

Programmering med ASP

3

Vi er nu kommet gennem det grundlæggende og er klar til at komme i gang med det interessante: programmeringen. I dette kapitel vil der blive beskrevet nogle grundlæggende programmeringsteknikker som brug af variable og metoder. Derefter vil der blive taget fat på ASP-programmeringen, hvor det vil blive gennemgået, hvordan man laver en HTML-side fra ASP, hvordan man overfører data fra en formular til en ASP-side, og hvordan man bruger nogle af de avancerede komponenter, der er en del af ASP.

Den dynamiske web-side

Den dynamiske web-side består ud over statisk HTML-kode også af ASP-kode. Det betyder, at der skal findes en måde at skelne mellem HTML-kode og ASP-kode. Her har man valgt at bruge `<%` til at angive, at det efterfølgende er ASP-kode, mens `%>` bruges til at angive, at ASP-koden slutter her. Den del af ASP-siden, der står uden for `<% .. %>`, opfattes som HTML. For at web-serveren kan finde ud af, at en fil er en ASP-side og ikke en vanlig

HTML-side, er det vigtigt at give ASP-filer filtypen ASP. Som det blev nævnt tidligere, kan man bruge flere forskellige programmeringssprog sammen med ASP. Selv om ASP-teknologien er den samme, uanset hvilket programmeringssprog man anvender, er web-serveren nødt til at vide, hvilket sprog ASP-siden skal fortolkes som. Derfor er det nødvendigt med en metode til at angive, hvilket sprog man bruger. Til det bruger man en global indstilling. Globale indstillinger gælder for hele den ASP-side, de er en del af. Globale indstillinger bliver beskrevet nærmere senere. Den indstilling, der angiver programmeringssproget, hedder `logisk nok - language`. Da man altid sætter et `@` foran globale variable, skal man skrive følgende for at sætte sproget til JavaScript:

```
<%@Language=JavaScript %>
```

Variable

Som nævnt i kapitel 1 under afsnittet om JavaScript er JavaScript et sprog uden typer. Det betyder, at det som regel ikke er nødvendigt at erklære variable. Der er dog enkelte undtagelser. Det gælder især arrays. Et array er en samling af data – for eksempel en række tal. Hvis man eksempelvis vil have tallene fra 0 til 9 i et array, kan man definere det således:

```
tal = new Array(10);
for (i = 0; i < 10; i++) {
    tal[i] = i;
}
```

Et array erklæres altså ved at sætte en variabel lig med `new Array()`. Tallet i parentes angiver array'ets størrelse. Som det ses i `for`-løkken tilgås de enkelte elementer i array'et ved at skrive navnet på array'et efterfulgt af elementets position i kantede parenteser. Bemærk, at den første position er 0 – det betyder, at et array med størrelsen 10 – som array'et i dette eksempel – har elementer fra position 0 til og med 9.

Metoder

Som det blev beskrevet i kapitel 1, er der i JavaScript ikke forskel på funktioner og procedurer. Det er op til programmeren selv at bestemme, om en metode skal være en funktion eller en procedure – hvis metoden slutter med `return`, er det en funktion, ellers er det en procedure. Ud

over de metoder, som man selv kan lave, har JavaScript en række indbyggede metoder. Det er uden for dennes bogs rammer at gennemgå samtlige metoder – dog vil de mest almindelige blive brugt i løbet af bogen.

For en komplet gennemgang af metoderne i JavaScript henvises til Gamperl og Nefzger: *JavaScript i praksis*, Teknisk Forlag. En af de indbyggede metoder er `sqrt`, som returnerer kvadratroden af et tal.

```
b = sqrt (9);
```

vil altså tildele variabelen `b` kvadratroden af 9.

Udskrift til HTML-siden

Udskrift til HTML-siden er nok den vigtigste funktionalitet i ASP. Der er jo ikke meget ved at kunne lave noget avanceret på web-serveren, hvis man ikke kan få det vist til brugeren. Derfor findes der i ASP flere måder at udskrive til HTML-siden på. Den simpleste er blot at skrive den tekst, der skal udskrives, uden for ASP-koden (uden for `<% . . . %>`). Nedenstående eksempel udskriver således teksten Dette er en test ti gange.

```
<%
    for (i = 0; i < 10; i++) {
%>
```

Dette er en test

```
<%  
  }  
%>
```

Eksemplet viser også, at man sagtens kan blande ASP- og HTML-kode – blot skal man huske, at ASP-koden skal omgives af `<%` og `%>`, mens HTML-koden skal stå udenfor.

Denne fremgangsmåde er imidlertid ikke nok. Vil man udskrive værdien af en variabel, kan man ikke bruge denne fremgangsmåde. Som bekendt er HTML statisk, og det er derfor ikke muligt at udskrive dynamiske oplysninger. Hvis man vil udvide eksemplet fra før, må man således bruge en anden metode. Denne anden metode er `<%=`. `<%=` udskriver værdien af det udtryk eller den variabel, der står umiddelbart efter lighedstegnet. Nedenstående kode viser, hvordan eksemplet fra før kan omskrives.

```
<%  
  for (i = 0; i < 10; i++) {  
%>
```

Dette er en test - i er `<%=i%>`

```
<%  
  }  
%>
```

Denne fremgangsmåde er dog ikke altid hensigtsmæssig. Hvis det eneste, man skal udskrive, er variabler eller ud-

tryk, er fremgangsmåden besværlig. I disse tilfælde kan man i stedet skrive `Response.Write`. Følgende eksempel viser, hvordan man kan udskrive tallene fra et til ti.

```
<%  
for (i = 1; i < 11; i++) {  
  Response.Write(i);  
}  
%>
```

Faktisk bliver `<%=` automatisk erstattet med `Response.Write`, før ASP-siden bliver fortolket. Dette skal man være opmærksom på, fordi det betyder, at et semikolon efter den værdi, der skal udskrives, resulterer i en fejl.

