
}
```

Při překladu řetězců na pole bajtů a obráceně využíváte kódování `Encoding.ASCII`. Všimněte si, že metodě `Receive()` předáváte objekt typu `IPEndPoint` odkazem. Vzhledem k tomu, že protokol UDP není spojový, mohlo by každé volání metody `Receive()` získávat data z jiného koncového bodu, takže metoda `Receive()` přebírá v tomto parametru adresu IP a číslo portu odesilatele.

Třídy `UdpClient` a `TcpClient` nabízejí abstraktní vrstvu nad nejnižší třídou `Socket`.

Třída Socket

Třída `Socket` poskytuje při síťovém programování nejvyšší úroveň řízení. Nejjednodušeji si ji vyzkoušíte, když s její pomocí přepíšete aplikaci `TcpReceive`. Aktualizovaná verze metody `Listen()` nyní vypadá takto:

```
public void Listen()
{
  Socket listener =
    new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
  listener.Bind(new IPEndPoint(IPAddress.Any, 2112));
  listener.Listen(0);
  Socket socket = listener.Accept();
  Stream netStream = new NetworkStream(socket);
  StreamReader reader = new StreamReader(netStream);
  string result = reader.ReadToEnd();
  Invoke(new UpdateDisplayDelegate(UpdateDisplay), new object[] {result} );
  socket.Close();
  listener.Close();
}
```

Při použití třídy `Socket` potřebujete pro dosažení stejného výsledku několik řádků navíc. Nejprve musíte konstruktoru této třídy předat schéma adresování IP pro proudový soket s protokolem TCP. Tyto argumenty představují pouze jednu z mnoha možných kombinací, které jsou pro třídu `Socket` k dispozici. Třída `TcpClient` konfiguruje tato nastavení za vás. Poté svážete naslouchací soket s portem a začnete naslouchat příchozím spojením. Po doručení příchozího požadavku můžete zavoláním metody `Accept()` vytvořit nový soket pro obsluhu připojení. Nakonec připojíte k soketu instanci třídy `StreamReader` a načtete příchozí data způsobem velmi podobným předchozímu.

Třída `Socket` rovněž obsahuje metody pro asynchronní připojování, přijímání a odesílání. Tyto metody můžete používat s delegáty pro zpětné volání stejným způsobem, jaký jste použili při asynchronním zobrazení stránky pomocí třídy `WebRequest`. Jestliže potřebujete pracovat s vnitřním nastavením soketu, budou se vám hodit metody `GetSocketOption()` a `SetSocketOption()`. Tyto metody umožňují prohlížet a konfigurovat volby týkající se časových limitů, životnosti a dalších možností nižší úrovně. Další podkapitola se věnuje dalšímu příkladu použití soketů.

Vytvoření serverové konzolové aplikace

Nahlédněme do třídy `Socket` trochu hlouběji. V dalším příkladě si vytvoříme konzolovou aplikaci, která se k příchozím požadavkům na soket chová jako server. Dále vytvoříme paralelně druhý příklad (další konzolovou aplikaci), která bude serverové konzolové aplikaci posílat zprávy.

První aplikace, kterou vytvoříme, je konzolová aplikace, jež se chová jako server. Tato aplikace na konkrétním portu PCP otevře soket a bude naslouchat všem příchozím zprávám. Kód této aplikace zde uvádíme v kompletní podobě:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
namespace SocketConsole
{
  class Program
  {
    static void Main()
    {
      Console.WriteLine("Začínám: Vytvářím objekt Socket");
      Socket listener =
        new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
      listener.Bind(new IPEndPoint(IPAddress.Any, 2112));
      listener.Listen(10);
      while (true)
      {
        Console.WriteLine("Čekám na spojení na portu 2112");
        Socket socket = listener.Accept();
        string receivedValue = string.Empty;
        while (true)
        {
          byte[] receivedBytes = new byte[1024];
```

```
          int numBytes = socket.Receive(receivedBytes);
          Console.WriteLine("Přijímám ...");
          receivedValue += Encoding. Default.GetString(receivedBytes, 0, numBytes);
          if (receivedValue.IndexOf("[FINAL]") > -1)
          {
            break;
          }
        }
        Console.WriteLine("Přijatá hodnota: {0}", receivedValue);
        string replyValue = "Zpráva byla úspěšně přijata.";
        byte[] replyMessage = Encoding.Default.GetBytes(replyValue);
        socket.Send(replyMessage);
        socket.Shutdown(SocketShutdown.Both);
        socket.Close();
      }
      listener.Close();
    }
  }
}
```

