Høgskolen i Østfold

HTML

Et utdrag fra: http://www.ia.hiof.no/~borres/ml

Dette utdraget fra et relativt dynamisk vevsted vil alltid være litt ute av synk. Det er bare ment som litt trøst for de som vil ha noe å lese på senga før de sovner.

Sidene er produsert ved hjelp av WXT (http://www.ia.hiof.no/~borres/wxtdoc/), Python og PrinceXML (http://www.princexml.com/)

Rettighetene er de samme som for vevstedet materialet er hentet fra:

- Bruk det til hva du vil så lenge du refererer til vevstedet
- Det følger ikke med noen som helst garanti for at koden og teksten er riktig i en hver forstand.



Høgskolen i Østfold

Innhold

- Markup
- HTML
- Hvilken leser
- Forms
- Form-text
- Form-files
- Form-button
- Form-select
- Form-hidden
- Form og skript

Markup

Hva:

Bakgrunn for dagens XML-teknologi. Noen viktige begreper og sammenhengen mellom dem, SGML, XML, HTML, XHTML, CSS

På den ene siden ønsker vi å tilføre dokumenter **utseende** eller layout og på den andre siden ønsker vi å gi dokumenter en **struktur**.

Behovet for utseende er åpenbart fordi vi ønsker å lage pene dokumenter, med overskrifter, innrykk, ulike fonter og fontstørrelser etc.

Struktur ønsker vi i hovedsak av to grunner:

- Vi vil bruke dokumentene som datasett og skrive eller bruke programmer som kan lese og endre innholdet som strukturerte data. F.eks ekstrahere ingressen fra en serie dokumenter og samle dem i et annet dokument. For at dette skal være mulig må begrepet ingress være definert, identifiserbart, i dokumentene.
- Vi vil lett kunne endre dokumentets layout ved å endre egenskapene til de enkelte elementene i dokumentet. F.eks. endre fonten på alle overskrifter, eller endre linjeavstand i all brødtekst. Dette er viktig også fordi vi ønsker å framstille dokumentet på forskjellige medier som papir, skjermer, små skjermer, overheadkanoner. For at dette skal være mulig må de enkelte elementtypene merkes, det er ikke effektivt å gi individuell form til alle deler av et dokument.

Denne to-sidigheten ved et dokument, utseende og struktur, er nøkkelen til å forstå dokumentbehandling og alle de ulike teknologiene som er under utvikling på dette området. Dette er på mange måter noe av det mest spennende som skjer innen databehandling for tiden. Behovet for å finne, presentere og utveksle innhold i den stadig voksende informasjonsjungelen som kalles Internettet er selvsagt hovedgrunnen til at denne utviklingen er sentral nå.

Markup av tekst er ikke noe nytt fenomen. Det er lett å tenke seg at behovet for å sette merker i en standardisert form i en tekst for å angi hvordan den skulle settes har vært nyttig og nødvendig i trykkeribransjen i lang tid. Elektronisk behandling av dokumenter gjør det selvsagt mulig å sette slike merker i teksten på en slik måte at framstillingsteknologien, skjermdriver eller printerdriver, kan tolke merkene. Begrepet WYSIWYG (What You See Is What You Get) som ble introdusert på 80-tallet i forbindelse med introduksjon av rastergrafiske skjermer er en god illustrasjon på dette. Det WYSIWYG egentlig betyr er at det du lager på skjermen er det samme som det du får ut på skriveren. Som bruker av tekstbehandlingsprogrammer, som f.eks. MSWorld, blir du tilbudt funksjonalitet som i tillegg til å behandle selve teksten, gjør det mulig å bestemme nøyaktig hvordan teksten skal se ut. For at dette skal være mulig må det legges igjen (skjult) informasjon i teksten som skriveren og skjermdriveren kan forstå og tolke. Slik informasjon om dataene kan vi kalle metainformasjon. Det har blitt introdusert en rekke formater, både proprietære og åpne, for å realisere dette.

Øvelse

Lage et enkelt dokument i Microsoft Word, med en overskrift og et par linjer tekst.

- Spar dokumetet som .doc format. Forsøk å åpne det i Ultraedit, og se om du kan finne igjen den teksten du har skrevet. doc er et proprietært binærformat fra Microsoft og du er avhengig av at Utraedit forsøker å tolke binærdataene som tegn.
- Spar dokumentet som .rtf (rich tekst format). Forsøk å åpne det i Ultraedit eller Notepad, og se om du kan finne igjen den teksten du har skrevet. rtf er et format som er introdusert av Microsoft. rtf har fått en viss utbredelse (og et litt frynsete rykte).
- Spar dokumentet som .html. Forsøk å åpne det i Ultraedit eller Notepad, og se om du kan finne igjen den teksten du har skrevet.

Forsøk å endre teksten slik du finner den i Utraedit (eller Notepad) i de forskjellige formatene og forsøk deretter å åpne den i Word.

Skriv dokumentet ved å velge en tilgjengelig skriver, men merk av at du vil skrive til fil. Forsøk å åpne den .prn- fila som lages og se om du kan finne igjen den teksten du har skrevet. prn fila vil for de fleste printere være i Postscript. Postscript er til forskjell fra eksemplene ovenfor et programmeringsspråk som utføres på skriveren.

Hovedfokus i WYSIWYG-strategien er layout, altså hvordan dokumentet skal se ut. Digital behandling av strukturerte data har historisk hatt fokus mot databaser, regnearkformater eller "kommasepararerte" filer. Framstilling av strukturerte data har i stor grad vært behandlet av diverse rapportgeneratorer eller spesialprogrammer i de enkelte tilfellene. Nå er ikke verden fullt så enkel: Microsoft har lagt ned en god del arbeid i å utvikle proprietære teknologier for å integrere verktøyene i Office-familien på en slik måte at data kan sirkulere mellom ulike verktøy i familien.

Mangelen på ryddighet og forståelse av de to perspektivene på et dokument, layout og struktur, er et problem som stikker dypt, både systemteknisk og mentalt. De fleste brukere av tekstbehandlere har ikke dette skillet klart for seg og bruker strukturerelementer og formatspesifikasjoner om hverandre for å oppnå visuelle effekter. Det samme skjer ved skriving av HTML.

XML-teknologien er interessant fordi:

- det gjør det mulig og naturlig å skille klart mellom struktur og layout
- det gjør det mulig å bearbeide begge aspektene ved et dokument hver for seg, og samtidig
- den gjør det teknisk enkelt å definere felles struktur for dokumenter innen en organisasjon eller et fagområde.
- det finnes en rekke produkter og programbiblioteker som gjør behandling av XML-dokumenter sikkert og enkelt.
- den er i motsetning til de fleste av de andre markup-språkene befriende enkel. Det er så enkelt at det er vanskelig å korrumpere selve ideen med proprietære varianter
- den er anvendbar både til tradisjonell dokumentbehandling og til veldefinerte mellomformater i systemløsninger, f.eks. til konfigurasjonsbeskrivelser.
- den trolig vil erstatte de enkleste databaseløsningene.
- den passer som hånd i hanske som mellomformat mot databaseløsninger.