Indsendelse af oplysninger

Som nævnt er formålet med at bruge ASP at tilføje dynamik til sit web-sted. Dette betyder, at brugeren skal have mulighed for at påvirke udseendet af web-stedet. Den mest udbredte måde at gøre dette på, er at lade ham udfylde en formular. I dette afsnit vil det blive beskrevet, hvordan man gør det. Afsnittet bliver afsluttet med en gennemgang af andre måder at lade brugeren angive oplysninger på.

Formularer i HTML

I ASP spiller formularer en meget vigtig rolle, derfor vil det i dette afsnit blive beskrevet, hvordan man laver formularer i HTML.

I HTML bruger man kommandoen `<form>` til at angive starten på en formular, og `</form>` til at angive slutningen af en formular. `<form>`-kommandoen har følgende syntaks:

```
<form name=navn action=ASP-side method=metode>
```

Navn angiver formularens navn og er ikke strengt nødvendigt. Bruger man JavaScript eller VBScript på klienten, er det nødvendigt at give formularen et navn for at kunne tilgå formen og dens elementer, men det er ikke nødvendigt med ASP. *ASP-side* er adressen på den side, der skal behandle de indsendte oplysninger. Hvordan denne side skal laves, bliver beskrevet senere. *Metode* er den måde, dataene skal indsendes på – de forskellige indsendelsesmetoder bliver ligeledes beskrevet senere.

Formularen kan indeholde en række forskellige felter, som brugeren kan udfylde. Den mest udbredte feltype er tekstfeltet, som giver brugeren mulighed for at indtaste noget tekst. Nedenstående eksempel viser, hvordan man laver et tekstfelt.

```
<input type=text name=brugernavn>
```

Eksemplet laver et tekstfelt, der har navnet brugernavn. Feltets navn får betydning, når ASP-siden skal laves. Man kan også lave tekstfelter, der er specialiserede til at indtaste adgangskoder i. For at lave et af disse skal man ændre typen fra `text` til `password`. Følgende eksempel viser, hvordan man kan lave et felt, hvor brugeren kan indtaste sin

adgangskode. Uanset hvilke tegn brugeren indtaster, vil de blive vist som stjerner.

```
<input type=password name=adgangskode>
```

Det er imidlertid ikke altid hensigtsmæssigt at lade brugeren indtaste alting. Hvis et spørgsmål kan besvares med ja eller nej, eller hvis svarmulighederne er begrænsede, er det bedre, at lade brugeren vælge mellem mulighederne. Hvis et spørgsmål kan besvares med ja eller nej, kan man bruge et afkrydsningsfelt. Man laver et afkrydsningsfelt ved at sætte `input-type` til `checkbox`. Nedenstående eksempel viser hvordan.

```
<input type=checkbox name=gem> Gem adgangskode
```

Eksemplet viser et afkrydsningsfelt efterfulgt af teksten Gem adgangskode. Hvis man vil have, at afkrydsningsfeltet som standard er afkrydset, kan man tilføje ordet `checked` i `input-kommandoen`. Følgende eksempel laver et afkrydsningsfelt, der som standard er afkrydset.

```
<input type=checkbox name=gem checked> Gem adgangskode
```

Der kan også være situationer, hvor man skal lade brugeren vælge en af en række muligheder. I disse tilfælde skal man bruge en alternativknop. I HTML laves alternativknapper ved at sætte `input-typen` til `radio`. Det interessante ved alternativknapper er, at det kun er én af en gruppe, der kan være markeret på samme tid. Man angi-

ver, hvilke knapper der hører sammen i en gruppe, ved at give dem det samme navn. For at man kan kende forskel på dem, kan man give dem forskellige værdier. Det gøres ved at bruge `value`-værdien. Ligesom man på et afkrydningsfelt kan angive, at det skal være afkrydset, kan man også vælge, at en af alternativknapperne skal være markeret. Det gøres ligeledes med ordet `checked`.

Hvilke rettigheder, vil du logges på med?

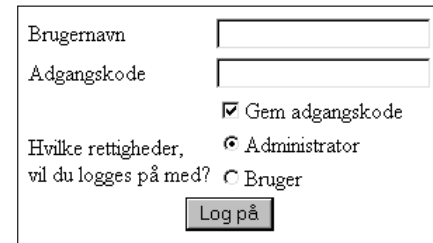
```
<input type=radio name=rettigheder  
value=administrator>Administrator  
<input type=radio name=rettigheder  
value=bruger checked>Bruger
```

For at indsende formularen skal man bruge en knap. Denne knap skal have `input`-typen `submit`. Knappens `value`-værdi angiver den tekst, der skal vises på knappen.

```
<input type=submit value="Log på">
```

For at oprette en formular skal man altså skrive den HTML-kode, der er vist på næste side (programliste 3.1). Bemærk, at elementerne er sat ind i en tabel for at gøre opstillingen pænere. De HTML-kommandoer, der udgør formularen, er fremhævet med fed skrift.

Eksemplet giver en HTML-formular, der ser ud som i figur 3.1.



Figur 3.1 HTML-formular til adgangskontrol.

