

## ELSŐ FEJEZET

# A webes alkalmazások alapjai

Ebből a fejezetből a következőket lehet megtanulni:

- HTTP-kérések értelmezése.
- A .NET-keretrendszer segítségével HTTP-kérések küldése böngésző nélkül.
- A HTML-kód értelmezése.
- Az IIS használata.
- Dinamikus webtartalom előállítása még az ASP.NET nélkül.

A fejezet a webes alkalmazások készítésének alapjait ismerteti. Az asztali alkalmazások fejlesztésétől eltérően, ahol az alkalmazás részei lokálisan rendelkezésre állnak (a felhasználó merevlemez-meghajtóján), a webes alkalmazások fejlesztése során az egyes szoftverösszetevőknek kapcsolat nélküli protokoll segítségével, elosztott hálózatban kell működniük. Az ASP.NET mögöttes technológiái már jó ideje rendelkezésre állnak. Az ASP.NET kissé rejtve, de ugyanakkor megközelíthető módon használja ezeket a technológiákat.

Bár az ASP.NET nagymértékben megkönnyíti a webes alkalmazások fejlesztését, az ASP.NET-alkalmazás fejlesztése során szilárd elméleti alapokra van szükségünk a rendszer működését illetően. Ezt remekül példázza az az eset, amikor egy eltevedt HTTP-kérést követünk nyomon, vagy azt próbáljuk meg kideríteni, hogy az oldal miért rossz fontkészlettel jelenik meg az ügyfél böngészőjében. Másik példa a weblapok egyedi vezérlőelemeinek készítése. Az egyedi vezérlőelemek esetén a megjelenítés kódját igen gyakran kézzel kell megírni. Azaz, meg kell győződnünk arról, hogy a vezérlőelem a HTML-címkéket a megfelelő sorrendben küldi. Ehhez pedig HTML-ismeretekre van szükség. A fejezetben az ASP.NET-tel való munka három elengedhetetlenül szükséges területével ismerkedhetünk meg:

- a HTTP-kérések működésével,
- a HTML működésével,
- a HTTP-kérések kezelésével Microsoft-platformon (IIS).

A három mögöttes ASP.NET-technológia ismerete biztosítja, hogy a rendszer többi részének működését is megértsük. Az ASP.NET tanulmányozásakor a problémás kérdésekre is választ kapunk.

## HTTP-kérések

A webböngészők a HTTP (**HyperText Transfer Protocol**, hiperszöveg átviteli protokoll) kommunikációs mechanizmus segítségével szólítják meg a webhelyeket. A ma ismert web kutatási projektnek indult a CERN-nél, Svájcban. Akkoriban a hiperszöveg fogalma – önkényesen egymáshoz kapcsolt dokumentumok – egyre népszerűbbé vált. Az alkalmazások, mint például az Apple Computer Inc. Hypercard programja, bemutatták a nagyközönségnek a hiperszöveges alkalmazásokat. Ha a dokumentumokat hálózaton keresztül össze lehet kapcsolni, az forradalmasíthatja az információk közzétételét. Ez hívta életre a HTTP-t, amely a TCP/IP felett az alkalmazási réteget alkotja.

Eredeti formájában a HTTP a hiperszöveg-dokumentumok átvitelére szolgált. Vagyis a HTTP célja eredetileg a dokumentumok összekapcsolása volt tekintet nélkül bármire, legyenek azok a webes felhasználói felületek, amelyek összekapcsolják a mai modern webhelyeket. A HTTP korai verziói egyetlen GET-kérést támogattak, amely visszaadta a megnevezett erőforrást. A későbbiekben a kiszolgáló feladata lett, hogy a fájl szövegfolyamként továbbítsa. Miután a válasz megérkezett az ügyfél böngészőjére, a kapcsolat lezárult. A HTTP első verziói kizárólag a szövegfolyamok átvitelét támogatták, semmilyen más adatátvitel nem volt lehetséges.

A HTTP első formális specifikációja az 1.0 verzióban, az 1990-es évek közepén jelent meg. A HTTP 1.0 az egyszerű szövegátviteli protokollon túl összetettebb üzenetküldést tett lehetővé. A HTTP különböző (a MIME [**Multipurpose Internet Mail Extensions**, többcélú internetes levélkiterjesztés] által meghatározott) médiaikat támogatott. A HTTP aktuális verziója az 1.1-es verzió.

Kapcsolati protokollként a HTTP néhány alapvető parancsra épül. Az ASP.NET-alkalmazások fejlesztése során a GET, a HEAD és a POST parancsokkal találkozunk.

A GET-parancs visszaadja a kérés URI-je (**Uniform Resource Identifier**, egy-  
sleges erőforrásazonosító) által meghatározott információt. A HEAD-parancs kizárólag a kérés URI-je által meghatározott fejrészadatokat adja vissza (más szóval, az üzenettörzset nem keresi meg). A POST-módszer segítségével a kiszolgálóra küldött kérésnek bizonyos „mellékhatásai” lehetnek. A weboldallal a kapcsolatot a GET-parancs segítségével vehetjük fel, és a további kommunikációt POST-parancsok révén bonyolíthatjuk.

## HTTP-kérések a böngészőből

Példaként tekintünk meg azt a kérést, amely a böngésző `helloworld.htm` localhost-on futó erőforrást az `aspnet2sbs` virtuális könyvtárból nyeri ki. Az 1.1. listában a kiszolgálóhoz küldött kérés szövegét látjuk. Ha szeretnénk megtekinteni az elküldött adatokat is, erre a célra rendelkezésünkre állnak TCP monitorozó eszközök. Ezen kívül a TELNET segítségével is küldhetünk GET kéréseket a kiszolgálóhoz. Monitorozó eszközöket az interneten is kereshetünk. A legtöbb eszköz használata meglepően egyszerű.