Historie i lommeformat

Historien om World Wide Web og tilhørende teknologier er skrevet i utallige versjoner og finnes en eller annen form i de fleste lærebøker i relevante emner. Denne historien tolkes og analyseres på forskjellig måte, alt etter hvilket fokus forfatteren har. Det du finner her er en svært kort versjon som har fokus på å forstå hvorfor XML er et svar på en utvikling som delvis har vært styrt av et felles mål, og dels har gått av sporet på noen avgjørende punkter.

SGML

(Standard Generalized Mark-up Language) [1]

SGML var opprinnelig et initiativ fra IBM for å muliggjøre utveksling av dokumenter mellom systemer. Arbeidet ble påbegynt så tidlig som på slutten av 60-tallet av Goldfarb, med hjelp av Mosher og Lorie (GML). Arbeidet endte opp som ISO-standard (ISO 8879) i 1986. SGML er en svært omfattende standard som har hatt en viss utbredelse innen organisasjoner med stort behov for dokumentadministrasjon. Typiske anvendelser er forlag som ønsker å kunne publisere i forskjellige formater og f.eks. flyfabrikker med stort behov for å kunne dokumentere produksjon, der kvalitetssikring og sporbarhet er nøkkelord.

SGML har de samme karakteristika som vi forbinder med XML. SGML er utvidbart, det vil si at det ikke er et endelig språk, men et regelsett for å definere språk, et metaspråk. SGML-språk var basert på definisjon av dokumenttyper, DTD (Document Type Definition). Det ble skrevet parsere for SGML og det ble definert et språk for å definere stilsett for SGML-dokumenter, DSSSL (Document Style Semantics and Spesification Language). [2]

Det er alltid vanskelig å vurdere hva som gjør at en teknologi "tar av" mens en annen ikke gjør det. Det finnes en god del logiske argumenter for at SGML kunne slått gjennom som en viktig nøkkelteknologi i et større marked. Hovedgrunnene til at det ikke skjedde var trolig for det første at det er for omfattende og for det andre at behovet for dokumentstrukturering ikke var allment annerkjent som en generell utfordring før bruken av Internett hadde modnet.

HTML

(Hyper Text Markup Language) [3]

HTML ble utviklet ved CERN (opprinnelig: Conseil Européen pour la Recherche Nucléaire, nå:Organisation Européenne pour la Recherche Nucléaire) i 1989 av Tim Berners-Lee. Hensikten var å kunne utveksle og binde sammen dokumenter innen organisasjonen. Arbeidet ble basert på SGML og HTML kan betraktes som et språk definert i SGML. HTML skiller seg altså fra SGML, og XML, ved at det er et definert språk med et fast ordforråd. HTML har et endelig sett med elementer.

HTML ble utviklet uten at noen var i stand til å forutse hvor stor utbredelsen av dette språket skulle bli. Word Wide Web eksploderte og vi fikk generelle nettlesere som kunne parse og framstille HTML.

Det er mange som i ettertid har fundert på hvordan HTML burde ha vært og hvordan nettleserne burde ha utviklet seg for at vi idag skulle hatt en litt enklere verden enn vi har. Slike spekulasjoner er interessante i den grad vi kan bruke dem til å belyse aktuelle problemstillinger i dagens utvklingsarbeid. Det er to forhold som fortjener oppmerksomhet som grunnlag for forståelse.

1. HTML bidrar ikke til å gjennomføre et klart skille mellom struktur og layout. I hovedsak ble HTML designet med elementer som er av strukturell art, men vi vet at mangelen på stilsett tidlig i utviklingen har ført til att de strukturelle elementene brukes til å tvinge fram layout.

2. Nettleserne ble alt fra starten laget med en betydelig toleranse for misformede dokumenter. SGML har til en viss grad en slik toleranse. Dette har opp gjennom mange generasjoner av HTML og nettlesere ført til en ganske uoversiktlig situasjon der dokumenter blir tolket og framstilt på forskjellig og ofte uforutsette måter. Det har dessuten ført til at den dokumentstrukturen som legges til grunn ved parsing blir tvetydig. Scripting, dynamisk HTML på klientsiden, blir en risikosport.

CSS

(Cascading Style Sheets). [4]

Vi må minne om at en av hovedpersonene bak CSS, Håkon Wium Lie, kommer fra Halden og har sin grunnutdanning i informatikk fra HiØ.

CSS er et språk som gjør det mulig å definere hvordan ulike elementer i et HTML- eller XML-dokument skal vises. Hensikten med dette er altså at dokumentet kan lages (i hovedsak) med tanke på struktur, mens utseendet kan defineres og endres uten å endre i selve dokumentet. Dette er interessant også fordi ett og samme dokument kan vises etter de behov ulike medier krever (Store skjermer, små skjermer, papir). CSS er et svar på noe av det som ble ivaretatt med DSSSL i forhold til SGML, men som ikke ble ivaretatt når HTML opprinnelig ble definert.

Det er mange forfattere som mener at CSS ikke er en fullgod løsning på layoutproblemet og de framholder XSL-FO (eXtensible Stylesheet Language-Formatting) som en alternativ eller komplementerende teknologi. I dette materialet vil du finne stoff om CSS og XSLT (eXtensible Stylesheet Language Transformations), og litt om XSL-FO.

XML

(eXtensible Markup Language) [5]

XML er i en viss forstand en ny start, der erfaringene fra historien er ivaretatt. XML er bygget på samme lest som SGML, og er i stor utstrekning et subsett av SGML. Det er vesentlig enklere enn SGML, det har ingen toleranse for ufullstendige formuleringer og det er altså, i motsetning til HTML, ikke et språk men et metaspråk. Det vil si: et regelsett for å definere språk. XML mangler noe viktig for å være et språk, nemlig ord.

Det betyr at når vi skriver et XML-dokument kan vi kontrollere at det er velformet i henhold til de grunnleggende generelle syntaksreglene, og vi kan kontrollere at det er validerbart som dokument av en bestemt type. Det siste forutsetter at vi har definert en dokumenttype i form av en DTD (Document Type Defnition) eller et SCHEMA. Å definere en DTD betyr egentlig at vi definerer vårt eget språk. En validering betyr at vi sjekker at et dokument er skrevet i henhold til dette spåket. XHTML er et slikt språk. Det er i hovedsak HTML4 med utvidede formelle krav til riktig syntaks som blandt annet innebærer at en del attributter som kontrollerer utseende er fjernet og formateringen forutsettes beskrevet som inkluderte eller lenkede stilark.