Indsendelsesmetoder

Oplysninger, som overføres til ASP-sider, sendes via HTTP-protokollen, der beskriver, hvordan oplysninger skal sendes fra browsere til web-servere og tilbage. HTTP-protokollen beskriver flere forskellige måder at indsende data på – de mest udbredte er `get` og `post`, som begge vil blive beskrevet i dette afsnit.

Når oplysninger indsendes med `get`-metoden, bliver de føjet til adressen på det dokument, de bliver indsendt til. Dokumentets adresse og oplysninger adskilles med et spørgsmålstegn. Indsender man eksempelvis oplysningen `farve=gul` til siden `mailing.asp`, vil adressen komme til at se således ud:

```
mailing.asp?farve=gul
```

Hvis man indsender mere end én oplysning, adskiller man dem med et `&`-tegn. Det betyder, at en adresse kan komme til at se således ud:

```

<form name=login action=login.asp method=get>
<table>
<tr>
<td>Brugernavn</td>
<td><input type=text name=brugernavn></td>
</tr><tr>
<td>Adgangskode</td>
<td><input type=password name=adgangskode></td>
</tr><tr>
<td>&nbsp;</td>
<td><input type=checkbox name=gem checked>Gem adgangskode</td>
</tr><tr>
<td rowspan=2>Hvilke rettigheder,<br>vil du logges på med?</td>
<td><input type=radio name=rettigheder value=administrator>Administrator</td>
</tr><tr>
<td><input type=radio name=rettigheder value=bruger checked>Bruger</td>
</tr><tr>
<td colspan=2 align=center><input type=submit value="Log på"></td>
</tr>
</table>
</form>

```

Programliste 3.1 *login.html*.

mailing.asp?farve=gul&blankhed=2

Get-metoden har den fordel, at den er forholdsvis let at benytte, og at det er let at se, hvilke data der bliver indsendt. Den har imidlertid også en række ulemper. Eksem-

pelvis har de fleste browsere en maksimal længde af adressefeltet – typisk på omkring 200 tegn. Det betyder, at man ikke kan indsende oplysninger, der fylder mere end 200 tegn. En anden ulempe ved get-metoden er, at man kan se oplysningerne i adressefeltet. Det betyder for ek-

sempel, at en brugers adgangskode vil blive vist, hvis den indsendes med get.

Post-metoden har ikke disse ulemper. Her bliver oplysningerne sendt i selve forespørgslen og ikke som en del af adressen. Det betyder, at de ikke er synlige for brugeren. Post-metoden har heller ingen begrænsninger med hensyn til datastørrelse og kan derfor bruges ved indsendelse af store datamængder.

Dette kunne umiddelbart tyde på, at man altid skal benytte post-metoden, men get-metoden giver nogle muligheder, som man ikke har med post. For eksempel kan man have brug for at angive oplysninger i et link i en HTML-side. Dette kan man ikke med post, men med get er det muligt, fordi man kan skrive oplysningerne i adressen.

Vil man lave en henvisning til en side, der anvender get, kan man blot i sin HTML-kode skrive:

```
<a href="mailing.asp?farve=gul">
```

Bruger man derimod post, findes der ingen måde, hvorpå man i en henvisning kan angive parametrene.

Udover disse praktiske grunde til at vælge metode, findes der nogle anvisninger, man bør følge. Man bør kun bruge get, når indsendelsen kan foretages flere gange med det samme resultat. Det kan eksempelvis være ved en søgning, hvor søgning efter det samme ord giver det samme resultat

hver gang. Kan indsendelsen derimod ikke gentages med det samme resultat, bør man bruge post. Det kan for eksempel være ved tilmelding, hvor gentagen indsendelse vil medføre, at man bliver tilmeldt flere gange. Selv om dette naturligvis kun er retningslinier for, hvilken metode man bør vælge, er det en god ide at overholde dem, da de fleste browsere forventer, at de bliver overholdt.

Indsendelse med get-metoden

Når man klikker på knappen Log på, skifter browserens adressefelt til adressen på den side, der skal behandle formularens data. I dette eksempel er det `login.asp`. Fordi eksemplet bruger indsendelsesmetoden get, vil parametrene også blive vist i adressen. Det betyder, at adressefeltet kan komme til at se således ud:

```
login.asp?brugernavn=kh&adgangskode=sesam&gem=on&rettigheder=bruger
```

Der er ikke meget ved at kunne indsende oplysninger, hvis man ikke kan behandle dem, når man modtager dem. Det skal der en ASP-side til.

For at hente parametre ud fra den forespørgsel, browseren sendte, bruger man `Request`-objektet. Dette objekt har en metode, der hedder `QueryString`. Denne metode bruges til at pille parametre ud af forespørgslens *query string* – det er den del, der står efter spørgsmålstegnet i adresselinien.

Hvis man vil have værdien af parameteren brugernavn fra eksemplet, kan man altså skrive

```
brugernavn = Request.QueryString("brugernavn");
```

Hvis man vil lave et simpelt eksempel, der blot udskriver de oplysninger, der bliver indsendt, kan man altså udskrive brugernavnet og adgangskoden med følgende linier:

```
<%=Request.QueryString("brugernavn")%>  
<%=Request.QueryString("adgangskode")%>
```

Det er straks lidt mere kompliceret at behandle værdien af feltet Gem adgangskode. Denne værdi bliver nemlig kun sendt med, hvis den er markeret. Hvis man forsøger at få værdien på en parameter, der ikke findes, får man null tilbage. Det vil sige, at man – ved at kontrollere om Request.QueryString ("gem") returnerer null – kan se, om parameteren er sat. Det betyder, at man kan bruge følgende linier til at kontrollere og udskrive, om adgangskoden skal gemmes.

```
<%  
    if (Request.QueryString("gem") != null) {  
        Response.Write("Ja");  
    } else {  
        Response.Write("Nej");  
    }  
%>
```

Bemærk, at der her bruges Response.Write til at udskrive til HTML-siden i stedet for <%=.

