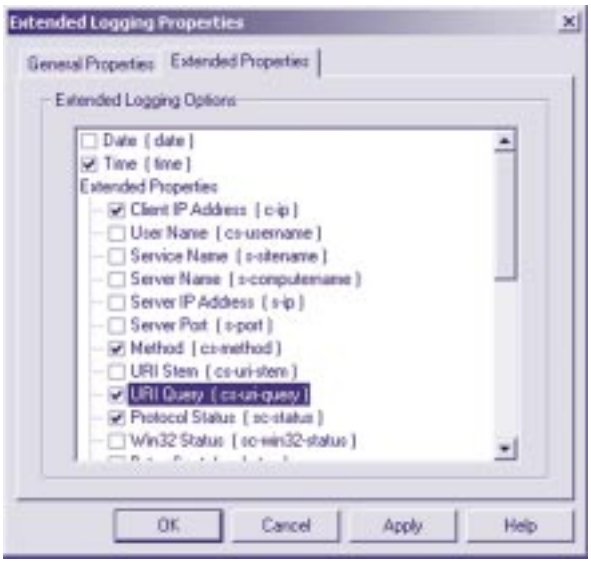
ASP suli 2.

Elôzô számunkban bekacsintottunk az ASP alapjaiba, szemügyre vettük a két talán legfontosabb objektumot, a HTTP kérést és választ jelképezô Request-et és Response-ot. Akkor kimaradt néhány dolog, ezért most folytassuk ott, ahol egy hónappal ezelôtt abbahagytuk, azután pedig rövid mese következik az ASP Session mibenlétérôl. E számunk példaprogramjai természetesen megtalálhatók a [1] címen.

Írjunk az IIS naplójába!

Így van, írhatunk, ha nagyon akarunk, de körültekintônek kell lennünk. A **Response.AppendToLog()** metódusnak átadott szövegrész bekerül az IIS naplófájljába *(tehát nem az Eseménynaplóba!)*. A szöveg nem tartalmazhat vesszôt, mert a naplófájl egyes mezôit vesszôk választják el, és ez megzavarhatná a naplók késôbbi feldolgozását. A bejegyzés csak akkor kerül be a naplóba, ha az IIS naplózás beállításai között bekattintottuk az „URI Query” mezôt.



A sikeres egyedi naplóbejegyzés kulcsa az URI Query mezô engedélyezése

Bánjunk óvatosan ezzel a lehetôséggel. Ha az IIS naplója W3C vagy NCSA formátumban készül, az általunk átadott szöveg a naplóban a kért URL helyén *(W3C formátum esetén)*, vagy ahhoz hozzáfûzve *(NCSA)* jelenik meg. Ennek nyilvánvalóan sok értelme nincsen, ezért én azt javaslom, hogy az ilyen bejegyzéseket írjuk inkább szövegfájlba, vagy – ha mindenképpen ennél a megoldásnál akarunk maradni – használjuk a Microsoft IIS naplóformátumot.

