DOM – Document Object Model: 1
Typer af noder i træet: 3
Reading: 4
document: 6
element: 7
attributter: 12
parser fejl: 13
XPATH: 13
Stier i XPATH: 13
Prædikater: 14
Operatorer: 15
Funktioner i XPATH: 15
XPATH test program: 17
writing: ..... **Fejl! Bogmærke er ikke defineret.**
Rekursive funktioner: ..... **Fejl! Bogmærke er ikke defineret.**
Rekursiv funktion der også viser attributter: ..... **Fejl! Bogmærke er ikke defineret.**
Visning af resultater i en browser: ..... **Fejl! Bogmærke er ikke defineret.**
Gennemløb af et XML dokument med løkker: ..... **Fejl! Bogmærke er ikke defineret.**
Skriv XML filer: ..... **Fejl! Bogmærke er ikke defineret.**
DOM metoder også i HTML: ..... **Fejl! Bogmærke er ikke defineret.**
Et eksempel på Java programmering af Xerces DOM parseren: ..... **Fejl! Bogmærke er ikke defineret.**
Xerces: En simpel parser i Java: ..... **Fejl! Bogmærke er ikke defineret.**
XPATH eksempel fra .NET: ..... **Fejl! Bogmærke er ikke defineret.**
Eksempel på DOM i .NET: XmlDocument: ..... **Fejl! Bogmærke er ikke defineret.**

DOM – Document Object Model:

DOM er en dokument standard som er udarbejdet af W3C Konsortiet. Standard dokumentet kan downloades fra http://www.w3.org. DOM er udgivet på tre forskellige niveauer eller **levels** men Level 2 er fortsat den gældende standard (Level 2 Document Object Model Core).

Der er meget materiale at hende på denne side om DOM modellen! Det er meget anbefalelsesværdigt at **downloade** specifikationen som definerer elementerne i DOM modellen!

W3C har defineret hvordan Java script eller ECMA script skal implementere DOM modellen. En sådan **language binding** eksisterer også for Java. Men **ikke** for andre programmeringssprog – som udtryk for den særlige forbindelse mellem Java og XML!

DOM specificerer at et DOM dokument skal have præcist een rod og være **velformet**. Et DOM dokument kan være f. eks. et HTML eller XML dokument.

DOM er beskrevet i et abstrakt sprog **IDL** – Interface Definition Language, et sprog som **abstrakt** definerer interfaces (objekter, metoder).

DOM er **uafhængigt** af operativ system og programmerings sprog. Helt konkret er W3C DOM specificeret i **OMGIDL** nemlig den IDL som er defineret af Object Management Group (OMG) som også definerer **CORBA** standarden.

DOM er grundlæggende det **samme** dokument format - uanset om det **implementeres** i Java, C, Perl eller Javascript! Heri ligger den store **fordel** i standarden. DOM kører stort set ens uanset om styre systemet er Windows, Linux, Mac eller Unix.

DOM definerer grundlæggende funktioner (metoder) som kan **læse** (parse) dokumenter, søge i disse dokumenter og **skrive** eller redigere dokumenter.

Vi vil i det følgende først se på hvordan DOM kan læse dokumenter, siden se på hvordan man med DOM metoder kan skrive dokumenter.

Et set af metoder kaldes et **API** – et Application Programming Interface. DOM er altså basalt en sprog- og platforms-uafhængig API. Med denne API kan man navigere rundt i et dokument og manipulere med et dokument.

Et <script> i en HTML fil (eller XML fil) er en **applikation** af DOM. Forudsætningen for at dette kan fungere er at operativ systemet har en **parser** som implementerer W3C standarden DOM. I det følgende vil vi se eksempler med JavaScript eller **ECMA** Script (en script standard fra European Computer Manufacturers Association) i MS Internet Explorer.

Internet Explorer har en indbygget parser der implementerer DOM. Hvis man har installeret Internet Explorer har man altså **allerede** installeret en XML parser! Den XML parser som IE6 anvender er msxml3.dll – altså Microsoft parseren i version 3. IE anvender altså ikke den sidste version 4 som kan downloades fra http://msdn.microsoft.com !

Der findes et utal af forskellige implementeringer af DOM i forskellige sprog og til forskellige styresystemer.

Grundlæggende kræver DOM at et dokument er et **træ** med (kun) een rod. En træ struktur er en **graf** af en bestemt type. En graf er blot et sæt af **knuder** eller **noder** som er forbundne. Et træ i DOM er en graf med netop een rod – documentElement - og sub noder sådan at et element eller et objekt **altid** har kun een **parent** eller **super**-objekt. Dette kan illustreres på følgende måde – en såkaldt **struktur** model:

1. dataliste
    a. person
        i. fornavn
    b. person
        i. fornavn
    c. person
        i. fornavn

Elementet **fornavn** har een parent eller forældre nemlig **person**. Alle **personer** har een parent nemlig **dataliste**. dataliste har **ingen** parent (dataliste er nemlig roden). person har kun een child

nemlig **fornavn**. dataliste har 3 børn og 3 børnebørn! De 3 personer er i DOM **siblings** eller søskende!