De to alternativknapper, der bruges til at angive rettigheder, er også lidt komplicerede at behandle. Her er det igen nødvendigt at kontrollere værdien af parameteren rettigheder, for at man kan se, hvilke rettigheder der er valgt. Det gør nedenstående kode.

```
<%  
    if (Request.QueryString("bruger")) {  
        Response.Write("Brugerrettigheder")  
    } else {  
        Response.Write("Administratorrettigheder");  
    }  
%>
```

For at det kommer til at se pænt ud, skal udskrifterne naturligvis integreres i HTML-kode. Det kan man gøre på følgende måde – igen er ASP-koden fremhævet med fed skrift:

```
<% @Language = JavaScript %>  
<HTML>  
<head>  
    <title>Eksempel</title>  
</head>  
<body>
```

```

<b>Følgende oplysninger blev indsendt:</b><br>
<table>
<tr>
<td>Brugernavn</td>
<td><%=Request.QueryString("brugernavn")%></td>
</tr><tr>
<td>Adgangskode</td>
<td><%=Request.QueryString("adgangskode")%></td>
</tr><tr>
<td>Gem password</td>
<td>
<%
    if (Request.QueryString("gem") != null) {
        Response.Write("Ja");
    } else {
        Response.Write("Nej");
    }
%>
</td>
</tr><tr>
<td>Rettigheder</td>
<td>
<%
    if (Request.QueryString("bruger")) {
        Response.Write("Brugerrettigheder");
    } else {
        Response.Write("Administratorrettigheder");
    }
%>
</td>

```

```

</tr>
</table>
</body>
</HTML>

```

Programliste 3.2 *login.asp*

Indsendelse med post-metoden

Som nævnt tidligere har get-metoden en række problemer – det sås også i det forrige eksempel, hvor adgangskoden fremgik af browserens adressefelt. Derfor vil det i dette afsnit blive gennemgået, hvordan man indsender data med post-metoden.

Det første sted, man skal ændre, er HTML-formularen. Her skal erklæringen af formularen ændres, så metoden sættes til post i stedet for get. I eksemplet fra før vil det betyde, at formularen skal erklæres således:

```
<form name=login action=login.asp method=post>
```

Der skal ikke foretages ændringer i resten af HTML-formularen.

Der skal også foretages ændringer i ASP-siden. Som det blev nævnt i forrige afsnit, bruges `Request.QueryString` til at få oplysninger om den del af adressen, der står efter spørgsmålstegnet. Ved post-metoden findes parametrene slet ikke i adresselinien, så derfor kan denne fremgangs-

måde naturligvis ikke bruges. I stedet skal man bruge den metode, der hedder `Request.Form`. Ligesom tidligere angiver man navnet på parameteren som argument til metoden. Det betyder, at ASP-siden fra get-eksemplet kommer til at se således ud, hvis man bruger post som indsendelsesmetode. Forskellene fra get-metoden til post-metoden er fremhævet med fed skrift.

```
<% @Language = JavaScript %>
<HTML>
<head>
  <title>Eksempel</title>
</head>
<body>
<b>Følgende oplysninger blev indsendt:</b><br>
<table>
<tr>
<td>Brugernavn</td>
<td><b>%Request.Form("brugernavn")%</b></td>
</tr><tr>
<td>Adgangskode</td>
<td><b>%Request.Form("adgangskode")%</b></td>
</tr><tr>
<td>Gem password</td>
<td>
<%
  if (Request.Form("gem") != null) {
    Response.Write("Ja");
  } else {
```

```
    Response.Write("Nej");
  }
  %>
</td>
</tr><tr>
<td>Rettigheder</td>
<td>
<%
  if (Request.Form("bruger")) {
    Response.Write("Brugerrettigheder")
  } else {
    Response.Write("Administratorrettigheder");
  }
  %>
</td>
</tr>
</table>
</body>
</HTML>
```

Programliste 3.3 *login-post.asp*.

Som det ses, er hovedparten af ASP-koden den samme. Det er udelukkende de steder, hvor oplysningerne fra formularen skal bruges, der er ændringer. Eksemplet viser også, at reglerne for, hvilke oplysninger der bliver indsendt fra formularen, er de samme, ligegyldigt om man bruger get eller post som indsendelsesmetode.

Scripts og applikationer

Det er nu blevet gennemgået, hvordan man kan bruge forskellige af ASPs muligheder med scriptsprog. Det er imidlertid ikke alt, der kan lade sig gøre fra et scriptsprog. Vil man for eksempel lave avancerede ting, der kræver adgang til filer eller andre ressourcer på computeren, kommer scriptsprogene til kort – de fleste scriptsprog er så generelle, at de ikke har funktionalitet til disse områder, der afhænger meget af den computer, de bliver afviklet på, og af det program der fortolker dem.

Da det ville være fuldstændigt uacceptabelt ikke at kunne gøre disse ting fra ASP, har man gjort det muligt at integrere ActiveX-komponenter i ASP. Det betyder, at man kan skrive en ActiveX-komponent i et mere avanceret programmeringssprog (som for eksempel C++) og derefter benytte komponentens funktionalitet fra et scriptsprog.

