# ASP.NET 2.0 (Whidbey)

## Mi várható a 2005. évi ASP.NET-ben?

### Client Callback (Ügyféloldali visszahívás)

Kevéssé reklámozott, de nagyon hasznos és trükkös új szolgáltatás a Client Callback. Segítségével anélkül változtathatjuk meg a böngészőbe betöltött lap tartamát, hogy a hagyományos postázásos módszerrel legeneráltatnánk újra a teljes lapot, így sokkal gyorsabb válaszidőket produkáló webalkalmazásokat írhatunk.

### Alapvetés

Hagyományos módon a webalkalmazások a HTTP protokollnak megfelelően kérés-válasz módon működnek, azaz a böngésző kér valamilyen oldalt, válaszul kap egy (általában) html oldalt. Ezt megjeleníti, majd a felhasználó által kitöltött adatokat újra postázza a kiszolgálóra, ami erre újabb választ generál, stb. Jó ez, mert minimális tudást követel meg a böngészőtől, de nem jó a felhasználónak, mert minden egyes interakció újabb, meglehetősen lassú oldalgenerálást szül. Amióta a web elindult, igyekeznek csökkenteni a visszapostázások számát, ennek két módja lehetséges: a letöltött adatokból táplálkozva programozzuk a böngészőben tárolt tartalmat, kihasználva a DHTML lehetőségeit, vagy a böngészőben futó kód „felnyúl" a kiszolgálóhoz, és az onnan kapott tartalmat beleszövi a már letöltött html tartalomba.
Mindkét megoldás hátránya, hogy sokkal többet vár a böngészőtől, mint a html tartalom puszta megjelenítését, a html tartalmat scriptből el kell tudni érni és módosítani.
DHTML programozás ma már minden modern böngészőben lehetséges, habár sokban kikapcsolható a scriptek futtatása, ilyenkor nincs mit tennünk, csak a hagyományos kérés-válasz modell fog működni.
A második módszer még szigorúbb a böngészőkre nézve, mert szükséges valamilyen scriptből elérhető komponens, amivel HTTP vagy egyéb kapcsolatot építhetünk ki az adatforrásként használandó kiszolgálóval. Internet Explorer 5.5 fölött ez megoldott a böngészővel kapott MSXML programcsomag XMLHTTP komponense révén. Szerencsére Mozilla alatt is használható, így minden, a következőkben leírt programnak működnie kell Mozilla alatt is. Én Firefox 1.0.4-gyel teszteltem őket.

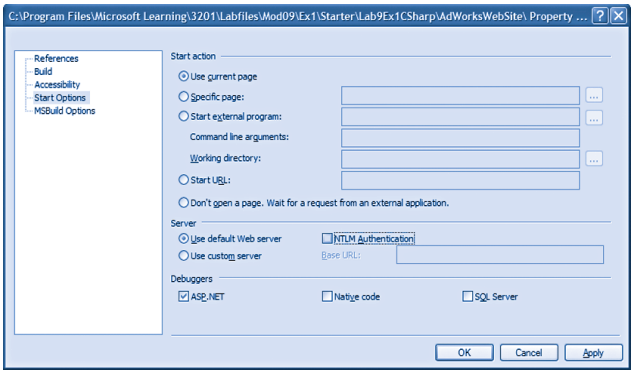
### A Client Callback működése

Írjuk meg a Hello World-öt, mint klasszikus bevezetőpéldát. A cél az, hogy egy gombnyomásra a Hello World szöveget kiszolgálóoldali kód generálja, de azt ne postázással adjuk vissza, hanem Client Callbackkel. Először jöhet a böngészőben futó kód váza:

```
<script type="text/javascript">
// <!CDATA[
function helloButton_onclick() {
    content.innerHTML = 'alma';
}
// ]]>
</script>

</head>
<body>
  <form id="form1" runat="server">
    <div>
      <input id="helloButton"
        type="button" value="Hello"
        onclick="return helloButton_onclick()" />
      <br />
      <div id="content">
      </div>
    </div>
  </form>
</body>
```

Az alma helyett szeretnénk a kiszolgáló által generált tartalmat beilleszteni. Mindjárt érdemes kipróbálni a fenti mégoly egyszerű scriptet Mozillában is, mert a böngészők objektummodellje sokban eltérhet egymástól. Az első kényelmetlenséggel azonnal szembesülünk: a Firefox jelszót vár tőlünk, annak ellenére, hogy az IE nem. Ennek oka, hogy a példát a B2-es Visual Studioban írom, és annak beépített webszerverét használom.
Az alapban Windows Authentikációt használ, ezért kért jelszót a Mozilla, míg az IE csendben elküldte a hitelesítési infókat. A kényelmesebb tesztelés kedvéért a projekt jellemzői között kikapcsoltam a hitelesítést.