Felhasználóazonosítás névvel és jelszóval

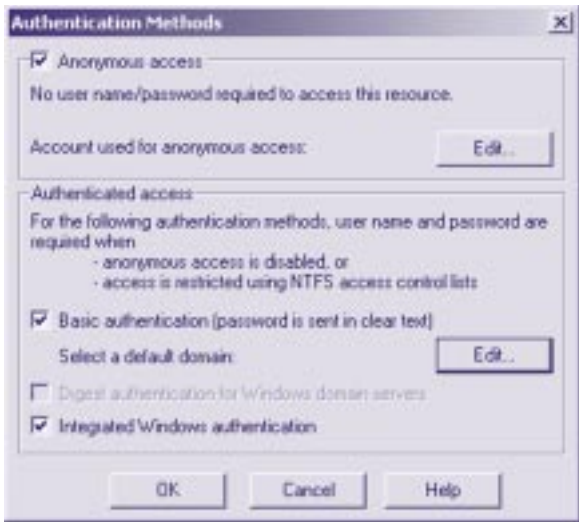
A webkiszolgáló tartalmának elérését sokszor korlátozni szeretnénk. Szép dolog a szabadság, de elôfordulhat, hogy bizonyos adatokhoz való hozzáférés elôtt szükség van a felhasználók azonosítására. A HTTP természetesen magában hordozza ennek lehetôségét is, és az sem kétséges, hogy a felhasználónevet és jelszót kérô kis ablakkal már mindannyian találkoztunk. De vajon tudjuk-e, mi zajlik ilyenkor a háttérben? Elôször is, a böngészô egy hagyományos kérést küld a kiszolgálónak. Amikor az alapértelmezett anonymous felhasználó *(aki az IIS esetén egyébként megfelel az IUSR_számítógépnév felhasználónak)* nem jogosult egy kért erôforrás elérésére, a kiszolgáló egy „401 Unauthorized” üzenettel válaszol:

```
HTTP/1.1 401 Unauthorized
Server: Microsoft-IIS/5.0
Date: Mon, 05 Feb 2001 21:05:25 GMT
WWW-Authenticate: Negotiate
WWW-Authenticate: NTLM
WWW-Authenticate: Basic realm="localhost"
Content-Length: 0
Content-Type: text/html
Cache-control: private
```

A lényeg az aláhúzott sorokban van: mindenekelôtt, természetesen a legfontosabb a 401-es kódú HTTP válaszüzenet, ami azt jelzi az ügyfélnek, hogy a hozzáférést megtagadtuk. A WWW-Authenticate HTTP fejlécek a lehetséges felhasználóazonosítási módszereket jelzik. Ezekbôl természetesen több is van, bár az Internet Explorer kivételével szinte mindegyik böngészô csak a Basic azonosítást ismeri. A Basic azonosítással viszont két nagy baj van:

1. A Basic felhasználóazonosítás során a jelszó kódolatlanul utazik a hálózaton mindaddig, amig valamilyen kiegészítô megoldással *(pl. https://)* ezt át nem hidaljuk
2. Az IIS5 alapértelmezésben nem engedélyezi a Basic típusú azonosítást; ez természetesen azt jelenti, hogy az Internet Explorer-en kívül más böngészôvel csak az anonymous által egyébként is elérhetô oldalakhoz férhetünk hozzá.

A használható felhasználóazonosítási módszerek listáját és beállításait az adott webhely tulajdonságlapján, a Directory Security fülön, az Anonymous access and authentication control mezôben található Edit... gombra kattintva megjelenô dialógusablakban találjuk:



A Basic (nyílt jelszavas) felhasználóazonosítás nem alapértelmezés, itt kapcsolhatjuk be

Alapértelmezés, hogy felhasználóazonosításra akkor van szükség, ha az anonymous felhasználó hozzáférési jogai egy adott feladathoz már nem elegendôk.
Felhasználóazonosítást tehát legegyszerûbben úgy kényszeríthetünk ki, ha az NTFS fájlrendszerre telepített IIS könyvtára alatt *(ez az \inetpub\wwwroot)* a kívánt fájlo*(ko)*n vagy könyvtár*(ak)*on korlátozzuk az IUSR_számítógépnév felhasználó hozzáférési jogait. Az IIS ilyenkor automatikus felhasználóazonosításba kezd, és csak akkor engedi „be” a felhasználót, ha ôt a megadott jelszó segítségével a felhasználói adatbázisban sikeresen azonosította.
Ha ezt nem akarjuk, az azonosítást kérhetjük kódból is: mint már tudjuk, a felhasználóazonosítást tulajdonképpen egy 401-es kódú HTTP válaszüzenet váltja ki. Vajon mi történik, ha a Response.Status segítségével mi magunk küldjük vissza ezt az üzenetet, valahogy így:

```
<% Response.Status = "401 Unauthorized" %>
```

Nos, elárulhatom, hogy a legjobbak történnek. A státuszüzenet mellé az IIS automatikusan mellékeli a megfelelô WWW-Authenticate mezôket és elvégzi helyettünk a felhasználóazonosítást.
A felhasználó neve, jelszava *(ha elérhetô)* és a felhasználóazonosítás típusa bármikor megtalálható a ServerVariables kollekcióban:

```
Request.ServerVariables("AUTH_TYPE")
Request.ServerVariables("AUTH_USER")
Request.ServerVariables("AUTH_PASSWORD")
```