Det skal bemærkes, at brugen af ActiveX-komponenter i denne sammenhæng ikke gør web-stedet browserafhængigt. Selv om det kun er Microsoft Internet Explorer, der kan afvikle ActiveX-komponenter, har det ikke betydning for ASP-sider med ActiveX-komponenter, da komponenterne bliver afviklet på serveren, og således slet ikke ses af de besøgendes browsere.

I forhold til at lave scripts med et scriptsprog er det forholdsvis kompliceret at lave ActiveX-komponenter, og en beskrivelse af udviklingen af ActiveX-komponenter ligger uden for rammerne af denne bog. Heldigvis er det ikke

nødvendigt selv at udvikle komponenterne – der er en del inkluderet i Internet Information Server, og derudover kan man finde en del på Internet. Nogle af disse er gratis, mens andre er kommercielle.

I resten af dette kapitel vil nogle af de indbyggede komponenter blive beskrevet. Det drejer sig om komponenter til at vise tilfældigt indhold, reklamer, indholdsfortegnelser og lignende. Også i de følgende kapitler vil der blive beskrevet ActiveX-komponenter. I kapitel 4 bliver det således beskrevet, hvordan man kan integrere egne ActiveX-komponenter eller ActiveX-komponenter fra tredjepart i ASP. Når databasetilgang bliver gennemgået i kapitel 5, er det også ved hjælp af ActiveX-komponenter.

Det er ikke alle de indbyggede komponenter, der bliver gennemgået – det er der simpelthen ikke plads til. Til gengæld er der en beskrivelse af alle komponenterne i kapitlerne 7, 8 og 9.

Når man skal bruge en ActiveX-komponent i ASP, skal man lave et objekt ud af den. Det gøres ved at bruge metoden `CreateObject`. Denne metode findes i `Server`-objektet, og som parameter skal man angive navnet på den ActiveX-komponent, man skal bruge. Hvis man således vil bruge en ActiveX-komponent, der hedder `activex.komponent` skal man altså skrive

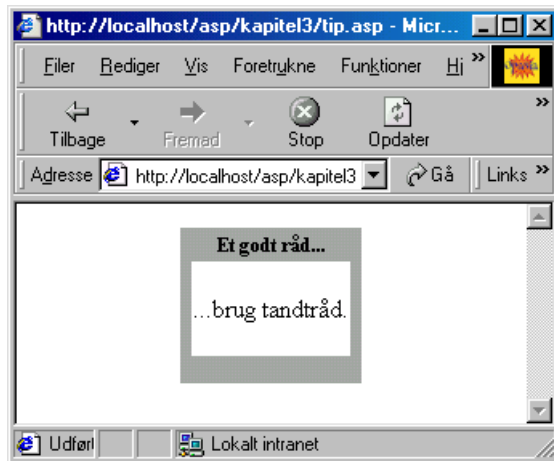
```
Server.CreateObject("activex.komponent");
```

Tilfældigt indhold

Det kendes allerede fra Microsoft Windows – de små tips, der bliver vist, når computeren startes. Det tip, der bliver vist, er blot ét af en større mængde af tips. Computeren vælger selv, hvilket tip der skal vises.

I ASP er den samme teknik til rådighed. Her skal man bruge den komponent, der hedder *content rotator*. Content rotator fungerer på den måde, at den indlæser en fil, der indeholder alle tippene, og vælger et tilfældigt. Som standard er content rotator-komponenten installeret under

```
<%@ language=JavaScript %>
<center>
<table cellspacing=0>
<tr>
<td width=5 bgcolor=#aaaaaa><font size = -1>&nbsp;</td>
<td align = center bgcolor=#aaaaaa><font size = -1> <b>Et godt råd...</b></font></td>
<td width=5 bgcolor=#aaaaaa>&nbsp;</td>
</tr>
<tr>
<td width=5 bgcolor=#aaaaaa><font size = -1>&nbsp;</td>
<td bgcolor=#ffffff>
<br>
<%
    tip = Server.CreateObject("IISSample.ContentRotator");
    dagens_tip = tip.ChooseContent("/data/tips.txt");
    Response.write("..." + dagens_tip);
%>
<br>
<br>
</td>
<td width=5 bgcolor=#aaaaaa>&nbsp;</td>
```

Figur 3.2 Et dokument med et tilfældigt tip.

Oplysninger om browseren

Et af de store problemer ved web-design er, at det ikke er alle browsere, der kan det samme. Ikke alene er der forskel på den måde, de to mest udbredte browsere (Microsoft Internet Explorer og Netscape Navigator) fungerer – de findes også i forskellige versioner. Hvis man vil bruge nogle af de mere avancerede web-teknikker – som for eksempel rammer, klient-scripts og Java-appletter – risikerer man, at ens web-sted ikke er tilgængeligt for alle. Vælger man i stedet at bygge sin side, så den er tilgængelig for alle, medfører det, at siden bliver kedelig, og at de, der har nye browsere, ikke får udnyttet disse muligheder fuldt ud.

Løsningen på dette problem er at registrere, hvilken browser den besøgende har, og stille en side til rådighed, der passer til browseren. Når man bruger ASP, er det simpelt at få oplysninger om browseren og dens egenskaber. Man skal blot benytte den komponent, som hedder `MSWC.BrowserType`. Man opretter et browsertype-objekt med `CreateObject`-metoden. Objektet har en række variable, som indeholder oplysninger om browseren og dens egenskaber.