**1. ábra: Windows hitelesítés kikapcsolása a VS.NET-ben**

Azonnal szembesülünk vele, hogy a fenti kód nem megy Mozilla alatt, semmi nem történik a gomb megnyomására! A Tools/JavaScript Console ablakában semmi nyoma hibának. Egy tesztként beszúrt `alert('alma')` működik, azaz legalább az eseménykezelőnk jó. A Mozilla DOM dokumentációt [1] tanulmányozva úgy tűnik, hogy az IE-ben megszokott rövidített objektumelérés nem megy a Firefoxban, hanem a `getElementById`-t kell használnunk. Igaz, ez a DOM szabványban [2] leírt működés, csak így jóval hosszabb kódot kapunk.
Ezzel a fenti eseménykezelőnk kódja így módosul:

```
document.getElementById('content').innerHTML =
'alma';
```

Itt az ideje átgondolni, mit kell tennünk kiszolgálóoldalon! A Client Callback technológia során az ügyféloldali script által küldött tartalmat egy `ICallbackEventHandler` implementáció képes lekezelni:

```
public interface ICallbackEventHandler {
  string RaiseCallbackEvent(string eventArgument);
}
```

Az `eventArgument` jön a böngészőtől, és a visszatérési értékként generált tartalom megy vissza ügyféloldalra.
Az interfészt implementálhatja egy control, így saját magával társaloghat az ügyféloldali és a kiszolgálóoldali része. Ezzel él is a `DetailsView`, `GridView` és `TreeView` is, ezeket megvizsgáljuk a cikk végén.

Ha nem egy vezérlőhöz kötődik a feldolgozás, akkor legegyszerűbb magában a lapban implementálni az interfészt:

```
public partial class HelloPage :
  System.Web.UI.Page, ICallbackEventHandler
{
  public string RaiseCallbackEvent(
    string eventArgument)
  {
    return "Hello World!";
  } ...
```

Már csak egy lépés hiányzik: kellene generálni egy olyan ügyféloldali kódrészletet, amely képes kiváltani a `RaiseCallbackEvent` meghívását. Ezt nem kézzel írjuk meg, hiszen nem ismerjük a Client Callback által használt script-függvények nevét, paramétereit, ezek az architektúra sajátjai. Segítségül vesszük viszont az erre a célra kialakított `GetCallbackEventReference` metódust:

```
string scriptDeclaration =
  this.ClientScript.GetCallbackEventReference(
    this, //1
    "arg", //2
    "HelloResponseArrived", //3
    "kornyezet", //4
    "HelloError", //5
    True //6
    );
```

A paraméterek jelentése:

1. Melyik vezérlő kezeli le az eseményt. Esetünkben a lap, ezért a `this`.
2. Hogyan hívjuk azt a paramétert az ügyféloldali függvényünkben, amiben a kiszolgálóra irányuló hívás paraméterét (string) adjuk át?
3. Ilyen néven kell létrehoznunk egy ügyféloldali függvényt a lapban, amely majd a 2. és a 4. nevű paramétert várja paraméterül. Ebben kapjuk meg a kiszolgáló válaszát.
4. Egy paraméternév, amelyen a hívást kiváltó kód (esetünkben a click eseménykezelő) paramétert képes indirekten átpasszolni a hívás végét jelző függvénynek (3. paraméter). Arra jó, hogy így összeilleszthetjük a hívás megindítását a tényleges befejezéssel.
5. Hiba esetén ez az ügyféloldali metódus hívódjon meg.
6. Aszinkron módon történjen-e a visszahívás a kiszolgálóra? Miért ne?