A jelszó csak akkor látható, ha a típus „basic”; más esetekben az AUTH_PASSWORD mezô értéke üres. Ha nem volt felhasználóazonosítás *(pl. mert anonymous módon értük el az adott scriptet)*, az AUTH_USER értéke is üres lesz. Lássunk ezután egy példát, ami felhasználóazonosítást kér, majd ha az sikeres volt *(azaz ha az IIS a Windows 2000/NT felhasználói adatbázisában a felhasználót megtalálta)*, kiírja a fenti adatokat *(logon.asp)*:

```
<%
  If Request.ServerVariables("AUTH_USER")="" Then
    Response.Status = "401 Unauthorized"
    Response.End
  End If
%>
<HTML>
  <HEAD><TITLE>User LogOn Page</TITLE></HEAD>
  <BODY>
    AuthType: <% =
    ↳ Request.ServerVariables("AUTH_TYPE") %><BR>
    Username: <% =
    ↳ Request.ServerVariables("AUTH_USER") %><BR>
    Password: <% =
    ↳ Request.ServerVariables("AUTH_PASSWORD")%>
    <BR>
  </BODY>
</HTML>
```

A fenti kód elsô része ellenôrzi, hogy a felhasználó azonosította-e már magát. Ha nem *(a felhasználónév üres)*, visszaküldi a státuszüzenetet, majd befejezi az oldal végrehajtását. Miután a felhasználó bejelentkezett, a vezérlés már eljut a valódi tartalomhoz: kiírjuk a felhasználóazonosítás típusát, a nevét és jelszavát. Érdemes megfigyelni, hogy a különféle böngészôk és azonosítási módszerek hogyan befolyásolják az adatokat: Basic azonosítás esetén például látható a jelszó, míg a Negotiate módon azonosított felhasználó nevében benne van a tartomány neve is *(a \ jel elôtti rész)*.

Ugyanaz az oldal, bejelentkezés után az Internet Explorer és a Netscape esetében. Jól látható a tartomány neve, illetve a Netscape oldalán a nyílt jelszó

A felhasználóazonosítás egy másik, új, IIS5-ben bemutatott módja a tanúsítványokkal, felhasználónév és jelszó nélkül történô, automatikus azonosítás; errôl egy késôbbi alkalommal beszélünk majd.

Egy megjegyzés a Request kollekciókról

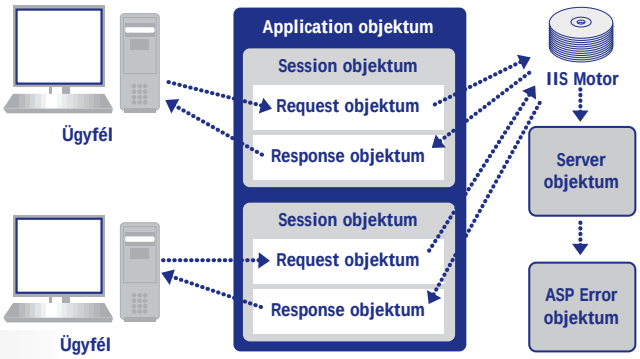
A Request objektum lehetôvé teszi, hogy a különbözô kollekciókban található adatokat a forrás megadása *(pl. Request.Form(„nev”))* nélkül, közvetlenül a Request*(„nev”)* hívással érjük el. Ilyenkor az IIS az összes kollekción végigfut, és visszaadja az adott nevû elem elsô elôfordulását. A sorrend a következô:

- QueryString
- Form
- Cookies
- ClientCertificate
- ServerVariables

Néha hasznos lehet, de általában nem jó, ha ezt a közvetlen hivatkozást használjuk: egyrészt, feleslegesen terheljük vele a kiszolgálót, másrészt pedig, nem lehetünk biztosak abban, hogy az adat onnan jött, ahonnan mi vártuk. Képzeljük el, hogy egy kérdôív „név” mezôjét szeretnénk beolvasni, ehelyett azt kapjuk, amit a felhasználó kézzel begépelt az URL végére!