Ha TELNET segítségével szeretnénk a kiszolgálónak HTTP-kérést küldeni, a következő lépéseket kell végrehajtanunk:

1. Nyissuk meg a Visual Studio parancssort, és csatlakozzunk számítógépünkhöz a 80-as porton keresztül!
2. A parancssorba írjuk be a következő parancsot

```
C:\>TELNET localhost 80
```

3. Miután a telnetügyfél kapcsolódik, írjuk be a következő GET-parancsot (feltételezzük, hogy számítógépünkön létezik az `aspnet2sbs` virtuális könyvtár, és tartalmazza a `HelloWorld.HTM` fájlt):

```
C:/> GET //aspnet2sbs/helloworld.htm
```

4. A parancssorban a fájl tartalmának kell megjelennie.

```
GET /aspnet2sbs/helloworld.htm HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-powerpoint, application/vnd.ms-excel,
application/msword, application/x-shockwave-flash, */*
Accept-Language: en-us Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; windows NT 5.1; SV1;
.NET CLR 1.1.4322; .NET CLR 2.0.50215)
Host: localhost:80
Connection: keep-alive
```

### 1.1. lista

Amikor a böngésző HTTP-kérést szeretne küldeni, a böngészőnek kell összeállítania a HTTP-kérést, amely egyéb információk mellett az URI-t is magában foglalja. A kérés fejrészadatai a böngésző működési környezetével kapcsolatos részleteket és a kiszolgáló számára hasznos egyéb információkat is tartalmaznak. Amikor a kiszolgáló megkapja a kérést, a kért erőforrást szövegfolyamként küldi vissza. A böngésző elemzi a folyamatot, és formázza a tartalmat.

Az 1.2. listában a kiszolgáló válaszát találjuk, amelyet a HelloWorld.htm fájl kérésére küld. Általában a fejrészadatokat nem látjuk, amikor az erőforrást a böngészőben megtekintjük. Egy jó TCP nyomkövető segédprogram megmutatja ezeket az információkat is. A későbbiekben megvizsgáljuk az ASP.NET nyomkövetési lehetőségeit, és a fejrészadatok ott is láthatók lesznek.

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.1
X-Powered-By: ASP.NET
Date: wed, 01 Jun 2005 23:44:04 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Sun, 22 May 2005 21:54:20 GMT
ETag: "04e9ace185fc51:bb6"
Content-Length: 130
```

```
<html>
  <body>
    <h1> Hello world </h1>
    Nothing really showing here yet, except some HTML...
  </body>
</html>
```

### 1.2. lista

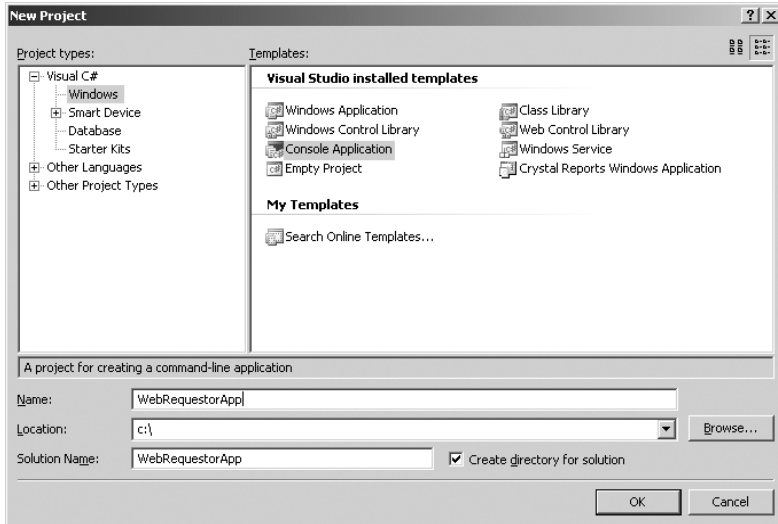
Az első sor a protokollt és a visszatérési kódot jelzi. A válasz további része (az első `<html>` címkéig) a kérés dátumát, a fájl utolsó módosításának dátumát, és a további tartalomtípus részleteit tartalmazza. Ezek az információk a későbbiekben lesznek hasznosak, amikor a lapok gyorsítótárzását és a böngésző tulajdonságainak kiderítését vizsgáljuk. A válaszfejléc-adatokat követő tartalom a kiszolgáló által visszaküldött tényleges HTML-fájl.

## HTTP-kérések küldése böngésző nélkül

Azon kívül, hogy a webes alkalmazások fordításának keretrendszere, a .NET fejlesztői környezet a nyers HTTP-kérések küldéséhez szükséges osztályokat is magában foglalja. A `WebRequest` osztály tagja a `GetResponse`, amely az URL által meghatározott címre küldi a kérést. Megtudhatjuk, hogy hogyan is kell a webkiszolgálónak böngésző nélkül közvetlen kéréseket küldeni, ha lefordítjuk és futtatjuk az alábbi rövid programot, amely a *Microsoft.com* kezdőlapot adja vissza.

## Egyszerű HTTP kérelmező készítése

1. Indítsuk el a Visual Studio.NET-et! A főmenüben válasszuk a **New | Project** menüpontot! A New Project párbeszédpanelben válasszuk a Console alkalmazást, és az alábbiaknak megfelelően legyen az új projekt neve **WebRequestorApp**.



A Visual Studio egy üres parancssori programot generál.

2. Írjuk be a kódot, amellyel webes kérést küldhetünk a programnak! A Visual Studio a Console alkalmazás belépési pontjait a Program.cs fájlba helyezi. (Ez a fájl az a kód, amely alapértelmezés szerint a kódatlakban megjelenik.) A webes kérés küldéséhez szükséges kód az alábbi kódsorokban félkövér betűkkel látható:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.IO;

namespace webRequestorApp
{
    class Program
    {
        static void Main(string[] args)
        {
            WebRequest req =
                WebRequest.Create
                    ("http://www.microsoft.com");
            WebResponse resp = req.GetResponse();
        }
    }
}
```

```

        StreamReader reader =
            new StreamReader(resp.GetResponseStream(),
                Encoding.ASCII);
        Console.WriteLine(reader.ReadToEnd());
    }
}
}