A `GetCallbackEventReference` által generált script:

```
WebForm_DoCallback('__Page',arg,DisplayDateTime,
kornyezet,ErrorCallback,true)
```

Ez képes kiváltani a kiszolgálóoldali `RaiseCallbackEvent`-et. Az egyszerűbb kezelés érdekében ezt érdemes becsomagolni egy saját JavaScript függvénybe:

```
scriptDeclaration = string.Format(
  @"function {1}(arg, kornyezet) {{ {0} }}",
  scriptDeclaration, "Hello");
```

Az így nyert egyszerűsített függvény, ami kiváltja a visszahívást:

```
function Hello(arg, kornyezet)
{
  WebForm_DoCallback('__Page',arg,
  HelloResponseArrived,kornyezet,HelloError,true)
}
```

Ezt már csak bele kell ágyazni a lapba:

```
ClientScript.RegisterClientScriptBlock(
  GetType(),
  "HelloCallScript",
```

```
  scriptDeclaration,
  true);
```

Az új Hello függvényünkkel már könnyedén hívható a kiszolgálóoldal (nem adunk át semmilyen paramétert, a JavaScript nem haragszik érte):

```
function helloButton_onclick() {
  Hello();
}
```

A válasz pedig ide érkezik meg (aszinkron módon, egyszer csak):

```
function HelloResponseArrived(valasz, kornyezet) {
  document.getElementById('content').innerHTML =
     valasz;
}
```

Álljon itt a teljes kód, hogy egyben végig lehessen olvasni. A teljes lap forrása, fejlécek nélkül:

```
<script type="text/javascript">
function helloButton_onclick() {
  document.getElementById('content').innerHTML =
    '';
  Hello();
}

function HelloResponseArrived(valasz, kornyezet) {
  document.getElementById('content').innerHTML =
    valasz;
}

function HelloError() {
    alert('Hiba a kiszolgáló hívása közben.');
}
</script>

<body>
  <form id="form1" runat="server">
    <div>
      <input id="helloButton"
        type="button" value="Hello"
        onclick="return helloButton_onclick()" />
      <br />
      <div id="content">
      </div>
      <asp:Label ID="statusLabel"
        runat="server" Text="Label" />
    </div>
  </form>
</body>
```

A háttérkód:

```
using Ez meg az;

public partial class HelloPage:
   System.Web.UI.Page, ICallbackEventHandler {
  public string RaiseCallbackEvent(
    string eventArgument) {
    return "Hello World!";
  }
  protected void Page_Load() {
    string scriptDeclaration =
      this.ClientScript.GetCallbackEventReference(
        this,
        "arg",
        "HelloResponseArrived",
        "kornyezet",
        "HelloError",
        true
        );
    scriptDeclaration = string.Format(
    @"function {1}(arg, kornyezet) {{ {0} }}",
    scriptDeclaration, "Hello");
    statusLabel.Text = scriptDeclaration;
    ClientScript.RegisterClientScriptBlock(
      GetType(),
      "HelloCallScript",
      scriptDeclaration,
      true);
  }
}
```