V tomto příkladu nastavíme soket pomocí třídy `Socket`. Vytvořený soket používá protokol TCP a je nastaven tak, aby zprávy ze všech adres IP přijímal na portu 2112. Hodnoty přijaté skrze otevřený soket se vypíší na konzolu. Aplikace bude přijímat příchozí data až do chvíle, kdy obdrží řetězec [FINAL]. Tento řetězec označuje konec příchozí zprávy a teprve nyní je možno zprávu přeložit.

Jakmile je celá zpráva přijata, zpráva s odpovědí se odešle témuž klientovi. Pak se soket pomocí metody `Close()` uzavře a konzolová aplikace bude až do příchodu další zprávy vyčkávat spuštěná.

Vytváříme klientskou aplikaci

Dalším krokem bude vytvořit klientskou aplikaci, která odešle zprávu předchozí konzolové aplikaci. Když dodržíte kritéria stanovená aplikací, bude moci klient konzolové aplikaci odeslat jakoukoliv zprávu. Prvním z takových pravidel je to, že konzolová aplikace naslouchá pouze konkrétnímu protokolu. V případě naší serverové aplikace se jedná o protokol TCP. Dalším pravidlem je, že serverová aplikace naslouchá pouze na konkrétním portu – zde je to port 2112. Poslední pravidlo říká, že ať už je odeslána jakákoliv zpráva, poslední bity musí mít podobu řetězce `[FINAL]`.

Následující klientská konzolová aplikace se všemi těmito pravidly řídí:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
namespace SocketConsoleClient
{
  class Program
  {
    static void Main()
```