Az ASP alkalmazás és munkamenet

Lássuk csak újra az elôzô számban bemutatott ábrát az IIS objektumhierarchiájáról:



Jól látható – és remélem, mostanra nyilvánvaló is –, hogy az eddig tárgyalt objektumok, a Request és a Response mindig egy aktuális HTTP kérést és az arra adott választ jelképezik. A következô kérés esetén mindkét objektumból új keletkezik. A következôkben az alkalmazás *(Application)* és a munkamenet *(Session)* objektumokról lesz szó. Ezek az objektumok már nem tûnnek el ilyen egykönnyen az IIS memóriájából, hiszen feladatuk pontosan az, hogy több kérést is átfogó mûveleteket, központi adattárolást tegyenek lehetôvé.

Az ASP alkalmazás

Mit nevezünk ASP alkalmazásnak? Egy ASP alkalmazás tulajdonképpen nem más, mint egy adott könyvtárban és az összes alkönyvtárában található .asp kódok halmaza. Csakhogy a valóságban ennél kicsit bonyolultabb a helyzet, hiszen ha csak errôl lenne szó, nem lett volna szükség az alkalmazások létrehozására.

Mint tudjuk, a HTTP eredetileg állapotmentes világ: jön egy kérés, mi kiszolgáljuk, azután jön a következô, és a kiszolgálónak végülis fogalma sincs arról, hogy melyik kérést éppen ki küldte. Lehet, hogy ugyanaz a felhasználó, lehet, hogy a világ két végérôl két olvasó jelentkezett. Mégis, milyen jó lenne, ha lenne egy globális „valami”, amiben adatokat, sôt, akár kész objektumokat tárolhatunk, és bármikor szükség van rá, csak „fel” kell nyúlnunk érte, és hopp, máris a rendelkezésünkre áll! Nos, pontosan erre való az Application objektum. Segítségével a felhasználók, kérések között adatokat oszthatunk meg egyszerûen, anélkül, hogy például lemezre kellene írnunk azokat. Az Application objektumba írt információ mindaddig megmarad, amíg az adott alkalmazást *(gyakorlatilag az IIS-t)* újra nem indítjuk. Amint az ábrán is látható, az ASP alkalmazás egy nagyszerû globális „felhô” a felhasználók kérései és az azokra adott válaszok fölött. Fontos, hogy Application objektum minden ASP alkalmazásban csak egy van.

Az ASP munkamenet

Ha már így belemelegedtünk a felhôkbe: az ASP munkamenet *(Session)* célja teljesen hasonló, csakhogy ez az Application-nel ellentétben nem felhasználók közötti, hanem egy adott felhasználó mûveletei fölötti globális objektum. Ha úgy tetszik, számos Request és Response fölött uralkodó valami, ami megmarad egészen addig, amig a felhasználó el nem hagyja a webhelyet, vagy be nem csukja a böngészôjét. Természetesen Session objektumból már nem csak egy van: ahány felhasználó, annyi Session.

Hoppá! Nem tûnik fel valami? Az elôbb éppen azt mondtam, hogy a HTTP állapotmentes protokoll. Akkor hogyan tudjuk megkülönböztetni Jenô és Benô különbözô kéréseit Benô két, egymás után küldött kérésétôl? *(Csak semmi trükk az IP címekkel: természetesen Jenô és Benô ugyanazt az IP címet használják, de akár az is elôfordulhat, hogy „menet közben” Benô IP címe megváltozik)*.