Følgende eksempel viser nogle af de oplysninger, man kan få fra browsertype.

```
<%@ language=JavaScript %>

<%
    browser = Server.CreateObject("MSWC.BrowserType");
%>

<center>Browser:
    <%=browser.browser%>&nbsp;&lt;%=browser.version%><br>
</center>
<center><table border=1>
<tr>
<td>Rammer</td>
<td>
<%
        if (browser.frames) {
            Response.Write("Ja");
        }
    %>
    </td>
</tr>
</table>
</center>
```

```

    } else {
        Response.Write("Nej");
    }
%>
</td>
</tr>
<tr>
<td>Tabeller</td>
<td>
<%
    if (browser.tables) {
        Response.Write("Ja");
    } else {
        Response.Write("Nej");
    }
%>
</td>
</tr>
<tr>
<td>Java</td>
<td>
<%
    if (browser.javaapplets) {
        Response.Write("Ja");
    } else {
        Response.Write("Nej");
    }
%>
</td>
</tr>

```

```

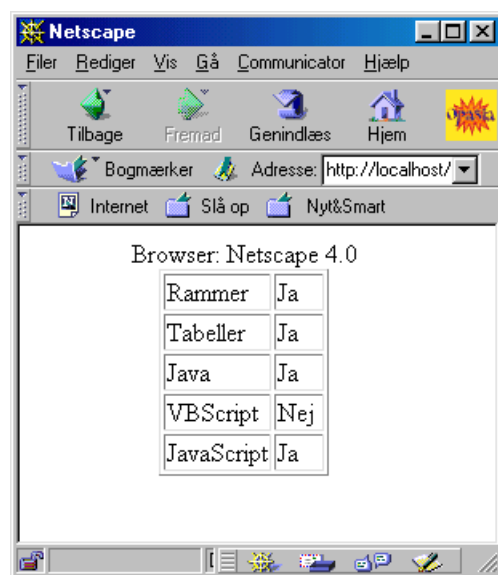
<tr>
<td>VBScript</td>
<td>
<%
    if (browser.vbscript) {
        Response.Write("Ja");
    } else {
        Response.Write("Nej");
    }
%>
</td>
</tr>
<tr>
<td>JavaScript</td>
<td>
<%
    if (browser.javascript) {
        Response.Write("Ja");
    } else {
        Response.Write("Nej");
    }
%>
</td>
</tr>
</table>
</center>

```

Programliste 3.5 *browser.asp*.

Tabel 3.1 *Variabler i BrowserType-objektet.*

Variabel	Betydning
Jscript	Understøtter browseren JScript?
Vbscript	Understøtter browseren VBScript?
Javaapplets	Understøtter browseren Java-appletter?
Frames	Understøtter browseren rammer?
Tables	Understøtter browseren tabeller?
BackgroundSounds	Understøtter browseren baggrundsmusik?



Figur 3.3. *Dokument med oplysninger om Netscape Navigator.*

Figur 3.3 viser ASP-programmet afviklet i Netscape Navigator.

Der findes en række variabler på browsertype-objektet. Disse kan ses i tabel 3.1.

Der er imidlertid ikke meget ved at kunne fortælle brugeren, hvilken browser han har – det ved vedkommende sikkert allerede. Det bliver først interessant, når ASP-siderne tager hensyn til brugerens browser. Følgende eksempel viser, hvordan man kan præsentere information på forskellige måder alt efter, om browseren understøtter tabeller.

```
<%@Language = JavaScript %>
<%
browser = Server.CreateObject("MSWC.BrowserType");
tables = browser.tables;
```



```

    }
%>

Borgergade 99

<%
    if (tables) {
%>

</td>
</tr>
</table>

<%
    }
%>

</body>
</HTML>

```

Programliste 3.6 *tabel.asp*.

Dynamisk indholdsfortegnelse

Hvis man har et omfattende web-sted, er det en god ide at give de besøgende en oversigt over web-stedet. Det kan man gøre ved en slags indholdsfortegnelse. Det er imidlertid en stor opgave at vedligeholde en indholdsfortegnelse over et stort web-sted. Derfor kan man bruge komponenten *Content Linking*. Denne komponent holder styr på alle

de sider, web-stedet består af. Til hver web-side er der tilknyttet en URL (adressen på web-siden) og en beskrivelse af web-siden. Oplysningerne om siderne i indholdsfortegnelsen får komponenten fra en tekstfil – mere om den senere. Fra programmet er det muligt at bede komponenten om næste og forrige URL og næste og forrige beskrivelse. Man kan også få at vide, hvor mange sider der er registreret i indholdsfortegnelsen, ligesom man kan få positionen på den aktuelle side. Tabel 3.2 viser, hvilke metoder Content linking-komponenten har.

Når man bruger en af de metoder, der henter oplysninger om siden på en bestemt position, er det vigtigt at vide, at den første side i listen har position 1 (og ikke position 0, som man måske kunne forvente).

Opbygning af indholdsfilen

For at Content linking-komponenten kan forstå indholdsfortegnelsen, skal den fil, oplysningerne ligger i, have en bestemt opbygning. Hver side skal beskrives på en linie for sig. Først skal adressen på siden skrives. Dette kan blot være navnet på den fil, der indeholder siden. Efter sidens navn skal der være et tabulator tegn (det *skal* være et tabulator tegn – det virker ikke, hvis man bruger mellemrum). Herefter kan beskrivelsen af siden skrives. Sætter man endnu et tabulator tegn, kan man også tilføje en kommentar til linien – brugeren ser ikke denne kommentar, og den er ikke nødvendig, men kan være rar at have, hvis indholdsfortegnelsen indeholder mange sider.