Når det i mange sammenhenger understrekes at XML er enkelt, så er det en sannhet med modifikasjoner. De grunnleggende syntaksreglene i XML er befriende enkle, men bildet kompliseres en god del av en del praktiske nødvendigheter som dukker opp når fragmenter fra ulike dokumenttyper skal blandes osv. Mer om slike komplikasjoner andre steder i dette materialet.

Andre markup-språk

Det er lett å se at det generelle behovet for å standardisere dokumenter slik at de kan utveksles mellom applikasjoner har ført til en rekke mer eller mindre utbredte standardformater opp gjennom databehandlingens historie. De som er nevnt nedenfor har nesten utelukkende blitt utformet med tanke på å beskrive layout. Det finnes sikkert en rekke andre som er kjente i ulike miljøer.

RTF

Rich Text Format er et format som ble introdusert av Microsoft. Se f.eks.: [6] for en beskrivelse. Du kan se hvordan det ser ut ved å lagre en MS Word fil som rtf og åpne den i Ultraedit eller Notepad eller en lignende teksteditor.

LaTeX

ble opprinnelig introdusert som TeX av Donald Knuth fordi han trengte et verktøy for å kontrollere teksten i en bok han skrev. LaTeX har hatt og har fortsatt utbredelse og tilhengere i akademiske miljøer. Mer om både LaTeX og Knuth på sidene til TeX Users Group: [7]

troff

Utdrag fra innledningen fra Troff User's Manual by Jospeh F. Ossanna and Brian W. Kernighan, se [8]

"troff is a text-formatting program for driving the Graphic Systems phototypesetter on the UNIX and GCOS operating systems. This device is capable of producing high quality text.... troff allows the user full control over fonts, sizes, and character positions, as well as the usual features of a formatter: right-margin justification, automatic hyphenation, page titling and numbering, and so on. It also provides macros, arithmetic variables and operations, and conditional testing, for complicated formatting tasks."

Postscript

Postscript representerer en litt annen innfallsvinkel til formatering. Vi markerer ikke for Postscript i dokumentet, men det finnes oversettere som genererer Postscript basert på annen markup. Hele dokumentet omgjøres til et program som så kan transporteres og kjøres på den enheten som skal presentere det. Printere er typiske enheter som kan utføre postscriptkode. [9]

Referanser og eksempler:

- [1] SGML (Standard Generalized Mark-up Language):
  www.w3.org/MarkUp/SGML/
- [2] DSSSL(Document Style Semantics and Spesification Language):
  www.jclark.com/dsssl/
- [3] HTML 4.01(HyperText Markup Language):
  www.w3.org/TR/html4/
- [4] CSS (Cascading Style Sheets):
  www.w3.org/Style/CSS/
- [5] XML (Extensible Markup Language):
  www.w3.org/XML/
- [6] RTF (Rich text Format):
  www.biblioscape.com/rtf15_spec.htm
- [7] LaTeX:
  www.tug.org/
- [8] troff:
  www.kohala.com/start/troff/troff.html
- [9] Postscript:
  www.adobe.com/products/postscript/main.html

HTML

Hyper Text Markup Language

Hva:

Ulike versjoner av HTML og strategier for å velge versjon, HTML4, XHTML, strict, transitional

Siden Tim Berners-Lee [1] ved CERN introduserte HTML i 1989, har vi hatt flere versjoner av HTML med litt forskjellige egenskaper, og vi har hatt en serie med nettlesere som i varierende grad har tolket de ulike HTML-versjonene litt forskjellig. Nettleserene har hatt det til felles at de har hatt en stor grad av toleranse når det gjelder å tolke HTML. De har akseptert HTML som ikke er velformet i henhold til krav om at elementer skal være lukkede, ikke overlappe osv. Nettleserne har også vært bakover kompatible slik at både gamle HTML-formuleringer og etablerte dårlige skrivevaner skal fungere. Nettlesere har også introdusert sine egne små finurligeheter for å oppnå spesielle effekter. Som regel er dette gjort slik at effektene ikke gjør noen skade i andre nettlesere, men slik at den spesielle nettleseren har hatt et fortrinn som framvisningsprogram. Det er lett å forstå at det har blitt slik ut fra markedsvurderinger. En nettleser som bare ville akseptere velformet HTML ville trolig få en skare idealistiske tilhengere, men ville neppe bli noen slager i markedet.

En konsekvens av denne utviklingen er at vi har en ganske forvirret verden, og det er praktisk talt umulig å holde seg ajour med hva som virker, hvordan det virker og under hvilke betingelser det virker. Det beste vi kan gjøre er å fokusere på det abeidet som gjøres i W3C og håpe at anbefalingene derfra får gjennomslag.

I et langsiktig perspektiv synes det klart at alt nytt materiale som publiseres på Internett bør være XML. Når alle de vanlige nettleserne kommer dit at de kan handtere XML koplet med et eller annet stilsett (CSS og/eller XSLT_FO) kan vi publisere i våre egendefinerte XML-språk eller vi kan publisere i et allment kjent og standardisert XML-språk som XHTML. I øyeblikket er vi ikke der, men det er i alle fall lurt å legge seg til vaner som er tilpasset en slik framtid.

Oversikten over definisjonene av alle de HTML-versjonene vi har hatt siden 1989 overlater jeg til de som har historiske interesser. W3C er en kilde for materiale. I denne modulen setter vi fokus på på HTML 4 og tilhørende varianter.

HTML 4

Nåværende definisjon av HTML er HTML 4.01. HTML 4, er som tidligere versjoner av HTML, definert på en slik måte at det er ganske mye toleranse for hva som akspeteres som lovlig HTML. HTML som avledet av SGML har i seg selv en viss toleranse for avvik fra det som er korrekt XML. For oss som publiserer på web er det av og til litt uklart hva som er lovlig i forhold til en HTML-definisjon, og hva som "virker" fordi nettleserne har stor toleranse.