DOM metoder er grundlæggende **'tree walking'** metoder som f.eks. at finde alle børn eller finde parent eller finde næste søskende.

DOM opbygger **to** sæt af **linkede lister**: Den **ene** liste linker parent og child elementer. Den **anden** liste linker den ene søskende eller sibling til den næste! Det er derfor altid **effektivt** – hurtigt – i DOM at gå mellem parent og child og gå til **næste** søskende! Men det er ikke altid effektivt eller hurtigt at gå til **forrige** søskende – fordi systemet så skal begynde forfra i dokumentet og søge!

Et DOM dokument skal **mindst** indeholde eet rod-element (**evt** med et subtræ under sig) og kan – **uden** for denne rod! - indeholde en eller flere **kommentarer**, processing **instructions** og/eller maksimalt een **doctype**.

Dette er altså **ikke** noget gyldigt XML dokument:

<?xml version='1.0'?>

Et XML dokument **skal** indeholde et rod element eller en **container**!

I en vis forstand er et DOM dokument ikke et 'rent' træ fordi visse objekter kan anbringes **uden** for roden.

DOM er en **objekt** model som har en super klasse node og objekter som document, nodelist, namednodemap, processinginstruction osv som **arver** fra **node.**

Typen node er alles **super** klasse. Metoderne i node arves altså videre i alle sub klasserne. **Alt** i et dokument er en **node**. Træet består af noder. Et element er en node. En attribut er en node. En namespace erklæring er også en node som f. eks. i dette eksempel:

<data xmlns="uri:data">Dette er data.</data>

xmlns="uri:data" er ikke en attribut (!) til elementet data – men en selvstændig **namespace node** som begrebsmæssigt nærmest står **oven over** elementet data – fordi namespacet jo dækker hele subtræet data!

Typer af noder i træet:

Der findes to helt forskellige slags noder:

2. **strukturelle** noder som documenttype, processinginstruction osv
3. **indholds** noder som element, document, attribut, cdata osv

Derudover er der defineret 12 **typer** af noder med tilhørende ID numre af **W3C** konsortiet:

Prototype Object Node

The Node class has the following constants:
Node.ELEMENT_NODE
This constant is of type **Number** and its value is **1**.
Node.ATTRIBUTE_NODE
This constant is of type **Number** and its value is **2**.
Node.TEXT_NODE
This constant is of type **Number** and its value is **3**.
Node.CDATA_SECTION_NODE
This constant is of type **Number** and its value is **4**.
Node.ENTITY_REFERENCE_NODE
This constant is of type **Number** and its value is **5**.
Node.ENTITY_NODE
This constant is of type **Number** and its value is **6**.
Node.PROCESSING_INSTRUCTION_NODE
This constant is of type **Number** and its value is **7**.
Node.COMMENT_NODE
This constant is of type **Number** and its value is **8**.
Node.DOCUMENT_NODE
This constant is of type **Number** and its value is **9**.
Node.DOCUMENT_TYPE_NODE
This constant is of type **Number** and its value is **10**.
Node.DOCUMENT_FRAGMENT_NODE
This constant is of type **Number** and its value is **11**.
Node.NOTATION_NODE
This constant is of type **Number** and its value is **12**.

Reading:

Note: En **ulempe** ved at anvende DOM modellen er at **hele** dokumentet skal indlæses i maskinens hukommelse/RAM. Hvis der er tale om **store** XML dokumenter kan dette medføre nogen forsinkelse. Selv dokumenter på cirka 5 Mega Bytes kan give mærkbare forsinkelser. Vi skal siden i afsnittet om SAX se på hvordan man kan måle performance: d.v.s. hvor lang tid det tager at indlæse et XML dokument med DOM metoder og med SAX metoder!

Maskinen opbygger en **intern** DOM struktur i hukommelsen (RAM) som gør at man kan springe frit rundt i dokumentet. Denne struktur kan fylde **meget** mere end selve dokumentet! Dette har sine ulemper. Denne DOM struktur er et såkaldt virtuelt træ som kun eksisterer i hukommelsen. Når jeg f. eks. redigerer i dette virtuelle træ – redigerer jeg altså kun i adresser i hukommelsen eller RAM! Det jeg gør bliver ikke uden videre gemt i en fil!

Vi skal siden se at en ren parser som en **SAX** parser kan arbejde meget hurtigere end DOM. SAX er velegnet når man skal søge og finde nogle få værdier i et stort XML dokument. Ulempen ved SAX er at man **ikke** kan springe frit rundt i dokumentet og **ikke** kan modificere dokumentet! SAX læser simpelthen filen tegn for tegn **forfra** og **read** only! Den grundlæggende parser i .NET fra Microsoft er /stort set) en sådan SAX parser (XmlTextReader).

I det følgende skal vi se på hvordan man kan læse eller parse et XML dokument.