## Fejlett HTML tartalomgenerálás WebControlokkal

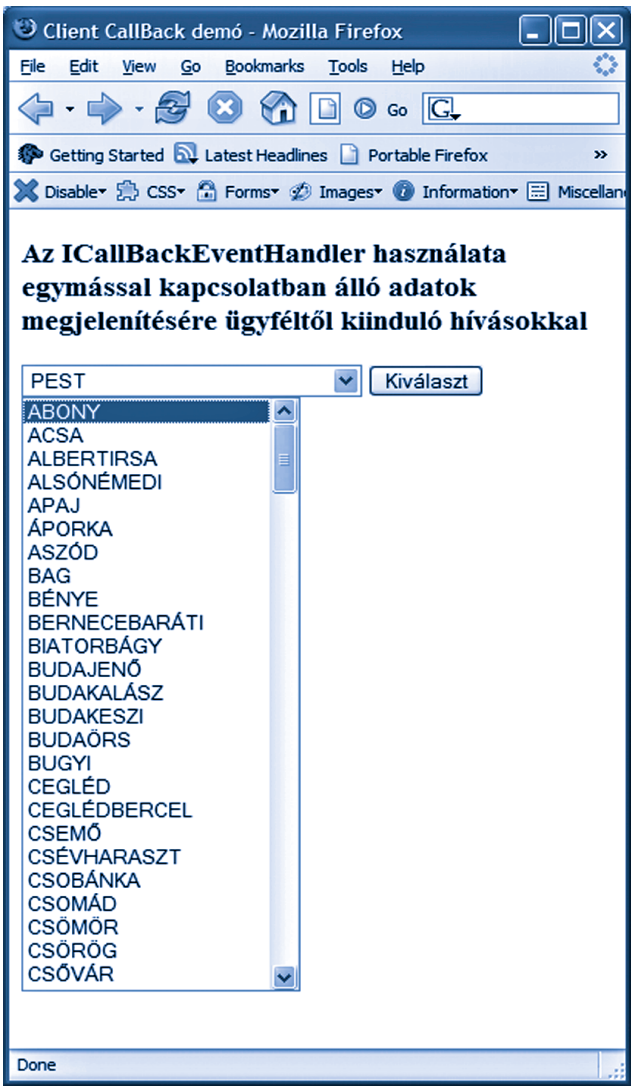
Az interneten keresgélve sok példát találni a Client Callback használatára, ám a legtöbb egy stringgé átalakított tömböt küldd vissza, és ebből generál a böngészőben futó script html tartalmat, pl. egy listát.
Alapvetően az nem tetszik ebben a megközelítésben, hogy a munka zömét ügyféloldalon kell leprogramozni, az pedig (szubjektív szempont) nekem nem barátságos környezet. Ha már egyszer vannak sokat tudó Server Controlok, mint pl. a `DataGrid` és társaik, akkor miért nem használjuk ki a html generálási képességeiket a Client Callbackek esetén is? A következő példa demonstrálja, hogy ez lehetséges. (Köszönet Kopinak és és Zokszigennek az ötletért).
A példa célja, hogy egy listából kiválasztva egy megyét, megjelenítsük az ahhoz tartozó települések listáját. Egy listában a megyék, másikban a települések, ám nem töltjük le az összes települést, hanem csak Client Callbackkel azt, amit éppen kiválasztottak (1. ábra).
A megoldáshoz kell egy megyék lista, amelyet `SqlDataSource` és egy `asp:ListBox` generál, a cikksorozat korábbi részeiben ismertetett adatkötés felhasználásával:

```
<asp:DropDownList ID="megyeLista" OnChange=
"TelepulesListaLetoltes(this.options[
  this.selectedIndex].value, 'soci');"
  runat="server" DataSourceID="megyeDataSource"
  DataTextField="Nev"
  DataValueField="ID">
</asp:DropDownList>

<asp:SqlDataSource ID="megyeDataSource"
  runat="server" ConnectionString="<%$
ConnectionStrings:MOConnectionString %>"
  EnableCaching="True"
  SelectCommand=
```

```
  "SELECT [ID], [Nev] FROM [Megye]
   ORDER BY [Nev]">
</asp:SqlDataSource>
```

Az érdekes részek az előbbi lista `OnChange` eseményében történnek. Ez az ügyféloldali esemény meghívja a TelepulesListaLetoltes függvényt, ami már a kiszolgálóoldalról tölti le a kiválasztott megye településeit.
Ez a már látott módon a GetCallbackEventReference és társaival beregisztrált függvény.




**2. ábra: Megyék-települések megjelenítése**

A példa érdekessége kiszolgálóoldalon lapul:

```
int megyeID;

public string RaiseCallbackEvent(string
eventArgument)
{
  megyeID = int.Parse(eventArgument);

  string telepulesListaHtml =
    Cache[megyeID.ToString()] as string;

  if (telepulesListaHtml == null)
  {
    //Lista generálás és cache-elés
    Cache[megyeID.ToString()] = telepulesListaHtml
        = GenerateTelepulesList();
  }
  return telepulesListaHtml;
}
```

A kapott megye paramétertől függően vagy elővesszük Cache-ből a településlistát, vagy generáljuk. A generálás a ravasz, hisz most egy Server Controlt arra kell megkérni, hogy ne a szokásos `Response.OutputSream`-be generáljon tartalmat, hanem adja nekünk a kimenetét. Azaz lesz egy lista és egy adatforrás vezérlőnk a lapon, de a listát nem mutatjuk meg a lap betöltése során, hiszen ezt majd ügyféloldalról kezeljük. Viszont a VS 2005 tervezőit remekül használhatjuk, hiszen a vezérlőket a megszokott módon a lap tartalmazza:

```
<asp:ListBox ID="telepulesLista"
  runat="server"
  DataSourceID="telepulesDataSource"
  DataTextField="Nev"
  DataValueField="ID"
  Rows="25" Visible="False">
</asp:ListBox>
```