XHTML

XHTML er HTML 4 formulert som ekte XML. Det vil si at det er definert en DTD som beskriver XHTML. Alle de kravene til velformethet som følger av XML skal være oppfylt. De viktigste kravene er:

Riktig	Galt
Elementer må ikke overlappe.	
`<h1>Overskrift</h1>` `<p>Avsnitt2</p>`	`<h1>Overskrift` `<p></h1>Avsnitt2</p>`
Element- og atributtnavn skal skrives med små bokstaver (minuskler).	
`<h1>Overskrift</h1>` `<p>Avsnitt</p>`	`<H1>Overskrift</H1>` `<P>Avsnitt</p>`
Ikke-tomme elementer skal avsluttes eksplisitt med en sluttag.	
`<p>Avsnitt1</p>` `<p>Avsnitt2</p>`	`<p>Avsnitt1` `<p>Avsnitt2`
Attributtverdier skal stå i anførselstegn.	
`<img src="up.gif" alt="Bildet"/>`	`<img src=up.gif alt=Bildet/>`
Verdi-attributt par skal skrives fullt ut	
`<dl compact="compact"/>`	`<dl compact/>`
Tomme elementer skal enten ha en sluttag, eller slutte med />	
`<hr/>` ` `	`<hr>` ` `

På W3C finner du en komplett liste og noen gode råd. Se: www.w3.org/TR/xhtml1/

XHTML kommer i tre versjoner: strict, transitional og frameset. Vi ser nærmere på de to første. Begge krever velformethet, men de skiller seg litt når det gjelder krav til hva vi kan legge inn av layout i selve HTML-koden, det vil i hovedsak si hvilke attributter vi kan anvende. Strict-versjonen aksepterer f.eks. ikke:

```
<img src="up.gif" border="1" alt="bilde"/>
```

Årsaken er at attributten border anses å være formatering som skal ivaretas i et stilsett. Vi kan imidlertid skrive:

```
<img src="up.gif" style="border:1px" alt="bilde"/>
```

Nedenfor finner du et dokument som er formulert på litt forskjellige måter for å tilfredstille kravene til strict og loose. begge versjonene bruker et stilsett som forutsettes å ligge i samme katalog som brukerfila og som ser slik ut:

```
body {
   color: #783E1E;
   font-family: "Geneva", "Verdana", sans-serif;
   text-align: left;
   margin-left:10px;
   margin-top:10px;
}

h1{
   font-size: 24px;
   font-weight: bold;}

p{
font-size:13px;
}

a:link    {color:#3333FF;}
a:visited {color:#6666FF;}
a:active  {color:#AAAAAA;}
a:hover   {color:#FF3333;}

img {
   border-style: none;
}
```

Strict

Dette er den formen som er strengest, og skillet mellom struktur og form er gjennomført i henhold til XHTML-definisjonen. En validerbar fil kan se slik ut:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <link rel="STYLESHEET" href="def.css"/>
  <title>Strict</title>
</head>
<body>
<h1>Hallo</h1>
<p>alle sammen</p>
<p>
  <a href="http://validator.w3.org/check/referer">
  <img src="../gfx/valid-xhtml.gif" alt="strict"/>
  </a>
</p>
<p>
<a href="javascript:history.back()">Tilbake</a>
</p>
</body>
</html>
```

Nettleserne aksepterer at vi utelater den første linja. Se fila i nettleseren: strict.html

W3C anbefaler at den første linja beholdes og at filene gis extension .xml. Dett kan imidlertid skape problemer siden en server kan sende en mime-angivelse som ikke tolkes riktig av nettleseren. Det er en del grumsete farvann her.

Transitional

Transitional, eller "loose", er en slags overgangsvariant som tillater layout spesifikasjon i elementene. Begrunnelsen for denne er to. Dels skal det være mulig å lage layout beregnet på nettlesere som ikke handterer stilsett. Dels skal det være mulig for forfattere som ikke behersker stilsett å lage layout, slik de er vant til. En validerbar fil kan se slik ut:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
        <link rel=STYLESHEET href="def.css">
        <meta name="GENERATOR" content="SiteLite 3">
        <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
        <title>loose</title>
</head>

<body>
<h1>Hallo</h1>
<p>alle sammen</p>
<p>
  <a href="http://validator.w3.org/check/referer">
  <img border="0" src="../gfx/valid-html401.gif" alt="loose"/></a>
</p>
<p>
<a href="javascript:history.back()">Tilbake</a>
</p>
</body>
</html>
```

DEMO: http://www.ia.hiof.no/~borres/ml/html/loose.html

Rydding og sjekking

Når vi skriver en ny fil er det lurt å validere den med jevne mellomrom mot valideringstjenesten hos W3C eller mot et annet valideringsverktøy.

Hvis vi har en gammel fil vi ønsker å oppdatere, så finnes det verktøy som hjelper oss å gjøre grovarbeidet. Tidy er et slikt verktøy, se tidy.sourceforge.net

Referanser og eksempler:

- [1] Tim Berners-Lee:
  HTML
  no.wikipedia.org/wiki/Tim_Berners-Lee
- Tidy: tidy.sourceforge.net
- W3C's valideringstjeneste: validator.w3.org/check/referer
- Multipage Validator: www.validator.ca/

Hvilken nettleser

Hva:

Hvordan finne ut vilken nettleser vi kjører, på klienten eller tjeneren

Dersom vi skal lage dynamiske elementer på en vevside, blir det fort nødvendig for et script å identifisere hvilken nettleser det opererer i. Scriptet må tilpasse seg nettleseren. Dette blir fort en risikabel øvelse fordi nettlesere endrer mening fra en versjon til den neste og fordi kombinasjonsmulighetene fort blir så store at scriptet blir komplekst og sårbart.

Det er i prinsipp tre strategier å følge:

- Vi kan identifisere nettleseren og splitte koden basert på denne informasjonen. Dette er risikabelt siden vi enten må sitte med kode for en lang rekke mulige nettleserversjoner. Nye versjoner kan velte strategien vår.
- Vi kan la tjeneren avgjøre hvilken nettleser som har sendt en forspørsel og sende tilbake en side som er tilpasset denne nettleseren. Denne strategien er sammenlignbar med den første.
- Vi kan teste eksistensen av de objektene og funksjonene vi ønsker å bruke og bruke dem hvis de finnes. Hvis vi ikke finner noe vi kan bruke kan vi gi en melding om dette. Denne strategien er å foretrekke når vi skal lage dynamiske vevsider. Det er i hovedsak denne strategien som blir brukt i dette materialet, se f.eks. modulen AJAX

Nedenfor ser vi på noen muligheter for å finne ut vilken nettleser som leser siden, uten tanke på å skrive dynamisk HTML. Vi ser både på nettleserens mulighet for å "kjenne seg selv", og tjenerens mulighet for å identifisere spørreren.

Klienten vet det selv

Vi kan bruke informasjon i objektet **navigator** som vi når fra JavaScript på klienten selv.