Nos, a válasz egyszerû: cookie. Az IIS trükkös kis sütit küld minden felhasználónak, akit azután felismer mindaddig, amig a cookie megmarad *(márpedig az csak addig marad meg, amig a böngészôt be nem zárjuk)*. Nézzük csak meg, mit mond az IIS egy teljesen átlagos kérésre:

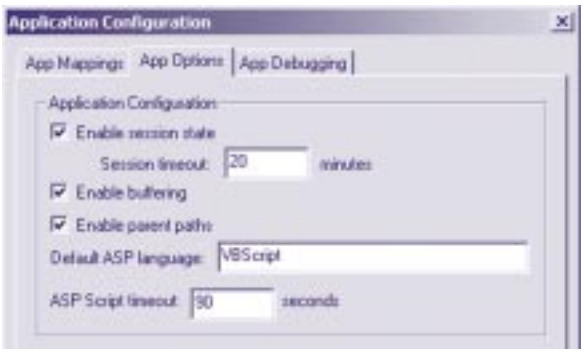
```
GET /default.asp HTTP/1.1
…

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Mon, 12 Feb 2001 22:21:53 GMT
Content-Length: 2354
Content-Type: text/html
Set-Cookie:
↳ ASPSESSIONIDGQGQGVDQ=IBGLAICAJDOELPGDLNDPODPM;
↳ path=/
Cache-control: private
…
```

Kapunk egy érdekes nevû cookie-t: az *(egyébként változó)* ASPSESSIONIDxxxxxxxx nevû cookie tartalmának segítségével az IIS egyértelmûen azonosítja a visszatérô böngészôt, és ugyanabba a környezetbe helyezi *(azaz, mindenki az elsô látogatáskor részére létrehozott, különbejáratú Session objektumba pottyan)*.

Elbúcsúzhatunk a munkamenetektôl, ha a felhasználó böngészôje visszautasítja a cookie-kat. Ha ilyenkor mégis valami hasonló funkcionalitást szeretnénk elérni, külsô gyártó termékeit kell használnunk, amelyek az URL-be ágyazva valósítják meg a munkamenetek kezelését (az IIS-ben szûrô-

ként mûködve minden, az oldalakban található URL hivatkozáshoz hozzáfûzik az azonosítót, míg a böngészôktôl érkezô kérésekbôl kiszûrik azokat). Ez a megoldás teljesen cookie-mentes, és elvileg minden böngészô boldogul vele (csak kicsit rondák lesznek az URL-ek).

Természetesen kell, hogy legyen egy bizonyos idôkorlát is: ha valaki 20 percen belül nem jelentkezik, a részére létrehozott Session objektum elveszik, és legközelebb újra tiszta lappal indul. Ez az idôkorlát is beállítható, mégpedig ugyanott, ahol az elôzô számban alapértelmezett scriptnyelv beállításait megtaláltuk:



A Session idôtúllépése alapértelmezésben 20 perc

Ha nincs rá szükségünk, a munkamenetek kezelését itt egy kattintással letilthatjuk *(hiszen semmi sincs ingyen)*. Ha globálisan ezt nem akarjuk megtenni, akár oldalanként is kikapcsolhatjuk, az oldal tetejére írt ASP direktíva segítségével:

```
<%@ENABLESESSIONSTATE="False"%>
```

A Session objektum

Jó szokásunkhoz híven, haladjunk ismét hátulról elôre: ismerjük meg a Session objektum rejtelmeit. Mindenekelôtt, próbálgassuk a *sessionid.asp* oldalt! Nyissunk meg egy böngészôt, nyissuk meg az oldalt, vándoroljunk tovább, térjünk vissza, és figyeljük meg, változik-e a Session azonosítója! Nyissunk meg egy másik böngészôt, és láthatjuk: ahány böngészô, annyi Session. Az oldal egyetlen említésreméltó sort tartalmaz:

```
Session ID: <% = Session.SessionID %>
```

A **Session.SessionID** jellemzô visszaadja a Session azonosítóját. Ez az azonosító sokszor jó szolgálatot tehet, hiszen segítségével azonosítani lehet a visszatérô felhasználót. *(Mielôtt valaki félreértené: visszatérô alatt most azt értjük, aki a Session idôkorlát lejárta elôtt „visszatér”, vagy azalatt barangol oldalainkon)*.