```

3. Futtassuk az alkalmazást! Az alkalmazást a főmenü **Debug | Start Without Debugging** menüpontjának választásával is futtathatjuk. A Visual Studio elindítja a parancssort, és futtatja a programot. Néhány másodperc elteltével a képernyő a következő HTML jelenik meg:

```

C:\WINDOWS\system32\cmd.exe
</head>
<body>
<script type="text/javascript">
<!--
var isW; isW=(document&&document.body.clientWidth&&document.body.clientWidth)>895
&&document.getElementById;
</-->
</script>
<a href="http://www.microsoft.com/default.aspx#cArea" class="hide">Click here to
jump to main page content</a>
<div id="Page" class="page">
<table cellpadding="0" width="100%"><tr><td colspan="2">
<table cellpadding="0" width="100%" style="height: 22px"><tr>
<td width="50%" style="filter:progid:DXImageTransform.Microsoft.Gradient(startCo
lorStr=#4B92D9, endColorStr=#CEDFF6, gradientType='1')"></td>
<td width="50%" style="filter:progid:DXImageTransform.Microsoft.Gradient(startCo
lorStr=#CEDFF6, endColorStr=#1E77D3, gradientType='1')"></td>
</tr></table></td>
<td id="msuiGlobalToolBar" height="22" nowrap align="left">
<table cellpadding="0"><tr>
<td class="gt0" nowrap onmouseover="this.className='gt1'" onmouseout="this.class
Name='gt0'"><p><a href="http://www.microsoft.com/worldwide&amp;&amp;HL=Microsof
tz2bworldwide&amp;CM=Masthead&amp;CE=geotargeting">Microsoft Worldwide</a></p></
td>
<td class="qtsep">|</td>

```

Természetesen ez a HTML emberi fogyasztásra alkalmatlan. A böngésző feladata, hogy ez észlelhetővé tegye. Azonban ez a példa nem mutatja be a webes kérések küldésének alapjait, és világosan látjuk, hogy a válaszban mi jön vissza.

Ebben az esetben a kiszolgálóhoz küldött kérés sokkal kisebb. A `WebRequest.GetResponse` a kérésben nem tartalmaz annyi információt, kizárólag a szükséges GET-parancsot, amelyet az URI, a hosztinformáció és a kapcsolattípus követ:

```

GET /aspnet2sbs/helloworld.htm HTTP/1.1
Host: localhost:80
Connection: Keep-Alive

```

A legtöbb böngésző alapvető feladata, hogy (1) becsomagolja a kérést és azt elküldje az URI által meghatározott kiszolgálónak, majd (2) megkapja a kiszolgálótól érkező választ, és azt érthető formába önti. A válasz rendszerint HTML-címkékkel jelölt szövegfolyamként érkezik.

# A HTML-nyelv

Az ASP.NET vizsgálata során elég sokszor találkozunk majd a HTML-lel. A legtöbb esetben a HTML-kódot ASP.NET kiszolgálóoldali vezérlőelemek generálják. Fontos, hogy megértsük a HTML-t, mert előfordulhat, hogy a semmiből kell kiszolgálóoldali vezérlőelemet írunk, és néha az ASP.NET-alkalmazásunk kimenetében kell hibát keresnünk.

A legtöbb HTTP-kérés eredményeként szövegfolyam érkezik vissza a kérelmezőhöz. Hallgatólagos beleegyezés szerint a HTML a dokumentumok formázásának nyelve, és a legtöbb böngésző megérti a HTML-kódot.

A HTML első használható verziója a 2.0 volt. A 3.2 verzió rengeteg új szolgáltatást foglalt magában – mint például a táblákat, az alkalmazásokat, a képeket körbeölelő szövegfolyamot, a felsőindexeket és tömbindexeket –, és biztosította a visszafelé kompatibilitást a létező HTML 2.0 szabvánnyal.

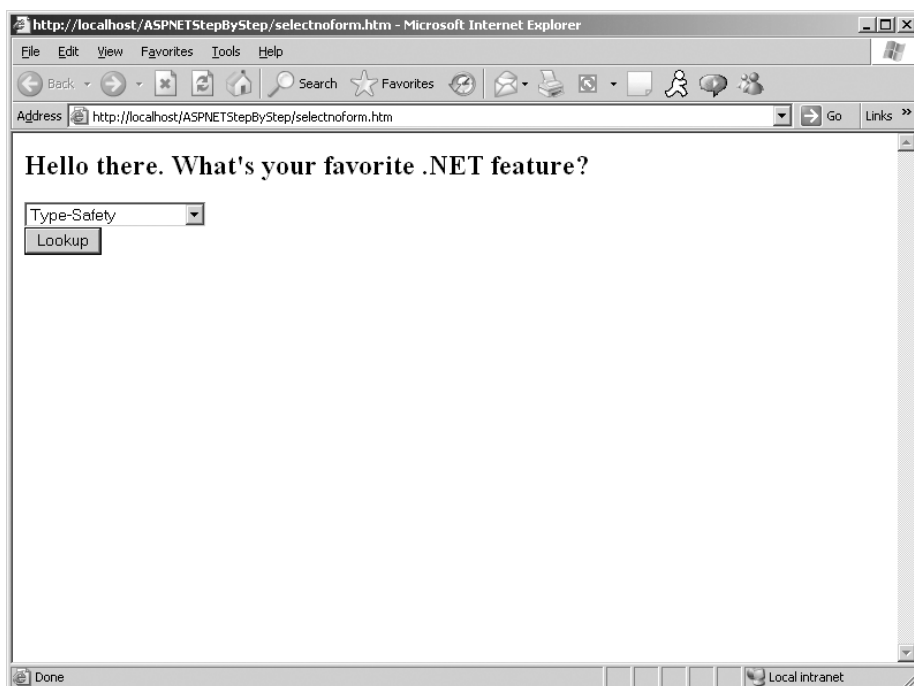
Végso sorban, alkalmas böngésző és jól felépített HTML segítségével a felhasználói felület fejlesztési technológiáját tudhattuk magunkénak. Mivel a HTML-t a különböző platformokon futó böngészők is megértették, az ajtó nyitva állt egy világszerte interaktív informatikai platform megvalósításához. A HTML érett verziója mellett a másik kulcsfontosságú tényező, amely mindezt lehetővé tette, a kiszolgálóknak az a képessége volt, amellyel futásidőben a meghatározott felhasználói kérésekhez igazították a kimenetet.