Emlékeztetőül, `Visible="False"`, mert a lap szokásos kimenetében nem akarjuk, hogy megjelenjen.
Az adatforrás:

```
<asp:SqlDataSource ID="telepulesDataSource"
  runat="server" ConnectionString="<%$
ConnectionStrings:MOConnectionString %>"
  SelectCommand="SELECT [Nev], [ID]
  FROM [Telepules]
  WHERE ([MegyeID] = @MegyeID)
  ORDER BY [Nev]"
  OnSelecting="telepulesDataSource_Selecting">
  <SelectParameters>
    <asp:Parameter Name="MegyeID"
      Type="Int32" />
  </SelectParameters>
</asp:SqlDataSource>
```

Az adatforrás `OnSelecting` eseményét le kell kezelnünk, mert át kell adnunk a szűrési feltételként, paraméterként a megye azonosítóját:

```
protected void telepulesDataSource_Selecting(
 object sender, SqlDataSourceSelectingEventArgs e)
{
  e.Command.Parameters[0].Value = megyeID;
}
```

A `megyeID`-t már beállítottuk a felküldött megyére a `RaiseCallbackEvent` metódusban (ld. pár blokkal korábban).
Már csak egy részlet hiányzik a teljes képhez: rá kell venni a listát, hogy egy stringbe generálja a tartalmat, ne a szokásos `Response`-ba:

```
private string GenerateTelepulesList()
{
  telepulesLista.Visible = true;
  telepulesLista.DataBind();
  telepulesLista.SelectedIndex = 0;
```

```
  StringBuilder telepulesListaHtml =
    new StringBuilder();
  HtmlTextWriter w =
    new HtmlTextWriter(
    new StringWriter(telepulesListaHtml));

  telepulesLista.RenderControl(w);
  return telepulesListaHtml.ToString();
}
```

A `ListBox` kap egy általunk létrehozott `HtmlTextWriter`-t, és ahogyan a lap is szokta, megkérjük, generálja le a html tartalmát a RenderControl metódussal. A példa cache-eli a tartalmat kiszolgálóoldalon, ez tovább javítható ügyféloldali cache-eléssel. Ha már egyszer letöltődött egy megyéhez tartozó településlista, miért ne tárolnánk el azt a böngészőben?
Egy lehetséges megoldást mutatok be a következő kódblokkban:

```
var telepulesek = new Object();
function TelepulesekLejottek(result, context) {
    telepulesek[context] = result;
    FillList(result);
}
function FillList(listContent) {
  document.getElementById(
    "telepulesListaPlaceholder").innerHTML =
    listContent;
}

//A korábbi eseménykezelő helyett
function MegyeListChanged() {
  var list =
    document.getElementById("megyeLista");
  var megyeID =
    list.options[list.selectedIndex].value;
  if (telepulesek[megyeID] != null) {
    FillList(telepulesek[megyeID]);
  }
  else {
    TelepulesListaLetoltes(
    megyeID, //arg
    megyeID); //context
  }
}
```

A megoldásban kihasználtam, hogy a `context` paraméterben átpasszoltam a megye azonosítóját a `MegyeListChanged`-ből a `TelepulesekLejottek`-be (`GetCallbackEventReference` 4. paraméter).
A letöltött településlistát a `telepulesek` objektum property-jeiben tároltam el (egyfajta asszociatív tár vagy Hastable jellegű működés), expando property a hivatalos neve. Jól megfigyelhető a gyorsulás, ha IE-ben rávisszük a fókuszt a megyelistára, majd a fel-le billentyűkkel mozogva a listában a települések lista először lomhán jön be, de miután egy-egy települést betárazott, nagyon gyors lesz.

## A hívások egyszerűsítése

A GetCallbackEventReference és társai logikus, de eléggé nehézkes felületet adnak a visszahívások kezeléséhez. Ha alkalmazásunk sok visszahívást használ, érdemes beburkolni az előkészítő funkciókat egy Server Controlba, ezzel megkönnyítve a szolgáltatás használatát.
Paul Glavich barátunk írt ehhez egy egyszerű vezérlőt [3], annak használatát mutatom itt be. A forráskódot alaposan átírva beleraktam a cikkhez mellékelt demóprojektbe is. Egyrészt az eredeti példa ViewState-ben tárolta a jellemzők adatait, ezt átírtam Control State-re, így nem hülyül meg a vezérlő, ha kikapcsolják a ViewState-et.
Másrészt nagyon sok redundáns kód volt benne, azokat megszüntettem (a szerző szereti a kopi-pasztát).
A vezérlő érdekessége, hogy nem csak kézzel váltható ki a visszahívás, mint az eddigi példákban, hanem periodikusan is, egy timerre építve is. Ez automatikusan frissülő tartalomhoz jó lehet, de vigyázni kell vele, mert pár száz futó ügyfél komoly terhelést okozhat a kiszolgálónak.
Példa a vezérlő használatára:

```
<cc1:AsyncClientScriptConnector
  ID="sc" runat="server"
  ClientSideCallbackMethod="NormalCallback"
  ClientSideErrorMethod="ErrorCallback"
  AutoRepeatInterval="1000"
  InitiateCallBackClientMethod="StartCallback"
  OnServerSideCallback="as_ServerSideCallback" />
<asp:TextBox ID="txtMsg" runat="server" />
<input id="Button1" type="button" value="Manual
Call" onclick="return InvokeCallback()" />
```