Tabel 3.2 *Metoder i ContentLinking-objektet.*

Metode	Beskrivelse
GetListCount(filnavn)	Returnerer antallet af sider i listen.
GetListIndex(filnavn)	Returnerer positionen af den aktuelle side i listen.
GetNextURL(filnavn)	Returnerer adressen på den næste side i listen.
GetNextDescription(filnavn)	Returnerer beskrivelsen af den næste side i listen.
GetPreviousURL(filnavn)	Returnerer adressen på den forrige side i listen.
GetPreviousDescription(filnavn)	Returnerer beskrivelsen af den forrige side i listen.
GetNthURL(filnavn, n)	Returnerer adressen på siden på position <i>n</i> i listen.
GetNthDescription(filnavn, n)	Returnerer beskrivelsen af siden på position <i>n</i> i listen.

Formatet af filen er altså således

sidenavn beskrivelse *kommentarer*

Følgende eksempel viser, hvordan en fil med oplysninger om en indholdsfortegnelse kan opbygges.

```
indeks.html Velkomstsiden Her starter de besøgende
priser.html Prislister over alle vores varer
nyepriser.html Priser på vores nyeste varer
tilbud.html Ugens tilbud Husk at opdatere hver uge!
```

Om filnavne

Som nævnt tidligere skal man angive navnet på indholdsfilen i metodekaldene i Content linking-objektet. Der er

dog nogle ting, man skal vide om filnavne i forbindelse med ASP.

I ASP opererer man med to forskellige slags filnavne: *fysiske* filnavne og *virtuelle* filnavne. Et fysisk filnavn er det navn, en fil har i virkeligheden – det vil være det navn, man for eksempel kan finde filen under i Windows Stifinder. De virtuelle navne er de navne, filerne har i web-verdenen. Det er disse navne, man bruger for at finde filerne i sin web-browser.

Hvis en fil hedder `priser.html` og ligger i biblioteket `c:\web\oplysninger`, vil dens fysiske navn være `c:\web\oplysninger\priser.html`. Hvis biblioteket `c:\web\oplysninger` på web-serveren er sat op til at svare

```
<% @Language=JavaScript %>

<HTML>
<body>
<h1>Indhold</h1>

<%
    content = Server.CreateObject("MSWC.NextLink");
    for (i = 1; i <= content.GetListCount("data/indhold.txt"); i++) {
        Response.write("<A HREF = \"");
        Response.write(content.getNthURL("data/indhold.txt", i));
        Response.write("\">");
        Response.write(content.getNthDescription("data/indhold.txt", i));
        Response.write("</A><BR> ");
    }
%>

</body>
</HTML>
```

Programliste 3.7 *indhold.asp*.

til stien /oplysninger, vil filens virtuelle navn være /oplysninger/priser.html.

Derudover kan man skelne mellem absolutte og relative filnavne. Et absolut filnavn angiver et filnavn, som entydigt udpeger en fil – det kan eksempelvis være

c:\web\oplysninger\priser.html. Her kan der ikke være tvivl om, hvilken fil der angives.

Bruger man i stedet et relativt filnavn, kan man ikke direkte se, hvilken fil der angives. Det kommer nemlig an på, hvilket bibliotek der er det aktuelle. Hvis det aktuelle bib-

liotek er `c:\web\oplysninger`, og man angiver navnet `priser.html`, vil det være `c:\web\oplysninger\priser.html`, der er tale om. Er det aktuelle bibliotek i stedet `c:\web`, og man angiver `priser.html`, vil det være filen `c:\web\priser.html`, der angives. Man kan angive filer i andre biblioteker end det aktuelle ved at angive biblioteksnavnet foran filnavnet – eksempelvis `data/oplysninger.html`, der angiver filen `oplysninger.html` i biblioteket `data`, der er et underbibliotek til det aktuelle bibliotek.

De fleste steder må man selv vælge, hvordan man vil angive, hvilken fil der skal bruges, men når man skal angive adresser, kan der være restriktioner. I den fil, der indeholder oplysninger om indholdsfortegnelsen, må man således kun angive virtuelle eller relative adresser.

Generering af indholdsfortegnelse

For at få dataene i indholdsfilen ændret til HTML-kode, som brugeren kan se, skal man først lave et objekt af Content linking-komponenten. Komponentens hedder `MSWC.NextLink`, og man laver et objekt af den med kommandoen `Server.CreateObject("MSWC.NextLink");`

Programliste 3.7 viser, hvordan man kan lave en indholdsfortegnelse med Content linking-komponenten.

Viderestilling

Selv om brugeren har valgt at se en bestemt side, kan det godt være, at det i virkeligheden er en anden, brugeren

skal se. Det kan for eksempel gælde, hvis den side, brugeren forsøger at hente, er blevet flyttet, eller hvis brugeren ikke opfylder nogle betingelser. Man kan eksempelvis kontrollere, at brugeren har indtastet sit kodeord rigtigt, og hvis ikke sende vedkommende videre til en fejlside. Man kan også bruge teknikken til at sikre, at brugeren starter på den rigtige side på web-stedet.

Viderestilling – eller omdirigering – sker ved hjælp af metoden `Redirect`, der findes i `Response`-objektet. Som parameter til denne metode giver man blot adressen på den side, brugeren skal sendes videre til – det behøver ikke engang at være en side på ens eget web-sted.