Az 1.3. listában látható HTML-folyam HTML-oldalt jelenít meg, amely egy gombot és egy opciókat tartalmazó legördülő mezőt tartalmaz. (A fájl a SelectNoForm.htm fájl, amely a fejezet példagyűjteményében is megtalálható.)

```
<html>
  <body>
    <h2>Hello there. What's your favorite .NET feature?</h2>
    <select name='Feature'>
      <option> Type-Safety</option>
      <option> Garbage collection</option>
      <option> Multiple syntaxes</option>
      <option> Code Access Security</option>
      <option> Simpler threading</option>
      <option> Versioning purgatory</option>
    </select>
    <br>
    <input type=submit name='Lookup' value='Lookup'></input>
    <br>
  </body>
</html>
```

### 1.3. lista

Az 1.1. ábrán a böngészőben megjelenített oldal képét látjuk.



1.1. ábra *Kiválasztási címkét bemutató egyszerű HTML-oldal (az ábrán Windows legördülő mező formájában jelenik meg) és a Submission gomb*



**Megjegyzés:** A következő fejezetekben valóban egy HTML-fájlhoz navigálunk. Addig a pontig kissé bonyolult eljutni, ezért egyelőre legyen elég annyi, hogy elhisszük, a HTML valóban így jelenik meg.

Ez egy statikus oldal. Bár az oldalon legördülő mezőt és egy gombot találunk, ezek az elemek nem tesznek semmi érdemlegeset. Lenyithatjuk a legördülő mezőt, és a mezővel a böngészőben dolgozhatunk. Megnyomhatjuk a gombot, de bármilyen műveletet lokálisan hajt végre a rendszer. Ennek az az oka, hogy a túloldali kiszolgálónak támogatnia kell a dinamikus tartalmat.

## A dinamikus tartalom

Az első webhelyek elsősorban statikus HTML-lapok segítségével készültek. Valahol egy oldalhoz navigálhattunk, és elolvashattuk a webhelyen tengődő HTML-dokumentumokat. Míg akkoriban ez lenyűgöző eredmény volt, a HTML azóta rengeteget fejlődött, és az egyszerű szövegformázásnál már jóval többre képes.

Például a HTML olyan címkéket tartalmaz, mint például a `<select></select>` címke, amelyet a legtöbb böngésző legördülő mezőként értelmez. Az `<input></input>` címke különböző attribútumainak révén a böngészőkben szövegmezők és gombok jelennek meg.

## HTML-űrlapok

A HTML `<form></form>` címkéje értesíti a böngészőt arról, hogy a HTML egy része vezérlőelemeket képviselő címkéket tartalmaz. A címke segítségével meghatározhatjuk, hogy a webes dokumentum hogyan kezeli a végfelhasználótól érkező bemenő adatokat (tehát nem csak a kimenetet befolyásolja).

A `<form>` címke fogja közre a vezérlőelemeket meghatározó címkék halmazát. Az 1.4. lista egy szolgáltatás választó lapot mutat be hozzáadott űrlapcímkével (a fájl a `SelectFeature2.htm` fájl, amelyet megtalálunk a kapcsolódó példák között is):

```
<html>
  <body>
    <form action="http://localhost/HttpHandlers/selectfeature.htm"
          method="get">
      <h2>Hello there. What's your favorite .NET feature?</h2>
      <select name='Feature'>
        <option> Type-Safety</option>
        <option> Garbage collection</option>
        <option> Multiple syntaxes</option>
        <option> Code Access Security</option>
        <option> Simpler threading</option>
        <option> Versioning purgatory</option>
      </select>
      <br>
      <input type=submit name='Lookup' value='Lookup'></input>
      <br>
    </form>
  </body>
</html>
```

### 1.4. lista

Az űrlapcímké több attribútummal rendelkezik, amelyek segítségével szabályozhatjuk az oldal viselkedését. A fenti példában vegyük észre, hogy a `<form>` címke beállítja az `ACTION` attribútumot, amely visszamutat arra a kiszolgálóra, amely megkapja az űrlap tartalmát. Ennek hiányában a rendszer az aktuális dokumentum URL-jét használja.

Az 1.4. listában található másik attribútum a `method` attribútum. A `method` attribútum meghatározza az űrlap küldése során alkalmazott HTTP-metódust. A példa a GET metódust alkalmazza, mivel a kiszolgáló szoftvere még nem érti a POST metódust. A GET metódus hatására a rendszer az űrlap tartalmát az URL-hez fűzi. A POST metódus használatával az űrlap tartalmát a rendszer az adattörzsben küldi el a kiszolgálónak.

Ha az űrlapcímkét hozzáadjuk a dokumentum törzséhez, már félúton vagyunk egy működő HTTP-alkalmazás létrehozásához. Most egy kicsivel több támogatásra lesz szükségünk a kiszolgáló oldalon. Amikor a Lookup gombra kattintunk, a böngésző újabb kört kezdeményez a kiszolgálóhoz (bár csak egy HTTP GET-parancsot hajt végre, amely segítségével újra behívja a dokumentumot).

Ezen a ponton egy normál HTTP GET-parancs kizárólag a dokumentumot adná vissza. Igazán interaktív környezetben a másik oldalon, ahogy a kérések jönnek-mennek a böngésző és a kiszolgáló között, a kiszolgálónak módosítania kell a tartalmat.

Képzeld el például, hogy a felhasználó GET-parancsokkal kéri az erőforrást, kiválaszt néhány szolgáltatást a legördülő mezőben, majd a Submit gombra kattint. Az interaktív alkalmazás működéséhez a böngészőnek a kiszolgálóhoz egy újabb kéréssel újabb kört kell tennie. A kiszolgáló megvizsgálja a böngészőtől érkező kérést, és eldönti, hogy mihez kezdjen vele.

## A CGI-interfész

Az első webkiszolgálók, amelyek támogatták a „dinamikus webtartalmat” a CGI-t (**Common Gateway Interface**, közös átjáró interfész) hívták segítségül. A CGI volt a webkiszolgálók létesítésével kapcsolatos egyik legelső szabvány. A CGI-programok valósídejű programok, és az alkalmazás állapotától, valamint a bejövő kéréstől függően módosítják a kimenetet. A CGI-t futtató webkiszolgálóhoz beérkező kérések a program külön példányait futtatják. Az alkalmazás bármilyen művelet végrehajthat, például adatbázis adatait keresheti, hitelkártyaszámokat fogadhat, és formázott információt továbbíthat.