Azaz paramétereznünk kell három ügyféloldali és egy kiszolgálóoldali függvénnyel, melyek felépítése hasonló a korábban leírtakhoz. Bővebben a példakódban [6].

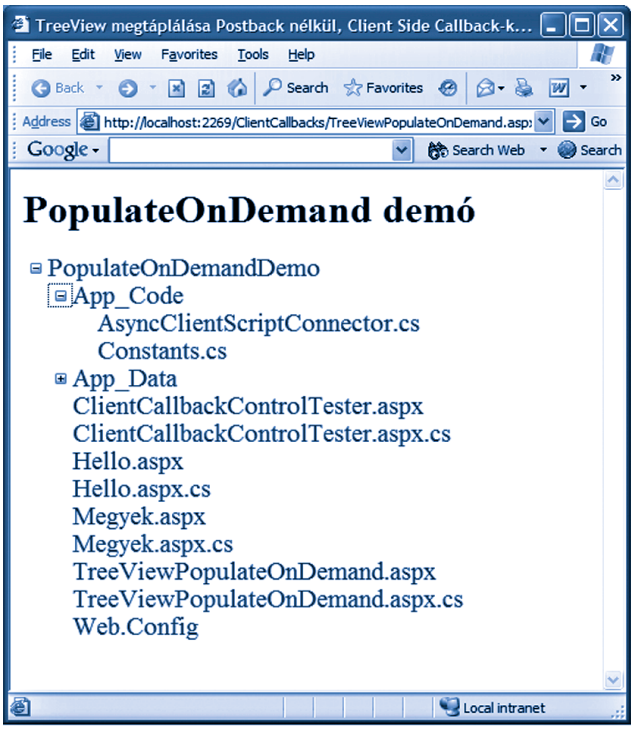
## Objektumorientált, típusos adatátvitel

Az eddigi példában stringek utaztak a két oldal között, mivel alapban csak erre ad lehetőséget a technológia. Némi kézimunkával azonban összetett objektumokat is átvihetünk, így pl. egy kiszolgálóoldalon definiált `Employee` objektumot át tudunk vinni ügyféloldalra, ahol annak tagjaira szokásos objektumorientált módon .-tal tudunk hivatkozni: `emp.LastName`.
A probléma azért nem egyszerű, mert az ügyféloldali JavaScript típusrendszere jelentősen egyszerűbb, mint a CLR típusrendszer, így hogyan lehet leképezni pl. egy `decimalt` vagy egy `Guid`-ot JavaScript típusokra?
Persze fordítva is eljárhatnánk, és korlátozzuk magunkat kiszolgálóoldalon a JavaScript típusainak megfelelő CLR típusokra. Mindkét megoldás kompromisszumokkal terhes. Az első, proxy jellegű működést az Ajax.NET-ben figyelhetjük meg [4], ami nem a cikkben ismertetett módon, de Client Callbacket tesz lehetővé.
A két típusrendszer JavaScript alapú közös nevezőjét kereső írást szintén érdemes elolvasni, az az ASP.NET Callbackre épít [5]. Szerintem a Microsoft a következő verzióban fog valamilyen megoldást adni a problémára, addig a fenti címeken ötleteket láthatunk a saját implementációhoz.

## Client Callbacket használó vezérlők: TreeView

Ha már van ez az új szolgáltatás, miért ne építhetnének rá az új vezérlők is? A `TreeView` egy új, de mégis ismert vezérlő. Az Internet Explorer WebControls már régóta tartalmaz egy fa implementációt, azt porolták le, írták át.

Nem célom magát a vezérlőt részletesen bemutatni, ami számunkra most érdekes, az, hogy amikor a böngészőben a fa egy elágazására kattintanak, akkor kiváltható egy kiszolgálóra irányuló visszahívás, amiben elő tudjuk állítani az éppen kinyitandó ág elemeit.