Test her

JavaScriptet er slik:

```
function who(targetid)
{
  var useragent = navigator.userAgent;
  S='';
  elt=document.getElementById(targetid);
  if(useragent!=null)
  {
    S+='klienten sier:<br/>';
    S+='<b>'+ navigator.userAgent + '</b>';

  }
  else
    S='Finner ikke ut av det';
  elt.innerHTML=S;
}
```

Tjeneren vet det

Tjeneren vet en masse om vilken "useragent" eller nettleser som henvender seg for å få lastet opp en vevside. Nedenfor finner du demonstrasjon av hvordan vi kan få tilbake informasjon fra en Linux/apache server via et Pythonscript og fra en Microsoft Server via et asp-script.

Microsofts IIS og asp

Hva sier serveren. Vi spør med asp-kode:

```
<%
Option Explicit
%>

<html>
<head>
<title>Who asked</title>
</head>
<body>
<h2>Server says: </h2>
<%
Response.Write Request.ServerVariables("HTTP_USER_AGENT")
%>
</body>
</html>
```

Dette er resultatet:

DEMO: http://donau.hiof.no/borres/ml/WhoAskedx.asp

Linux Apache og python

Dette scriptet returnerer det det vet om html-fila som spør:

```
#! /usr/bin/python2.3
import os, string
print """Content-type: text/html

<html>
  <head>
    <title>Who asked</title>
  </head>
  <body>
  <h2>Server says</h2>
  <p>
"""
print str(os.environ['HTTP_USER_AGENT'])
print """
  </body>
</html>
"""
```

Dette er resultatet:

DEMO: http://www.ia.hiof.no/~borres/cgi-bin/what/who.py

Microsoft vet det

Vi kan altså plukke opp den ønskede informasjonen i Javascript på klienten selv eller vi kan finne det vi ønsker med skriptet på tjeneren. Hva hvis vi ikke skriver dynamiske sider ved hjelp av Javascript og ikke bruker noen tjenerstøtte ?

Microsoft har implementert en tolkning av et par kommentarformater som kan hjelpe oss å identifisere Microsoft Internet Explorer og lage HTML- eller CSS-fragmenter som brukes bare av IE.

Se på følgende to formuleringer:

1. ```
   <!--[if IE]>
   Hei du bruker (IE5 eller høyere).
   <![endif]-->
   ```

2. ```
   <![if !IE]>
   Hei du bruker ikke IE.
   <![endif]>
   ```

Den første er en komplett kommentar som ignoreres av alle andre nettlesere enn MSIE. Den andre plukkes også opp av MSIE og innholdet ignoreres fordi vi har sagt !IE (not IE), men alle andre lesere viser fram teksten: "Hei du bruker ikke IE.". Den siste fungerer fordi alle andre lesere ignorerer de to meningsløse taggene over og under teksten.

Problemet med formulering 2 er at den ikke validerer som strict XHTML. Formulering 1 derimot validerer fordi det er ren kommentarsyntaks. Det ser ut til at vi kan reformulere uttrykk 2 slik for å få det til å validere:

```
<--![if !IE]><-->
Hei du bruker ikke IE.
<!--><![endif]>
```

Merk at vi kan spesifisere vilken MSIE verson vi spør etter ved å sette versjonsnummeret etter IE, feks.:

```
<!--[if IE 6]>
Hei du bruker IE versjon 6.
<![endif]-->
```

Og vi kan bruke lt (less than), gt(greater than), lte (less or equal), gte(greater or equal) for å sjekke versjonsnummer. F.eks.:

```
<!--[if lt IE 6]>
Dette er en tidligere versjon av IE enn versjon 6
<![endif]-->
```

Du kjører forresten ikke Microsoft Internet Explorer

Det kan være delte meninger om en skal bruke slike produsentavhengige finurligheter. Argumentet **for** er at så lenge vi bruker MS triks til å overkomme MS mangler så blir det så og si i familien. Argumentet **mot** er selvsagt at vi roter oss inn i løsninger som blir sårbare og lite vedlikeholdbare. En bruksmåte som er ganske vanlig er å inkludere CSS som er nødvendig for å få IE til å vise HTML som andre lesere.

I alle fall så finner du noen eksempler på egne sider. Du må teste dem i forskjellige nettlesere for å se effekten:

- En enkel side som sier fra om nettleseren, men som ikke validerer:

  DEMO: http://www.ia.hiof.no/~borres/ml/vilkenleser/ifhack1.html
- En enkel side som sier fra om nettleseren, og validerer:

  DEMO: http://www.ia.hiof.no/~borres/ml/vilkenleser/ifhack2.html
- En enkel side som setter en egen stil(CSS) for IE:

  DEMO: http://www.ia.hiof.no/~borres/ml/vilkenleser/ifhack3.html
- En enkel side som gjør det samme, men med include:

  DEMO: http://www.ia.hiof.no/~borres/ml/vilkenleser/ifhack4.html
- ... det kommer flere

Referanser og eksempler:

- Hvilken nettleser er det ?
  www.mozilla.org/docs/web-developer/sniffer/browser_type.html
  eller se: www.mozilla.org/catalog/web-developer/
- En "sniffer" som rapporterer datatransporten i en http-forespørsel: web-sniffer.net/

Forms

Hva:

Hvordan fange opp data fra en HTML-side

En vevside oppfattes i hovedsak som en framvisningsside, men ganske ofte ønsker vi å sende data fra vevsiden til tjeneren. I (X)HTML har vi en egen mekanisme for dette. Vi fyller inn data eller velger blandt muligheter på vevsiden og sender vår input ved å trykke på en knapp. Våre data sendes til tjeneren og behandles videre av et program, typisk et CGI-skript. Et eksempel:



Den HTML-koden som tar seg av dette ser slik ut:

```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py"
      method="get">
<div>
    <input type="text" name="message"/>
    <input type="submit" value="Send"/>
</div>
</form>
```

Attributten action angir hvilket skript som skal motta dataene, get angir en av to måter å sende data på. Mer om dette senere. Vi ser videre at vi har to elementer inne i form-elementet: Et element for å skrive inn en tekst, og et element, submit, som er en knapp som utløser sending av data. Hvordan mottagerskriptene på tjeneren er bygd opp er ikke tema for denne modulen. Se CGI. Et skript sender et resultat tilbake til nettleseren, oftest i form av en html-side.

Logikken er altså at vi definerer en form og fyller den med så mange input-elementer vi ønsker. I tillegg til tekst-input kan vi ha radioknapper, sjekkbokser, skjulte elementer, passord, filer og mer generelle knapper. Vi ser litt på hver av disse i de følgende modulene.