A **Session.LCID** és a **Session.CodePage** jellemzôk a szokásos locale és karaktertábla-azonosítók; az objektum negyedik, s egyben utolsó jellemzôje pedig a **Session.TimeOut**, amit átállítva egy adott munkamenet erejéig felülbírálhatjuk az alapértelmezett 20 perces idôkorlátot.

Sokkal érdekesebb ennél az, amire a Session objektumot eredetileg kitalálták: írhatunk bele és olvashatunk belôle, gyakorlatilag bármit; a beleírt érték pedig természetesen megmarad mindaddig, amíg az objektum életben van. A *session.asp* bemutatja mindazt, amit a Session objektumról tudni kell. Lássuk, hogyan tárolhatunk el egy értéket, hogy azután késôbb felhasználhassuk:

```
Session("counter") = 1
Session("counter") = Session("counter") + 1
x = Session("counter")
```

A *session.asp* elsô részében a Session*(„counter”)* értékének megfelelôen köszöntjük a látogatót. Érdekes még az oldal alján található adat is: a **Session.Contents** ugyanis egy kollekció, aminek segítségével visszanyerhetjük mindazt, amit sikerült az objektumba belapátolnunk (*anélkül, hogy tudnánk a nevét)*, valahogy így:

```
For Each oItem In Session.Contents
  Response.Write(Session(" & oItem & ") = " &
  ↳ Session(oItem) & "<br>" )
Next
```

Ebbôl a kollekcióból persze törölni is lehet. Az alábbi példa elsô két sora egy-egy elemet *(pl. a „counter” nevût, vagy éppen a másodikat)*, míg a harmadik a teljes tartalmat törli:

```
Session.Contents.Remove("counter")
Session.Contents.Remove(2)
Session.Contents.RemoveAll()
```

A tartalom tehát ilyenkor elveszik, de az objektum megmarad. Azért fontos ezt megemlíteni, mert a Session objektumot önmagát is lehet törölni:

```
Session.Abandon()
```

A **Session.Abandon()** hívás után *(természetesen miután az adott oldalt már végleg kiküldtük)* a Session objektum törlôdik a memóriából. A felhasználó legközelebbi jelentkezésekor új, üres objektum jön majd létre. Érdemes megfigyelni a *session.asp* és az *abandon.asp* kódok viselkedését.

Természetesen nem csak számot és szöveget tárolhatunk a Session-ben, hanem akár komplett objektumokat is. Emlékszünk még az elôzô számban használt ADODB.Stream objektumra, amivel binárisan tudtunk olvasni a lemezrôl és adatot küldeni a böngészô felé? Valahogy így:

```
Set oStream = Server.CreateObject("ADODB.Stream")
oStream.Type = 1 ' adTypeBinary
oStream.Open
oStream.LoadFromFile( Server.MapPath("ms.jpg") )

Response.ContentType = "image/jpeg"
Response.BinaryWrite( oStream.Read )
```

Vágjuk ketté ezt a kódot, és az oStream objektumot használjuk átmeneti tárolóhelynek! A *loadpic.asp* betölti az objektumot a Session-be:

```
<%
  Set oStream =
  ↳ Server.CreateObject("ADODB.Stream")
  oStream.Type = 1 ' adTypeBinary
  oStream.Open
```

```
   oStream.LoadFromFile( Server.MapPath("ms.jpg") )

   Set Session("mypic") = oStream
%>
```

A *showpic.asp* pedig kiolvassa és elküldi a böngészônek:

```
<%
     If IsObject( Session("mypic") ) Then
       Set oStream = Session("mypic")

       Response.ContentType = "image/jpeg"
       Response.BinaryWrite( oStream.Read )
     Else
%>
     <HTML><HEAD></HEAD>
     <BODY>A kép nem található.</BODY>
     </HTML>
<%
     End If
%>
```