**3. ábra. A fa ágainak kinyitásakor kiszolgálóoldalon fut le a tartalmat generáló kód**

Indulásként egyetlen node, a gyökér lesz csak a fában:

```
<asp:TreeView ID="MyTree"
  PathSeparator="|"
  OnTreeNodePopulate="PopulateNode"
  ExpandDepth="1" runat="server">
  <Nodes>
    <asp:TreeNode Text="PopulateOnDemandDemo"
     PopulateOnDemand="true" />
  </Nodes>
</asp:TreeView>
```

Ha egy ágnak adatokra van szüksége, Client Callbackkel visszahívja a kiszolgáló `OnTreeNodePopulate` eseményben megadott metódusát.
A `PopulateNode` felépítése:

```
static readonly char[] _slashArray =
  new char[] { '/' };

protected void PopulateNode(Object source,
TreeNodeEventArgs e)
{
  TreeNode node = e.Node;
  if (node.Value == "PopulateOnDemandDemo")
    node.Value = "~/";

  string rootDirectory =
    Request.MapPath("~/",
    Request.ApplicationPath, false);
  string fullPath =
    Request.MapPath(node.Value,
    Request.ApplicationPath, false);

  //Egyszerű, de nem biztos,
  //hogy tökéletes védelem,
  //hogy ne lépjenek
  //fel felsőbb könyvtárakba.
  if (!fullPath.StartsWith(rootDirectory))
  {
    return;
  }

  AppendDirectoryNodes(node, fullPath);
  AppendFileNodes(node, fullPath);
}
```

Azaz kapunk egy referenciát egy, az ügyféloldali Node-ot reprezentáló objektumra (ami persze már egy kiszolgálóoldali típus), amiből megtudjuk, hová kattintottak. Ha nem akarják egyből a windows\system32-t olvasni, akkor legeneráljuk a könyvtárakat és a fájlokat tartalmazó node-okat:

```
private void AppendDirectoryNodes(TreeNode parent,
string fullPath)
{
  string[] dirs =
    Directory.GetDirectories(fullPath);

  //Nézzük végig az összes könyvtárat
  foreach (string dir in dirs)
  {
    string virtualDir =
      parent.Value.TrimEnd(_slashArray) + "/" +
      Path.GetFileName(dir);

    TreeNode newNode =
      new TreeNode(
      Path.GetFileName(dir), virtualDir);

    //Ha még van alatta bejegyzés,
    //akkor ezt se töltjük fel előre
    if (Directory.GetFiles(dir).Length > 0
      || Directory.GetDirectories(dir).Length > 0)
    {
      newNode.PopulateOnDemand = true;
    }
    parent.ChildNodes.Add(newNode);
  }
}
```

A dinamikusan létrehozott könyvtárak is `PopulateOnDemand = true`-ra vannak állítva, így őket is ki lehet majd nyitogatni. A fájlok listázásában egyedül a fájlnevekkel való bűvészkedés okozott némi fejtörést:

```
private void AppendFileNodes(TreeNode parent,
string fullPath)
{
  string[] files = Directory.GetFiles(fullPath);
  foreach (string file in files)
  {
    TreeNode newNode = new TreeNode(
      Path.GetFileName(file),
      Path.GetFileName(file));

    //Tudom, tudom, csúnya...
    newNode.NavigateUrl =
      Request.ApplicationPath + '/' +
      file.Replace(
      Request.PhysicalApplicationPath, "").
      Replace('\\', '/');
    parent.ChildNodes.Add(newNode);
  }
}
```

### Client Callbacket használó vezérlők: GridView, DetailsView

A GridView képes Client Callbackkel rendeztetni valamely oszlopra kattintva a táblázat tartalmát, illetve képes lapozni, anélkül, hogy a teljes oldal újratöltődne. Ehhez csak annyit kell tennünk, hogy az EnableSortingAndPagingCallbacks jellemzőt true-ra kell állítani. A DetailsView lapozni tudna callbackekkel, ha az EnablePagingCallbacks-szel azt engedélyezzük. Sajnos egyik vezérlőben sem működnek még a visszahívások a Beta2-ben (nekem), ezért ezt a témát egyelőre elnapoljuk.

### Példakódok

A cikkben kidolgozott példakódok jóval hosszabbak és jobban kommentezettek, mint ahogyan itt közöltük, ezért érdemes megnézni őket teljes „pompájukban” a [6] címen.