Vi skal se litt på noen muligheter vi har med selve form-elementet. Du kan inspisere de Pythonskriptene som er brukt under referanser i disse form-modulene.

action

Attributten action er adressen til det skriptet som skal motta data fra formen.

target

Attributten target brukes akkurat som den brukes i HTML. Du kan angi hvilket vindu du vil ha resulatet fra skriptet presentert i. Dersom du bruker frames er dette en mulighet til styre resulater av skriptets behandling ut i en annen frame.

method

Hvilke overføringsmetoder vi kan bruke er bestem av den protokollen som kontrollerer dataoverføring mellom klient og tjener. Når det gjelder vanlig dataoverføring mellom en nettleser og et skript så er dette HTTP (Hyper Text Transfer Protocol). HTTP gir oss for vårt formål valget mellom to metoder, **post** og **get**.

get

get har begrensninger når det gjelder datavolumet, men har noen fordeler når det gjelder å hardkode parameterstyrte skript.

Anta at vi har en konstruksjon som eksempelet øverst på siden.



Hvis du skriver inn teksten hei, og trykker på send-knappen vil du se følgende i adressefeltet på nettleseren:

```
http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py?message=hei
```

Dersom vi hadde hatt flere datafelt som skulle overføres ville de ha kommet etter hverandre, adskilt med tegnet &, f.eks. slik:

```
http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py?message=hei&fra=bs
```

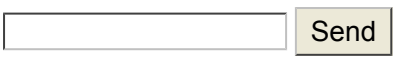
Vi kunne oppnådd den samme effekten av skriptet ved å skrive adressen med parameterlista direkte inn i nettleseren, eller ved å legge en link som her:

http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py?message=hei

Metoden get har den fordelen at det er lett å rekonstruere en bestemt side, siden parametrene så og si inngår i adressen.

post

post viser ikke dataene som skal overføres eksplisitt, men kan ta større datamengder. Du kan prøve eksempelet nedenfor som er identisk med det forrige, bortsett fra at metoden er skiftet til post:




```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py"
      method="post">
    <div>
    <input type="text" name="message"/>
    <input type="submit" value="Send"/>
    </div>
</form>
```

Referanser og eksempler:

W3C's beskrivelse av form-elementet i HTML: www.w3.org/TR/html4/interact/forms.html

Python skriptet som er brukt i denne modulen: std

En samleside for testing av forms: formtester.html

Form: text

Hva:

text-elementet i en form

Vi behandler følgende input typer under denne overskriften: input:text, input:password og textarea.

text

text er en enkel mekanisme for å lese inn en tekststreng uten linjeskift. For eksempel slik:



HTML-koden ser slik ut:

```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py"
      method="get">
    <div>
    <input type="text" name="message"/>
    <input type="submit" value="Send"/>
    </div>
</form>
```

Vi kan spesifisere flere attributter enn name og type til et text-felt: value, maxlength og size. F.eks. slik:



HTML-koden ser slik ut:

```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py"
      method="get">
    <div>
    <input type="text" name="message" value="hei" maxlength="10" size="15"/>
    <input type="submit" value="Send"/>
    </div>
</form>
```

value har en forutbestemt verdi, som kan endres. Det er denne verdien som plukkes opp av skriptet. Verdien endres når vi skriver inn i feltet. maxlength sier oss hvor mange tegn feltet skal akseptere. size sier hvor stort (langt) feltet er. Her er size større enn maxlength, og vi vil ikke få skrevet inn i de 5 siste tegnposisjonene. Dersom det er omvendt, vil teksten scrolle. Dersom vi slår return vil vi aktivere send-knappen. Dette er en oppførsel vi kjenner fra dialogbokser i vanlig programmer, der OK-knappen ofte fungerer som default avslutt/lukk knapp.

password

Passordfelt er i hovesak som tekstfelt. Den eneste forskjellen er at ekko på den intastede teksten markeres som stjerner, prikker e.l.

Melding: Passord: Send

HTML-koden ser slik ut:

```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py"
      method="get">
    <div>
    Melding:
    <input type="text" name="message" size="15"/>
    Passord:<input type="password" name="pwd" size="15"/>
    <input type="submit" value="Send"/>
    </div>
</form>
```

Merk at passord-feltet har ingen annen sikkerhet enn at teksten skjules for øyet i inntastingsformen. Dataene trasporteres som om det var vanlig tekst. Dersom vi bruker metoden get, som ovenfor, vil du se passordet i klartekst i adressefeltet til nettleseren.

textarea

textarea skiller seg litt fra de andre tekstfeltene ved at elementet selv heter textarea. F.eks. slik:

Send

```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py"
      method="post">
   <div>
<textarea name="message" cols="15" rows="5">
</textarea>
<input type="submit" value="Send"/>
   </div>
</form>
```

Attributtene cols og rows er selvforklarende. Du kan kontrollere hvordan teksetn brekkes ved hjelp av stilsett. Den gamle wrap-attributten er ikke lovlig i XHTML

Merk en liten detalj som kan være ganske irriterende. Dersom kildekoden til html-fila innholder en tabulator foran textarea-elementet, vil tabulatoren være effektiv i selve tekstfeltet. Du merker dette hvis du plasserer cursor i feltet og klikker. Ovenfor er tekstfeltet plassert uten tabulator.

Referanser og eksempler:

Python skriptet som er brukt i denne modulen: std

Form: files

Hva:

opplasting av filer fra en form

Vi kan laste opp filer fra vår egen maskin, klienten, til tjeneren.

Grunnformen er slik:

Last opp en textfil:



Koden er slik:

```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py"
      accept="text/*"
      method="post"
      enctype="multipart/form-data">
      <div>
    <input type="file" name="minfil" size="40" />
    </div>
    <div>
    <input type="submit" value="Send"/>
    </div>
</form>
```

Vi ser at feltet gir opphav til både et tekst-felt og en browser knapp.

Her må vi bruke metoden: post. Det er mulig å bruke attributten accept som brukes til å angi hva slags filer som skal aksepteres av tjeneren. Verdien er lovlige MIME-typer. F.eks. vil accept="text/*" godta MIME-typer: text/plain, text/html osv. Dersom accept ikke brukes vil alle filtyper ansees lovlige.