## A Microsoft-platform mint webkiszolgáló

Microsoft-platformon rendkívül költséges lenne minden kérés számára új folyamatot indítani (à la CGI). A Microsoft ezt a problémát a 80-as portot figyelő egyetlen démonfolyamattal oldotta meg, amely a tartalom változásainak megfelelően betölti a külön kérések kezeléséhez szükséges DLL-eket. A Microsoft webes platformjának alapja az IIS (**Internet Information Services**, internet információs szolgáltatások).

## Az Internet Information Services

Alapvetően az összes webes alkalmazáskörnyezet működése hasonló. A hardver-/szoftverplatformtól függetlenül valamilyen szoftverre van szükség ahhoz, hogy a bejövő HTTP-kéréseket a kiszolgálón a 80-as porton figyeljük. Amikor egy kérés érkezik, a kiszolgáló dolga, hogy értelmes módon válaszoljon a kérésre. Microsoft-platformon az IIS figyeli a beérkező HTTP-kéréseket a 80-as porton, ez a HTTP-kérések szokásos bejövő portja. Az internetes kiszolgálók más portokat is használnak. Például a HTTPS (**Secure HyperText Transfer Protocol**, biztonságos hiperszöveg átviteli protokoll) a 443-as portot használja. Azonban most összpontosítsunk a 80-as port szokásos internetes forgalmára!

Amikor a böngésző Microsoft-platformon futó kiszolgálónak küld kérést, az IIS fogadja a kérést, és megkeresi az URL segítségével azonosítható erőforrást. Az IIS saját könyvtár térközét kezelhető darabokra, úgynevezett virtuális könyvtárakra bontja. Képzeljük el, hogy az egyik felhasználó az alábbi URL segítségével szeretne a kiszolgáló valamely erőforrásához hozzáférni:

*<http://www.aspnetstepbystep.com/examples/showfeatures.htm>*

Az „aspnetstepbystep” tartomány kitalált, csak a példa kedvéért használjuk. Ha a tartományban létezne ezzel a névvel regisztrált kiszolgáló, az URL a teljes erőforrást azonosítaná. Az URL-en belül, a *http://www.aspnetstepbystep.com* rész azonosítja a kiszolgálót, és a rendszer a kérést útválasztók labirintusán keresztül a megfelelő helyre irányítja. Míhelyest a kérés a kiszolgálóra érkezik, a kiszolgáló megkeresi a *showfeatures.htm* erőforrást az *examples* nevű könyvtártípusú entitásban. Ha a kiszolgálón IIS fut, az *examples* egy virtuális könyvtárra hivatkozik.

Az IIS a munka térközt több *virtuális könyvtárra* bontja. A virtuális könyvtárak általában egyetlen alkalmazásra hivatkoznak. Ily módon az IIS több alkalmazást is ki tud szolgálni. A virtuális könyvtárak különböző konfigurációs beállításokat foglalnak magukban, többek között biztonsági beállításokat, hibakezelési átirányításokat és alkalmazás-elszigetelési beállításokat. A konfigurációbeli beállítások a különböző fájl kiterjesztések és az ISAPI DLL-ek leképezéseit is magukban foglalják.

## Az IIS alkalmazásprogramozói interfészének függvénytárai

Microsoft-platfomon a folyamatterület létrehozása (rendszererőforrások és órajelciklusok tekintetében) igen költséges feladat. Képzeljük el, hogy olyan kiszolgáltót próbálunk üzembe helyezni, amely minden kérésre külön program indításával válaszol. Szegény kiszolgáltól igen hamar padlóra kerülne, és e-kereskedelmi webhelyünk nem termelne profitot.

A Microsoft architektúra DLL-ek segítségével válaszol a kérésekre. A DLL-ek betöltése viszonylag olcsó, és a kódok a DLL-ben igen gyorsan futtathatóak. A webes kéréseket kezelő DLL-ek az *ISAPI DLL-ek* (**Internet Services Application Programming Interface**, internetszolgáltatások alkalmazásprogramozási interfésze).

Nem merülünk el az ISAPI DLL-ek belső működésének részleteiben, de futó pillantást vetünk a DLL-ek architektúrájára, és megvizsgáljuk, hogy milyen kapcsolatban áll az ASP.NET-tel.

A rendes HTTP-kéréseket kezelő ISAPI DLL-ek a *HttpExtensionProc* belépési pontot definiálják. Bár az ISAPI-kiterjesztés DLL-ek az *HttpExtentsionProc* mellett további belépési pontokat is definiálnak, messze ez a legfontosabb metódus az ISAPI DLL-ekben. Az ISAPI-kiterjesztés DLL-ek kapcsán a lényeg, hogy a DLL-eknek ez az egyedüli funkciója, amikor a HTTP-kérésekre válaszolnak. Azonban a különböző DLL-ek különbözőképpen válaszolnak.

- A *HttpExtensionProc* metódus egyetlen paraméterrel rendelkezik, egy `EXTENSION_CONTROL_BLOCK` struktúrával. Az `EXTENSION_CONTROL_BLOCK` a kérés teljes kontextusát magában foglalja. Nem kell a teljes struktúrát látnunk. Látni fogjuk ennek kezelt megfelelőjét az ASP.NET-ben, amikor majd a *HttpContext* osztállyal ismerkedünk.

A kérés fogadásakor az IIS az `EXTENSION_CONTROL_BLOCK` struktúrába csomagolja az információkat. Ezt követően az IIS a struktúrát a *HttpExtensionProc* belépési ponton keresztül átadja az ISAPI DLL-nek. Az ISAPI-kiterjesztés DLL feladata a bejövő kérés elemzése, és a DLL felelőssége, hogy valamit kezdjen a kéréssel. Az ISAPI-kiterjesztés DLL tetszése szerint bármit tehet a kéréssel. Például az ügyfél küldhet olyan kérést, amely a lekérdezősztringben paramétereket tartalmaz (lehet, hogy ügyfél keresést vagy valami hasonló tevékenységet illetően). Az ISAPI-kiterjesztés DLL a lekérdezősztring-paraméterek segítségével hozza létre a webhelyspecifikus adatbázis-lekérdezést. Ha kereskedelmi webhelyről van szó, lehet, hogy az adatbázis-lekérdezés az aktuális raktárkészletet érinti. A kérés feldolgozása után az ISAPI DLL az eredményeket visszaömlészi az ügyfélnek.