### Zárszó

Érdekes új szolgáltatás a Client Callback, amelyen még látszik az újszülött kor, de ennek ellenére újabb ravasz fegyvert ad a kezünkbe kényelmesebb Webalkalmazások írásához. Ahogyan egyre többen használják a 2.0-s ASP.NET-et, úgy valószínűleg sokféle, magasabb szintű megoldás fog kialakulni, amelyek egyszerűbbé teszik a visszahívásos elven működő alkalmazások írását. Ám, mint minden új, kevésbé kitaposott út, izgalmas gondolkodási lehetőséget biztosít a kreatív elmék számára eddig nem látott alkalmazások kifejlesztésére.
Sok sikert ehhez a Kedves Olvasónak!

Soczó Zsolt
*zsolt.soczo@netacademia.net*
*A szerző a NetAcademia vezető fejlesztőoktatója*
*ASP.NET MVP, MCSE, MCSD, MCDBA, MCT*

**A cikkben szereplő URL-ek:**

[1] www.mozilla.org/docs/dom/
[2] www.w3.org/DOM/
[3] weblogs.asp.net/pglavich/archive/2005/05/11/406348.aspx
[4] weblogs.asp.net/mschwarz/
[5] weblogs.asp.net/bleroy/archive/2005/05/19/407539.aspx
[6] netacademia.net/tudastar/articlepage.aspx?upid=6658

# Microsoft Worldwide Partner Conference 2005

„Az elkövetkező két évben valamennyi termékvonalunkon fogunk új, integrált szoftvermegoldásokat szállítani — olyan megoldásokat, amelyek páratlan üzleti értéket és jelentős előnyöket teremtenek partnereink számára világszerte” - mondta Steve Ballmer, a Microsoft vezérigazgatója a vállalat nemzetközi partnerkonferenciáján.

A konferencián első alkalommal mutatták be nyilvánosan azon űrlapfunkciókat, amelyek a Microsoft Office Rendszer jelenleg „Office 12” névvel jelölt következő generációjában fognak szerepelni. A Microsoft bemutatta az RTC eszközkészletet, amely a Microsoft Visual Studio projektekben felhasználható vizuális vezérlőket tartalmaz. Ezek használatával az alkalmazásfejlesztő partnerek egyszerűen be tudják építeni a jelenléttel, azonnali üzenetküldéssel és hívásvezérléssel kapcsolatos funkciókat a saját üzleti alkalmazásaikba.
A Microsoft Windows következő jelentős verziója egy újfajta számítástechnika alapjait teremti meg. A Longhorn a biztonsági és megbízhatósági alapfunkcióktól kezdve, a látványos, új megjelenésig minden területen jelentős előrelépéseket hoz. A Microsoft arra ösztönzi a fejlesztőket és a független szoftvergyártókat, hogy teljes mértékben használják ki a Longhornhoz kapcsolódó, jelenleg WinFX néven ismert új programozási modellt, amely kiterjeszti a .NET-keretrendszert, és újszerű alkalmazások létrehozására ad lehetőségeket az iparági partnerek számára. A Longhorn első bétaverziója idén nyáron várható.
A Microsoft piacra dobja a Windows Small Business Server 2003 R2 verzióját, amely a Windows Server 2003 R2 kibocsátását követő 60-90 napon belül kerül gyártásba. A kisvállalkozásokat célzó új termékek és szolgáltatások többek között a következők: SQL Server 2005 Workgroup Edition, javítás- és frissítés-kezeléssel, illetve megnövelt méretű postafiókokkal (75 GB).
A Microsoft System Center Data Protection Manager (DPM) a Microsoft első lépését jelenti a lemez alapú biztonsági mentési és helyreállítási területen. A DPM csomag egy DPM kiszolgálói és három kezelési licencet fog tartalmazni. A Data Protection Manager termék a Microsoft Partner Program keretében a Speciális Infrastruktúra kompetencia Rendszerfelügyeletébe lesz integrálva, ilyen módon a partnerek számára többek között projektkalauzokból, online tanfolyamokból, vizsgákból és minősítésekből álló átfogó eszközkészlet és oktatás válik hozzáférhetővé.
Az SQL Server 2005, a Visual Studio 2005 és a Biz Talk Server 2006 ez év novemberében fognak megjelenni.

http://www.microsoft.com/partner/events/wwpartnerconference