Referanser og eksempler:

Python skriptet som er brukt i denne modulen: std

Form: button

Hva:

knapper

Knapper kommer i følgende varianter: input:submit, input:reset, input:button, input:image og button.

submit

En submit-knapp sender data til tjeneren slik som angitt av action og method attributtene i den omgivende formen. Vi har sett eksempler på dette i de foregående modulene.

hei Send

HTML-koden ser slik ut:

```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py"
      method="get">
<div>
    <input type="text" name="message" value="hei"
           maxlength="10" size="15"/>
    <input type="submit" value="Send"/>
</div>
</form>
```

value attributten sendes til tjeneren og kan identifisere hvilken handling som skal gjøres i det mottagende skriptet. Dersom vi ikke angir value for en submit-knapp, får den en default verdi.

reset

En reset-knapp tømmer feltene som ligger i formen. Det vil si den blanker ikke nødvendigvis, men den setter verdiene tilbake til initialverdiene. Vi kan betrakte dette som en hjelpefunksjon når brukeren angrer på alt han/hun har fylt ut, og ønsker å starte på nytt.

hei Send Reset

HTML-koden ser slik ut:

```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py"
      method="get">
<div>
    <input type="text" name="message" value="hei"
           maxlength="10" size="15"/>
    <input type="submit" value="Send"/>
    <input type="reset" value="Reset"/>
</div>
</form>
```

button

Et button felt har en sluttag og alt som ligger inne i elementet er klikkbart. Dersom det skal skje noe når vi klikker må vi ta vare på det selv. Slik sett kan vi si at en button ikke er direkte koplet til en form. F.eks.

Klikk her !

HTML-koden ser slik ut:

```
<p>
    <button onclick="javascript:alert('jada')">Klikk her !</button>
</p>
```

Eller vi kan lage noe slikt:

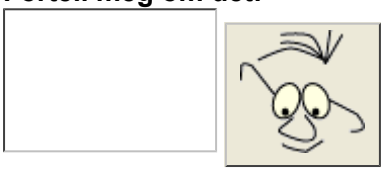


HTML-koden ser slik ut:

```
<button style="background-color:yellow"
        onclick="javascript:alert('jada, ikke mas')">
    <img src="bs1.gif" alt="bilde"/>
</button>
```

Vi kan imidlertid kombinere en form-relatert funksjon som submit eller reset med et button-element. Dette gjør vi ved å bruke attributten type for en button. Vi kan f.eks. lage:

Fortell meg om det:



HTML-koden ser slik ut:

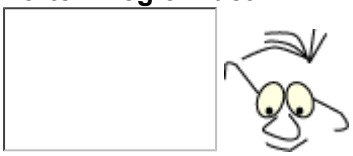
```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py"
      method="post">
<div style="font-weight:bold">Fortell meg om det:</div>
<div>
<textarea name="message" cols="15" rows="5" >
</textarea>
<button type="submit" value="Send">
<img src="bs1.gif" alt="bilde"/>
</button>
</div>
</form>
```

Vi kan lage tilsvarende reset knapper ved å gi type-attributten verdien reset.

image

Du kan lage bilder som submit-knapper med input type image, slik:

Fortell meg om det:



HTML-koden ser slik ut:

```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py"
      method="get">
    <div style="font-weight:bold">Fortell meg om det:
    </div>
    <div>
<textarea name="message" cols="15" rows="5">
</textarea>
<input type="image" src="bs1.gif"/>
</div>
</form>
```

Merk at koordinatene til museklikket på bildet sendes til scriptet. Origo i øvre venstre hjørne og positiv y-akse nedover.

Referanser og eksempler:

Python skriptet som er brukt i denne modulen: std

Form: selection

Hva:

valg blandt flere muligheter

Vi kan formulere valg ved hjelp av tre mekanismer: input:radio, input:checkbox og select.

radio

Radioknapper er en standardisert måte å gi brukeren mulighet for å velge høyst ett blandt et antall alternativer. F.eks. slik:

Hvordan kom du hit ?
Busset
Syklet
Bilte
Gikk
På annen måte
Send

HTML-koden ser slik ut:

```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py" method="get">
  <div style="font-weight:bold">Hvordan kom du hit ?</div>
  <div>
  <div>
    <input type="radio" name="transport" value="1"/>Busset
  </div>
  <div>
    <input type="radio" name="transport" value="2"/>Syklet
  </div>
  <div>
    <input type="radio" name="transport" value="3"/>Bilte
  </div>
  <div>
    <input type="radio" name="transport" value="4"/>Gikk
  </div>
  <div>
    <input type="radio" name="transport" value="5"/>På annen måte
  </div>
  <div><input type="submit" value="Send"/></div>
  </div>
</form>
```

Merk at alle radio feltene har samme verdi i name-attributten. Det er dette som gjør at de oppfattes som en gruppe, og det er i denne gruppa det er lov å gjøre høyst ett valg. En gruppe radioknapper sender bare en verdi til tjeneren, nemlig value- attributten. I eksempelet ovenfor er det valgt å nummerere alternativene. Vi kunne ha valgt å gi en mer beskrivende verdi, f.eks. den teksten som er knyttet til alternativet (Busset, Syklet osv.)

Dersom vi skal ha flere grupper radio-knapper på en side må vi altså grupppere dem logisk ved å gi forskjellig name i hver gruppe, og vi bør selvsagt gruppere dem visuelt på en eller annen måte.

checkbox

I en gruppe checkboxer skal brukeren kunne velge ingen, ett eller mange alternativer. F.eks. slik:

Hvilke aviser leser du daglig ?
☐ Dagbladet
☐ VG
☐ Dagsavisen
☐ Aftenposten
☐ Klassekampen
☐ Andre
Send

HTML-koden ser slik ut:

```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py" method="get">
  <div style="font-weight:bold">Hvilke aviser leser du daglig ?</div>
  <div>
    <input type="checkbox" name="aviser" value="1"/>Dagbladet
  </div>
  <div>
    <input type="checkbox" name="aviser" value="2"/>VG
  </div>
  <div>
    <input type="checkbox" name="aviser" value="3"/>Dagsavisen
  </div>
  <div>
    <input type="checkbox" name="aviser" value="4"/>Aftenposten
  </div>
  <div>
    <input type="checkbox" name="aviser" value="5"/>Klassekampen
  </div>
  <div>
    <input type="checkbox" name="aviser" value="6"/>Andre
  </div>
<div><input type="submit" value="Send"/></div>
</form>
```

Selv om checkboxene over logisk sett er å berakte som en gruppe, behandles hvert input-felt for seg. Det går en uavhengig melding for hvert felt. Ovenfor er valgt å gi alle samme name-attributt. Det betyr at mottagerskriptet må finne fra alle parametre med det navnet og se hvilke verdier som er koplet til navnene. Det vil i mange tilfelle trolig være enklere å programmere skriptet dersom vi bruker forskjellig name-attributter.

Du kan forhåndssette et valg ved hjelp av attributten checked="checked".

select

Select er en fleksible input mekanisme som kan gis forskjellig utseende (scroll-list og drop-down) og forskjellig funksjonalitet (radio eller checkbox).

Den enkleste formen fungerer som en radio-grupp på den måten at brukeren kan velge høyst en av mange. F.eks.:

Hvordan kom du hit ?