```
    {
      byte[] receivedBytes = new byte[1024];
      IPHostEntry ipHost = Dns.Resolve("127.0.0.1");
      IPAddress ipAddress = ipHost.AddressList[0];
      IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2112);
      Console.WriteLine("Začínám: Vytvářím soket");
      Socket sender =
        new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
      sender.Connect(ipEndPoint);
      Console.WriteLine("Úspěšně spojeno s {0}", sender.RemoteEndPoint);
      string sendingMessage = "Ahoj, zkouška soketu";
      Console.WriteLine("Vytvářím zprávu: Ahoj, zkouška soketu");
      byte[] forwardMessage = Encoding. Default.GetBytes(sendingMessage + "[FINAL]");
      sender.Send(forwardMessage);
      int totalBytesReceived = sender.Receive(receivedBytes);
      Console.WriteLine("Zpráva ze serveru: {0}",
         Encoding. Default.GetString(receivedBytes, 0, totalBytesReceived));
      sender.Shutdown(SocketShutdown.Both);
      sender.Close();
      Console.ReadLine();
    }
  }
}
```

V tomto příkladě se pomocí adresy IP localhost vytvoří objekt `IPEndPoint` a použije se port 2112 tak, jak to vyžaduje serverová konzolová aplikace. V našem případě se vytvoří socket a vyvolá se metoda `Connect()`. Jakmile se soket otevře a spojí se s instancí soketu serverové konzolové aplikace, odešle se serverové aplikaci prostřednictvím metody `Send()` textový řetězec. A protože serverová aplikace zprávu vrátí (a umístí ho do bajtového pole), použije se k získání této zprávy metoda `Receive()`. Bajtové pole se pak konvertuje na řetězec a ten se před zavřením soketu zobrazí v konzolové aplikaci.

Když aplikaci spustíte, uvidíte totéž, co na obrázku 41.13.

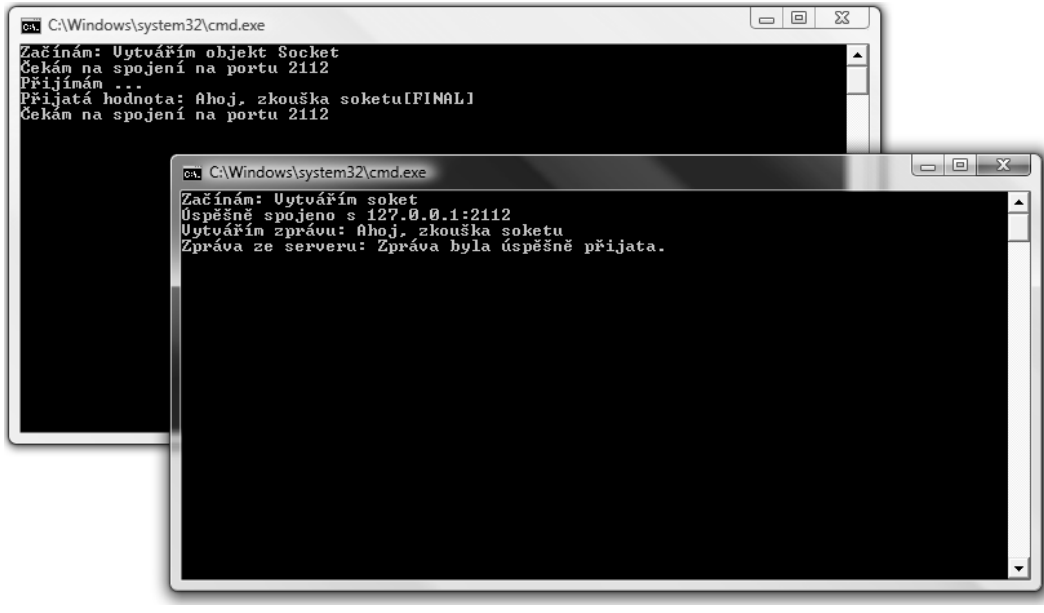
Když si na obrázku obě konzolové aplikace prohlédnete, všimnete si, že serverová aplikace se spustí a čeká na příchozí zprávy. Ty jsou odeslány z klientské aplikace. Odeslaný řetězec se poté zobrazí v serverové aplikaci. Ta bude čekat i na další zprávy, a to i po přijetí a zobrazení první zprávy. Můžete si to vyzkoušet sami, zavřete a znovu spusťte klientskou aplikaci. Uvidíte, že serverová aplikace zobrazí přijatou zprávu znovu.

Shrnutí

V této kapitole jsme se věnovali třídám .NET ze jmenného prostoru `System.Net`, který je určen pro síťovou komunikaci. Seznámili jste se s několika základními třídami, které otevírají klientská spojení na síti a v Internetu, odesílají požadavky a přijímají odpovědi (jejich nejčastější oblastí uplatnění je příjem stránek HTML). Také jste se naučili využívat výhody nového ovládacího prvku

`WebBrowser` v prostředí .NET verze 3.5 při snadném nabízení funkčnosti prohlížeče Internet Explorer ve vašich aplikacích.

Z dosavadních zkušeností je zřejmé, že při programování pomocí tříd definovaných ve jmenném prostoru `System.Net` byste měli vždy pracovat s co možná nejobecnějšími třídami. Například použitím třídy `TcpClient` namísto třídy `Socket` ušetříte svůj kód množství podrobností z nižší úrovně soketu. Krok o stupeň výše umožňuje využít třídu `WebRequest` pro práci s architekturou vyměnitelných protokolů platformy .NET Framework. Váš kód tak bude připravený na využití výhod nových protokolů aplikační úrovně, jakmile společnost Microsoft nebo jiní dodavatelé zavedou novou funkčnost.



Obrázek 41.13

Nakonec jste se naučili využívat asynchronní schopnosti tříd síťové komunikace, které formulářovým aplikacím Windows poskytují možnost nabízet profesionální vzhled stále reagujícího uživatelského rozhraní.

A nyní je načase seznámit se s technologií Windows Communication Foundation.