A kód elején található ellenôrzés azért kell, mert ha a Session*(„mypic”)* nem tartalmazza még az objektumot *(például mert a loadpic.asp-t még nem futtattuk, vagy Abandon() hívására került sor)*, akkor az értékadás azonnal hibát jelezne. Az IsObject() függvény értéke akkor igaz, ha a neki paraméterként átadott változó tényleg egy objektum – ha nem az, a kép helyett egy egyszerû HTML hibaüzenetet küldünk vissza.
Figyeljük meg, hogy az objektumok kezelésénél – a más típusú változókkal ellentétben – használnunk kell a Set utasítást:

```
Set oStream = Server.CreateObject("ADODB.Stream")
Set Session("mypic") = oStream
Set oMyObj = Session("mypic")
```

A global.asa fájl

Most egy kicsit elôreugrunk az idôben: a global.asa fájl tulajdonképpen az Application objektum leírásánál kellene, hogy elôkerüljön, azonban az a következô hónapra csúszik. A fájl azonban tartalmazhat a Session objektumra vonatkozó részeket is, ezért a tárgyalásától nem tekinthetünk el.
A global.asa fájl egy speciális állomány, amit az ASP alkalmazás gyökérkönyvtárában lehet elhelyezni *(az alapértelmezett ASP alkalmazás gyökérkönyvtára például természetesen az \inetpub\wwwroot)*. A fájl arra való, hogy ebben helyezhessük el a globális objektumokat létrehozó és -eseményeket kezelô kódrészleteket, úgyismint:

- Az Application objektum eseményeit (létrehozását, megsemmisítését) kezelô rutinok *(Application_OnStart és Application_OnEnd)*
- A Session objektumok létrehozását és megsemmisítését kezelô eljárások *(Session_OnStart és Session_OnEnd)*
- Globális objektumok létrehozása az <OBJECT> elem segítségével
- Típuskönyvtárak *(type libraries)* betöltése

A **Session_OnStart** szubrutin értelemszerûen az adott Session létrejöttekor, a **Session_OnEnd** pedig az objektum megsemmisülése elôtt fut le. Ezeket a rutinokat a global.asa fájlba <SCRIPT></SCRIPT> elemek közé kell megírnunk, például:

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
Sub Session_OnStart
  Session("starttime") = Now
  Set Session("oFSO") = Server.CreateObject
  ↳ ("Scripting.FileSystemObject")
End Sub

Sub Session_OnEnd
  Set Session("oFSO") = Nothing
End Sub
</SCRIPT>
```

A fenti példában a Session létrejöttekor beleírjuk a pontos idôt, és létrehozunk egy FileSystemObject objektumot, amit a továbbiak során majd kényelmesen használhatunk, a Session_OnEnd szubrutinban pedig felszabadítjuk az FileSystemObject objektum által lefoglalt memóriaterületet. Objektumokat globálisan is létrehozhatunk, az <OBJECT> elem segítségével, például:

```
<OBJECT RUNAT=Server SCOPE=Session ID="oGlobFSO"
↳ PROGID="Scripting.FileSystemObject">
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
...
</SCRIPT>
```

Az így létrehozott objektumokra azután alkalmazásszerte az .asp oldalakban közvetlen nevükkel hivatkozhatunk:

```
<%
  If oGlobFSO.FileExists(strFileName) Then
    ...
  End If
%>
```

Az <OBJECT> elem SCOPE paramétere határozza meg, hogy az objektum hány példányban jöjjön létre. A paraméter értéke esetünkben természetesen „session”, ami azt jelenti, hogy minden egyes Session létrejöttekor újabb FileSystemObject keletkezik.

A StaticObjects kollekció

Egyetlen kollekció maradt a végére: a **Session.StaticObjects** kollekció a global.asa-ban „session” scope-pal létrehozott objektumokat tartalmazza. A kollekció tartalmát a szokásos módon érhetjük el *(ld. session.asp)*:

```
For Each oItem In Session.StaticObjects
  Response.Write( "Session(" & oItem & ") = " &
  ↳ Session(oItem) & "<br>" )
Next
```

Fülöp Miklós
mick@netacademia.net

A cikkben szereplô URL-ek:
[1] http://technet.netacademia.net/feladatok/asp/2
ASP objektummodell-referencia a weben:
http://msdn.microsoft.com/library/psdk/iisref/vbob74bw.htm