Busset Send

HTML-koden ser slik ut:

```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py"
      method="get">
<div style="font-weight:bold">Hvordan kom du hit ?</div>
<div>
<select name="transport">
  <option value="Busset">Busset</option>
  <option value="Syklet">Syklet</option>
  <option value="Bilte">Bilte</option>
  <option value="Gikk">Gikk</option>
  <option value="Annet">På annen måte</option>
</select>
<input type="submit" value="Send"/>
</div>
</form>
```

Den alternative formen gir oss en scrollbar meny. Eksempelet nedenfor er dessuten satt opp med attributten multiple="multiple" slik at vi kan velge flere ved å hold ctrl-tasten nede.

Vilke aviser leser du daglig ?

Dagbladet
VG
Aftenposten
Dagsavisen
Klassekampen
Andre

Send Blank

HTML-koden ser slik ut:

```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py"
      method="get">
<div style="font-weight:bold">
   Vilke aviser leser du daglig ?
</div>
<div>
<select name="aviser" multiple="multiple" size="4">
  <option value="Dagbladet">Dagbladet</option>
  <option value="VG">VG</option>
  <option value="Aftenposten">Aftenposten</option>
  <option value="Dagsavisen">Dagsavisen</option>
  <option value="Klassekampen">Klassekampen</option>
  <option value="Andre">Andre</option>
</select>
<input type="submit" value="Send"/>
<input type="reset" value="Blank"/>
</div>
</form>
```

Vi ser dessuten at attributten size er satt til 4 for å vise 4 linjer i menyen.

Vi kan utvide den siste formen med optgroup-elementer for å gruppere valg. F.eks. slik:

Vilke aviser leser du daglig ?

Dagbladet
VG
Aftenposten
Dagsavisen
Klassekampen
Andre
Halden arbeiderblad
Halden dagblad

Send Blank

HTML-koden ser slik ut:

```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py"
      method="get">
<div style="font-weight:bold">
  Vilke aviser leser du daglig ?
</div>
<div>
<select name="aviser" multiple="multiple" size="4">
  <optgroup label="Riksaviser">
  <option value="Dagbladet">Dagbladet</option>
  <option value="VG">VG</option>
  <option value="Aftenposten">Aftenposten</option>
  <option value="Dagsavisen">Dagsavisen</option>
  <option value="Klassekampen">Klassekampen</option>
  <option value="Andre">Andre</option>
  </optgroup>
  <optgroup label="Lokalaviser">
  <option value="HA">Halden arbeiderblad</option>
  <option value="HD">Halden dagblad</option>
  </optgroup>
</select>
<input type="submit" value="Send"/>
<input type="reset" value="Blank"/>
</div>
</form>
```

Referanser og eksempler:

Python skriptet som er brukt i denne modulen: std

Form: hidden input

Hva:

skjulte felt

Vi kan legge inn skjulte felt i en form. Det betyr at vi sender name-value par etter ønske.

Formen er enkel. Vi kan f.eks skrive:

```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py"
      method="get">
  <div>
   <input type="hidden" name="navn" value="ole"/>
   <input type="submit" value="Send"/>
  </div>
</form>
```

Vi vil ikke se annet en Send-knappen:

Send

og brukeren vil ikke se hva som sendes dersom han ikke inspiserer kildekoden til vevsiden.

En anvendelse av skjulte felt er i sesjoner, eller dialoger mellom klient og tjener. Den protokollen som brukes, HTTP, er tilstandsløs. Det vil si at det finnes ingen hukommelse som husker fra en henvendelse til en annen. Vi må altså på en eller annen måte etablere en mekanisme som husker. Det er flere måter å ordne dette på. En måte er å bruke cookies, en annen er å bruke skjulte felt. Cookies er beskrevet i modulen: Sesjoner

Eksempelet nedenfor illustrerer hvordan et skjult felt kan brukes som hukommelse i en sesjon.

Start her

Denne mekanismen med skjulte felt kan selvsagt kombineres med data som lagres i filer på tjeneren. Det skjulte feltet kan identifisere en fil som akkumulerer data fra en sesjon.

Referanser og eksempler:

Python skriptetene som er brukt i denne modulen:

- std
- scriptsequence

Forms - Scripting

Hva:

scriptkontroll av form-elementer

Det er to problemstillinger som melder seg i forbindelse med form-elementer:

- Kan vi bruke Javascript til å kontrollere/bearbeide input-verdier før de går til tjeneren. Input kontroll kan være et stikkord.
- Kan vi bruke form-elementer som f.eks. select uten å kople dem til overføring av data til tjener. Altså bruke dem for å kontrollere aktiviteter lokalt på klienten, i nettleseren.

Vi skal se litt på disse problemstillingene i denne modulen.

inputkontroll

Vi ser på et enkelt eksempel:

Medlemsnr: *
Melding:
meldingen her

Send Blank

HTML-koden:

```
<form action="http://www.ia.hiof.no/~borres/cgi-bin/forms/std.py"
      method="post"
      onsubmit="javascript:return validate()">
<div>
  Medlemsnr: <input type="text" size="14" name="no" id="no"/> *
</div>
<div>Melding:<br/>
<textarea name="message" cols="25" rows="7">meldingen her
</textarea>
<br/>
<input type="submit" value="Send"/>
<input type="reset" value="Blank"/>
</div>
</form>
```

Vi vil sjekke at medlemnummeret er fylt inn og at det består av bare siffere. Det er mange måter å gjøre dette på. Pass på at du velger en løsning som virker på de nettleserne du har som målgruppe. Koden nedenfor er ikke særlig sofistikert, men ser ut til å fungere greitt på nye nettlesere. Det er rimelig greitt å generalisere koden til flere felter og andre betingelser.

```
function IsEmpty(aTextField)
{
   if ((aTextField.value==null) || (aTextField.value.length==0))
      return true;
   return false;
}

function IsNumeric(theText)
{
   var legals = "0123456789";
   if (theText.length==0)
       return false
   for (i = 0; i != theText.length; i++)
      if (legals.indexOf(theText.charAt(i)) == -1)
        return false;
   return true;

}

function validate()
{
   mnummer=document.getElementById("no");
   if(IsEmpty(mnummer))
   {
      alert('Mangler medlemsnumemr')
      return false;
   }

   if (!IsNumeric(mnummer.value))
   {
      alert('Bare siffere i medlemsnummeret')
      return false;
   }
  return true;
}
```

Merk spesielt den mekanismen som gjør at vi "dreper" submit knappens ambisjoner om å sende data til tjeneren.

```
onsubmit="javascript:return validate()"
```

Dersom javascriptet returnerer false, så vil ikke submit-aksjonen bli effektuert.

Referanser og eksempler:

Python skriptet som er brukt i denne modulen: std