Ha rendelkezünk némi klasszikus ASP tapasztalattal, akkor a struktúra nagy része ismerős lehet számunkra. Például az ASP belső *Response* objektumán keresztül meghívott *Write* végső sorban azt a metódust hajtja végre, amelyre a *WriteClient* mutat.

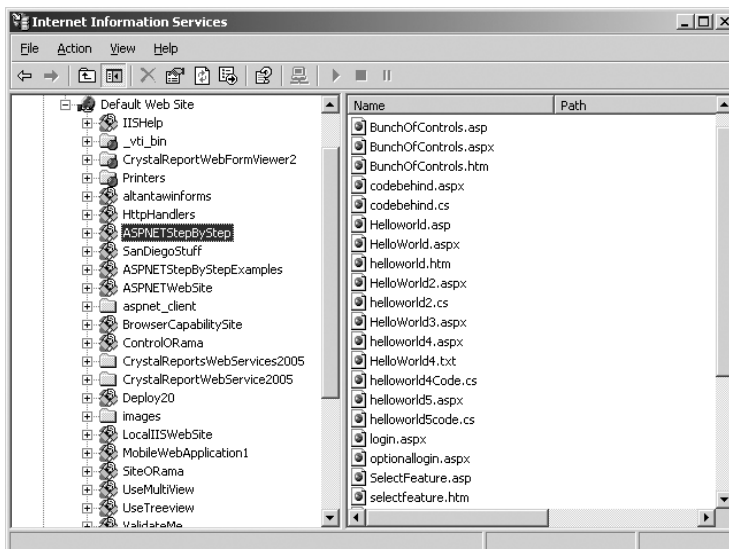
Rövid felfedezőutat tettünk az ISAPI DLL-ek belső felépítésében. Most azt vizsgáljuk meg, hogy ezek a DLL-ek hogyan illenek bele az IIS-be.

## Az Internet Information Services

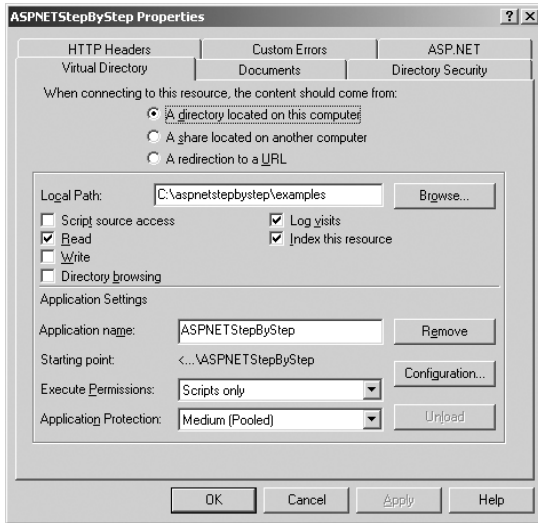
Az IIS felhasználói felülete a Control Panelen keresztül érhető el. Tegyük egy rövid kitérőt, és nézzük meg az IIS 5.1 adminisztrációját! Fontos, hogy az IIS valahol rendelkezésünkre álljon, mivel az ASP.NET az IIS-re támaszkodik a webes kérések kiszolgálása során. Az IIS 5.x és a 6.0 annyiban hasonlít, hogy mindkét verzió virtuális könyvtárakra bontja a kiszolgáló alkalmazás térközét. Az IIS 6.0 rengeteg új szolgáltatást foglal magában, például az alkalmazások elszigetelését és újrafelhasználását. De ezekkel a tulajdonságokkal most nem foglalkozunk.

### Az IIS futtatása

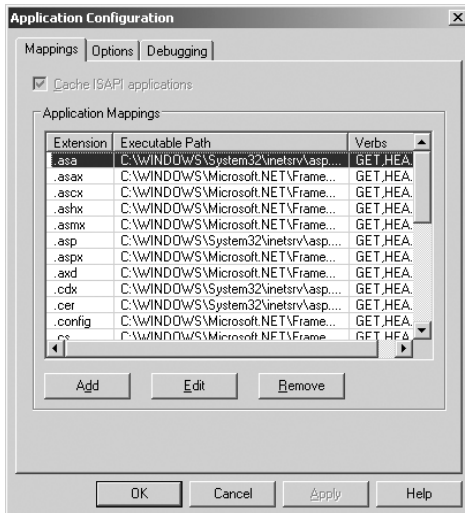
1. **Futtassuk az IIS-t!** Az IIS futtatásához először válasszuk az **Administrative Tools** menüpontot. A Windows XP Professional operációs rendszeren ezt a Control Panelen keresztül tehetjük meg. Indítsuk el az IIS-t, és a képernyőn megjelenik az IIS-beli felhasználói felület:



2. **Tekintsük meg egy adott virtuális könyvtár beállításait!** A képernyő bal oldalán kibontható fa mutatja a számítógépen az IIS-en keresztül rendelkezésünkre álló webhelyeket és virtuális könyvtárakat. A könyvtárak beállításával kapcsolatos bővebb információkat a könyvtárra jobb gombbal kattintva, a helyi menü **Properties** pontjának segítségével tekinthetjük meg. Megjelenik a Properties párbeszédpanel:



Mint látjuk, a Properties párbeszédpanel elég terjedelmes, bemutatja a könyvtár külső hozzáféréseinek minden jellemzőjét. Nem töltünk sok időt ezek tanulmányozásával, mert a jellemzők nagy részét az ASP.NET kezeli (és ezt a feladatot nem hárítja az IIS-re).



3. **Tekintsük meg a virtuális könyvtárfájl leképezéseit!** A **Configuration** gombra kattintva megtekinthetjük a fájl leképezéseket. Az IIS párbeszédpanel listában megjeleníti a könyvtárfájl leképezéseit.