Muligheden for at dirigere brugeren videre til en side på et andet web-sted, kan blandt andet bruges til at lave statistik. Hvis man eksempelvis har reklamer på sine sider, kan man lave et ASP-dokument, der registrerer, at der er blevet klikket på en reklame, hvorefter brugeren sendes til annoncørens side. Følgende eksempel viser hvordan.

```
<% @Language=JavaScript %>
<%
    ad = Request.QueryString("ad");

    // Registrer, at der er blevet klikket på annoncen.
    // Hvordan man kan gøre dette gennemgås senere.

    Response.Redirect(Request.QueryString("url"));
%>
```

Eksemplet skal have to parametre: `ad`, der angiver, hvilken annonce der er blevet klikket på, og `url`, der angiver, hvilken side brugeren skal sendes til. Hvis man gemmer filen som `Redirect.asp`, kan man bruge den med følgende link:

```
redirect.asp?ad=1&url=http://www.tekniskforlag.dk
```

Dette link vil registrere, at der er klikket på annonce nummer 1 og derefter sende brugeren videre til `www.tekniskforlag.dk`.

Ovenstående eksempel er imidlertid forholdsvis simpelt. Det viser nemlig ikke en af de ting, man skal være opmærksom på, når man bruger `Redirect`-metoden. Man kan ikke foretage en viderestilling, hvis man allerede har skrevet noget af HTML-siden (brugt `<%=` eller `Response.Write`). Selv om man som hovedregel således skal kontrollere, om brugeren skal viderestilles, før man begynder at lave HTML-kode, kan man omgå reglen. Man kan nemlig slette det, man tidligere har skrevet. Det gør man ved at bruge metoden `Response.Clear`. For at kunne bruge `Clear`-metoden skal man bruge *buffering*. Buffering bliver gennemgået i næste afsnit – på nuværende tidspunkt er det nok at vide, at man slår buffering til ved at skrive

```
Response.Buffer = true;
```

Følgende eksempel viser, hvordan man kan bruge `Clear`-metoden til at viderestille, selv om man allerede har skrevet noget af HTML-siden.

```
<% @Language = JavaScript %>

<%
    Response.Buffer = true;
%>

<HTML>
<body>
Beregninger med de tal, du indtastede:<br>

<%
    tal1 = Request.QueryString("tal1");
    tal2 = Request.QueryString("tal2");
%>
<%=tal1%> + <%=tal2%> = <%=tal1+tal2%><br>
<%=tal1%> - <%=tal2%> = <%=tal1-tal2%><br>
<%=tal1%> * <%=tal2%> = <%=tal1*tal2%><br>

<%
    if (tal2 == 0) {
        Response.Clear();
        Response.Redirect("error.asp");
    }
%>

<%=tal1%> : <%=tal2%> = <%=tal1/tal2%><br>
</body>
</HTML>
```

Eksemplet arbejder på to tal, der gives som parametre. Der vises en række beregninger med de to tal. For at sikre, at divisionen går godt, er man nødt til at kontrollere, at det tal, der divideres med, ikke er 0. Er tallet 0, bliver brugeren sendt videre til en fejlside.

Buffering

Buffering giver ASP-programmøren mulighed for at kontrollere, hvornår brugeren modtager den side, ASP-dokumentet genererer. Når buffering er slået til, bliver den genererede side opbevaret på serveren, indtil hele dokumentet er genereret. Er buffering ikke slået til – hvilket er standardindstillingen – bliver HTML-siden sendt til brugeren, efterhånden som den bliver genereret. Bemærk dog at buffering i ASP 3.0 er slået til som standard.

Det ville ikke umiddelbart være interessant, hvis ikke der havde været metoder, som kan bruges til at sende en del af siden på et bestemt tidspunkt. Disse metoder er `Flush` og `End`. `Flush` bruges til at sende den del af HTML-dokumentet, der er skrevet på det tidspunkt, metoden kaldes, mens `End` bruges til at sende HTML-dokumentet og samtidig indikere, at dokumentet er færdigt. Når ASP-programmet kommer til et `End`-metodekald, stopper det – det vil sige, at eventuelle linier derefter ikke vil blive udført.

I forbindelse med buffering findes også metoden `Clear`. `Clear`-metoden sletter den del af siden, der er skrevet, men endnu ikke sendt til brugeren. `Clear`-metoden er beskrevet tidligere i dette kapitel.

Fælles for `Flush`, `End` og `Clear` er, at de kun kan bruges, hvis buffering er slået til. Man slår buffering til ved at skrive `Response.Buffer = true`. Vil man slå buffering fra, skriver man `Response.Buffer = false`.

Nedenstående eksempel viser, hvordan buffering kan benyttes.

```
<% @Language=JavaScript %>

<%
    Response.buffer=true;
%>
<HTML>
<body>
Se, hvor hurtigt jeg kan tælle til en million:<br>
<%
    for (i = 0; i <= 1000000; i++) {
        if(i % 10000 == 0) {
            Response.write(i);
            Response.write("<BR>");
        }
        if (i % 100000 == 0) {
            Response.flush();
        }
    }
%>
```

```
</body>  
</HTML>
```

Programliste 3.9 *countbuffer.asp*.

Eksemplet tæller fra nul til en million og udskriver hele titusinder. Hver gang der er talt hundredtusind op, sendes udskriften til browseren, hvilket vil sige, at brugeren vil få præsenteret ti tal ad gangen.

Fjernede man linierne `Response.buffer=true` og `Response.flush()`, ville brugeren se tallene et efter et.