Ezek a leképezések szabják meg az IIS-nek, hogy az adott kérést melyik DLL-nek kell kezelnie. A statikus fájl típusokat, például a HTM-et, a rendszer közvetlenül az ügyfélnek küldi vissza. Azonban a dinamikus lapok – amelyek tartalma a kérések között módosulhat – további feldolgozást igényelnek, ezért a rendszer ezeket meghatározott ISAPI DLL-ekhez rendeli. A fenti listában található DLL-típusok nem különösebben változatosak. Valójában, a könyvtárban a legtöbb fájl típust az ASPNET\_ISAPI.DLL fájl kezeli. Hamarosan részletesen megvizsgáljuk ezt a DLL-t. Vegyünk észre a listában egy másik DLL-t is: az ASP.DLL-t. Ez a DLL kezeli a klasszikus ASP-kéréseket (**Active Server Pages**, aktív kiszolgálóoldalak).

## A klasszikus ASP

A Microsoft eredetileg azért fejlesztette ki az ASP-t, hogy a kizárólag C++-szal fejlesztők tömegénél nagyobb fejlesztői táborot ösztönözzön a webfejlesztésre. Az IIS megjelenésével a Microsoft-platfomon rendelkezésre állt a webhelyek fejlesztésére alkalmas környezet. Még ma is találkozhatunk olyan webhelyekkel, amelyek tiszta ISAPI DLL webhelyek; csak a böngésző és a kiszolgáló közötti lekérdezősztringeket kell megvizsgáljunk. Például a lekérdezősztringbe ágyazva olyan fájlnevekkel találkozhatunk, mint az ACMEISAPI.DLL.

Azonban eléggé elszántnak kell lennünk ahhoz, hogy egy teljes webhelyet ISAPI DLL-ek segítségével hozzunk létre. Ha ISAPI DLL-jeinket C-ben vagy C++-ban írjuk, az irányítás a kezünkben van, és megszabhatjuk, hogy hogyan működjön, és mi működtesse a webhelyet. Az irányítással azonban felelősség is jár, mivel a C- vagy C++-szoftverfejlesztés kihívásokkal teli feladat.

Az ASP megjelenésével a Microsoft egyetlen ISAPI.DLL-t biztosított, az ASP.DLL-t. Az ASP webfejlesztők .asp kiterjesztésű (például az akarmi.asp) fájlokban kódolnak. Az ASP-fájlok gyakran a statikus HTML és kimenetet futásidőben előállító végrehajtható (szkirptnyelven megírt) szakaszok keverékét tartalmazzák. Például az 1.5. lista ASP-program kódja megjeleníti a HelloWorld oldalt, amely statikus HTML-t és futásidőben generált szöveget is tartalmaz. (A fájl a HelloWorld.asp fájl, amelyet megtalálunk a kapcsolódó példák között is.)

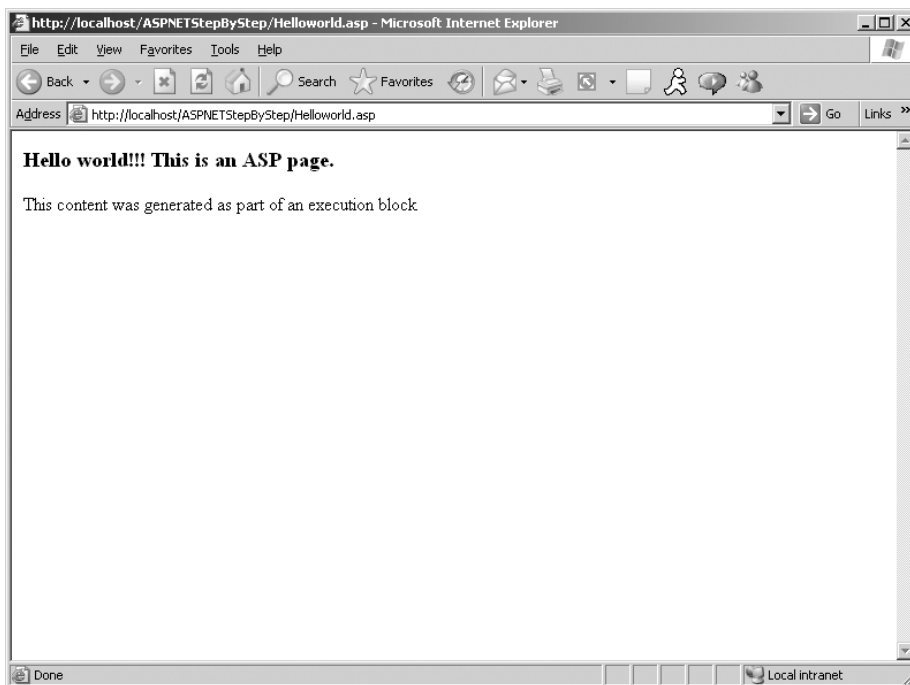
```
<%@ Language="javascript" %>
<html><body><form>

    <h3>Hello world!!! This is an ASP page.</h3>

    <% Response.write("This content was generated ");%>
    <% Response.write("as part of an execution block");%>
</form>
</body></html>
```

### 1.5. lista

Az 1.5. lista kódja a következő oldalt jeleníti meg. Az IIS a 80-as porton figyelte a kéréseket. Amikor a HelloWorld.asp fájl kérése beérkezett, az IIS látta az .asp fájlki-terjesztést, és (a fájl leképezések beállításai alapján) az ASP.DLL segítségét kérte a kérés kezeléséhez. Az ASP.DLL a statikus HTML-t a „Hello world!!! This is an ASP page.” sztringként jelenítette meg. Majd, amikor az ASP.DLL mókás megjelenésű végrehajtási címkékkel (<% és %>) találkozott, a blokkokat a JavaScript elemzőn keresztül hajtotta végre. Az 1.2. ábrán az Internet Explorerben megjelenő oldalt látjuk.



1.2. ábra Az 1.5. lista ASP-programjához küldött kérés eredménye

A könyv ASP.NET-szoftverfejlesztésről szól, ezért figyelmünket erre a területre összpontosítjuk. Mielőtt elbúcsúznánk a klasszikus ASP témakörétől, tekintsük meg az 1.6. listát, amely a SelectFeature.htm oldalt mutatja klasszikus ASP-oldalként átdolgozva. Ez az egyszerű ASP-alkalmazás bemutatja a webfejlesztés néhány központi témáját, valamint azt is szemlélteti, hogy a Microsoft miért írta át webkiszolgáló-technológiáját, és miért fejlesztette ki az ASP.NET-et. (A kapcsolódó fájl a SelectFeature.asp fájl.)

```
<%@ Language="javascript" %>
<html><body><form>

    <h2>HelloWorld</h2>

    <h3>what's your favorite .NET feature?</h3>
    <select name='Feature'>
        <option> Type-Safety</option>
        <option> Garbage collection</option>
        <option> Multiple syntaxes</option>
        <option> Code Access Security</option>
        <option> Simpler threading</option>
        <option> Versioning purgatory</option>
    </select>
    <br>
    <input type=submit name="Submit" value="Submit"></input>
    <p>
        Hi, you selected <%=Request("Feature") %>
    </p>
    </form>
</body></html>
```

#### 1.6. lista

A SelectFeature.asp szövege nagyon hasonlít a SelectFeature.htm fájl kódjához, ugye? A két fájl közötti különbség az első sorban (amely a végrehajtható blokk szintaxisát határozza meg), valamint a `<% és a %>` címkék által közrefogott végrehajtható blokkban rejlik. A statikus HTML többi része választó vezérlőelemet jelenít meg az űrlapon.

Tanulmányozzuk a végrehajtható blokkokat, és azt, hogy a blokkok a *Response* objektum – amelyet az ASP-infrastruktúra kezel – segítségével hogyan küldik el a szöveget a böngészőnek. A végrehajtható blokk megvizsgálja a *Feature* vezérlőelemet (amelyet a `<select>` címke határoz meg) és kiírja a felhasználó által kiválasztott értéket.

Az 1.3. ábrán a SelectFeature.asp által az Internet Explorerben megjelenített oldalt látjuk.

Az 1.3. ábrán látható képernyő kissé furcsának tűnhet, mert a legördülő listamezőben a „Type-Safety” értéket látjuk miközben az oldalon a „Multiple syntaxes” olvasható. Anélkül, hogy bármit is tennünk kellene, a legördülő listamezőben mindig az első elem jelenik meg a kiválasztott elemként.



böngésző értelmezi a HTML kódot. A végpont, amelyhez böngésző csatlakozik, egy webkiszolgáló (vagy kiszolgálófarm). Végül, az ügyfél és kiszolgáló által alkalmazott kapcsolati protokoll közvetett (és a kérés valóban bejárhatja az egész világot, mielőtt a felhasználó bármilyen eredményt látna).

A webes alkalmazásfejlesztésben a program elsődleges feladata, hogy a „kintről” jövő kéréseket fogadja, és a kérelmezők számára értelmes válasszal szolgáljon. Ez gyakran összetett HTML-kód generálását jelenti, amely az ügyfél böngészőjében olvasható formában jeleníti meg az információkat. A feladat elég bonyolult, mint például, amikor korszerű kereskedelmi webhelyet kell készítenünk. Az ügyfelek egészen biztosan érdeklődni fognak az aktuális árakról, a kérésnyilvántartási szintekről, és talán még cikkeket és szolgáltatásokat is rendelnek a webhelyről. Az „érthető HTML generálása az ügyfél számára” folyamat hirtelen olyan feladatokat jelent, mint például adatbázis-hozzáférések, az ügyfél azonosságának hitelesítése és az ügyfél rendelésének nyomon követése. Képzeljük el, hogy mindezt a semmiből kell felépítenünk!

Míg a klasszikus ASP-hez hasonló keretrendszerek nagyban hozzájárulnak ahhoz, hogy a webfejlesztés megközelíthetőbb legyen, rengeteg szolgáltatás megvalósítása továbbra is a fejlesztőkre vár (ezeknek a tulajdonságoknak a többsége a rész elején már említett két fontos kérdéshez kapcsolódnak). Például egy biztonságos, de kezelhető webhely elkészítése a klasszikus ASP-vel rendszerint annyit jelentett, hogy megírhattuk (vagy megvásárolhattuk) a biztonsági alrendszert. A webhely felhasználói felületének állapotkezelése is igen unalmas munka volt.

## ASP.NET

Mindezek elvezetnek bennünket az ASP.NET-hez. A könyvben sokszor visszaköszönő téma, hogy az ASP.NET fogja azokat a szolgáltatásokat, amelyeket a fejlesztők rendszerint (újra és újra) megvalósítanak, és belegyűrja az ASP.NET-keretrendszerbe.

Az ASP.NET 2.0 magasabb szintre emeli az ASP.NET 1.1-et, és a leggyakrabban használt szolgáltatások közül további szolgáltatásokat tol a keretrendszerbe. Az ASP.NET 1.1 hitelesítési és meghatalmazási szolgáltatásai remekül példázzák azt, hogy az ASP.NET 2.0 tökéletesíti az ASP.NET 1.1-et; az ASP.NET 1.1 elfogadható és könnyen kezelhető hitelesítési modellt foglalt magában. Azonban a fejlesztőknek gyakran saját hitelesítési rendszereket kellett megvalósítaniuk az egyes webhelyeken. Az ASP.NET 2.0 engedélyezési alrendszert foglal magában. Az ASP.NET űrlapos hitelesítésével és a többi biztonsági szolgáltatással részletesen a 10. fejezetben foglalkozunk.

A következő fejezetekben a legfontosabb ASP.NET-szolgáltatásokkal foglalkozunk. Az utolsó fejezet végére egy ASP.NET alapú webhely fejlesztéséhez elegendő ismeretanyaggal rendelkezünk majd.

## 1. fejezet – Gyorsreferencia

<b>Ehhez</b>	<b>Ezt kell tenni</b>
Az Internet Information Services konzol elindításához	Menjünk a Control Panelhez! Válasszuk az Administrative Tools menüpontot! Válasszuk az Internet Information Services-t!
Új virtuális könyvtár létrehozásához	Nyissuk meg az IIS Console-t! Nyissuk meg a Web Sites csomópontot! Nyissuk meg a Default Web Site csomópontot! Jobb egérgombbal kattintsunk a Default Web Site csomópontra! Válasszuk a New Virtual Directory menüpontot! Kövessük a varázsló utasításait!
Egy erőforrás megkereséséhez IIS-ből	Jobb egérgombbal kattintsunk az erőforrásra! Válasszuk a Browse menüpontot!
Az IIS által támogatott fájltypusok megtekintéséhez	Jobb egérgombbal kattintsunk a virtuális könyvtárra! Válasszuk a Properties menüpontot! Nyomjuk meg a Configure gombot!