Vi vil her – i første omgang - anvende en såkaldt **data island** der ser sådan ud:

```
<xml id="xml_document">

<?xml version="1.0" encoding="iso-8859-1"?>
<!-- data islands tolererer ikke doctypes! -->
<bil_lager
firma='CBilerX.aps'
xmlns="uri:cbilerx.1.1"
>
<?instruktion Dette er en PI processing instruction?>
<!-- bil_lager lister alle biler paa lageret -->
<bil km="77567">
<id>b12345</id>
<pris xmlns="uri:cbilerx.1.1"
>12456</pris>
</bil>
<bil km="12447">
<id>b54321</id>
<pris xmlns="uri:cbilerx.1.1"
>13456</pris>
</bil>
</bil_lager>
</xml>

<script>

function parse_dom(){
//begge er OK:
//var doc=document.all("xml_document").XMLDocument;
var doc=xml_document.XMLDocument;
alert(doc.xml);

}
parse_dom();


</script>
```

I første omgang anvender vi DOM objektet **document**. En data **island** vil sige at XML dokumentet **indlægges** direkte i HTML dokumentet. Dette er meget anvendeligt hvis XML data **sendes** fra en server til en klient. **IE** kan direkte takle data islands. I andre browsere skal anvendes andre metoder – som vi skal se.

document:

I eksemplet kaldes et script som viser selve dokumentet:

I denne omgang – vi ser kun på DOM **reading** - de fleste metoder i document er **write** metoder! – kan vi anvende disse metoder i document:

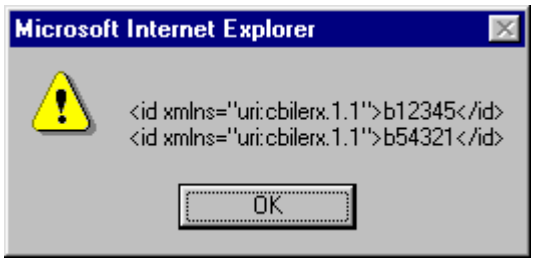
1. getElementsByTagname

Vi kan skrive dette script:

```
<script>

function parse_dom(){
//begge er OK:
//var doc=document.all("xml_document").XMLDocument;
var doc=xml_document.XMLDocument;
alert(doc.xml);

var ids=doc.getElementsByTagName("id");
var s="";
for(i=0;i<ids.length;i++) {
s+="\n"+ids(i).xml;
}
alert(s);

var priser=doc.getElementsByTagName("pris");
var str="";
for(i=0;i<priser.length;i++) {
str+="\n"+priser(i).xml;
}
alert(str);

}
parse_dom();
```

</script>

Resultatet er bl. a. disse bokse som returnerer to XML fragmenter:



Vores DOM metoder returnerer **alle** elementer af typerne pris og id.

element:

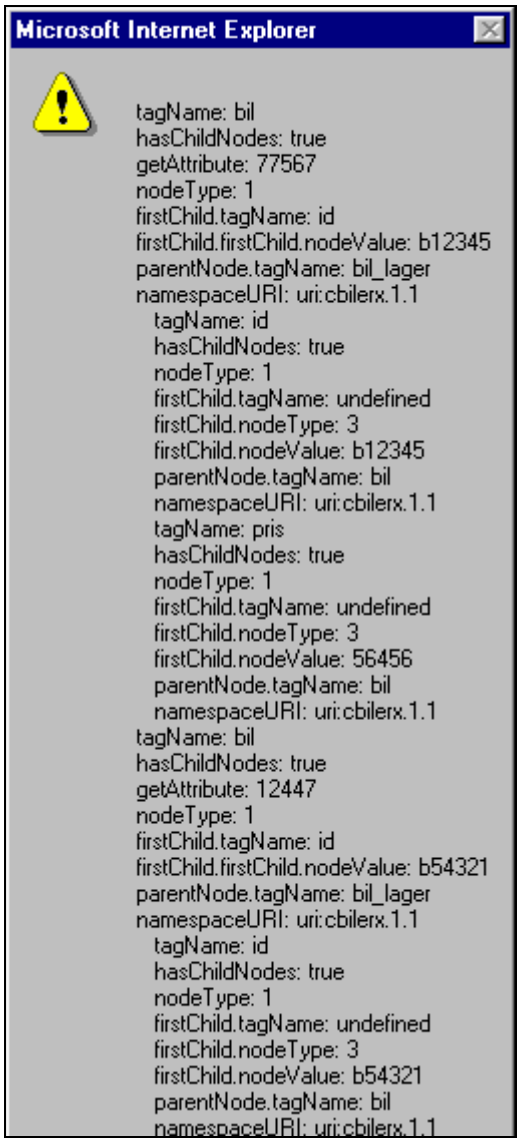
De fleste metoder og egenskaber findes i objektet element. I vores eksempel er pris og id eksempler på **element** noder.

Vi vil nu implementere en række **metoder** og **egenskaber** i objektet element:

- tagName
- getAttribute
- getAttributeNode
- getAttributeNodeNS
- getAttributeNS
- getElementsbyTagName
- getElementsbyTagNameNS
- hasAttributes
- hasAttributeNS
- nodeName
- nodeValue
- type
- parentNode
- childNodes
- firstChild
- lastChild

- previousSibling
- nextSibling
- attributes
- namespaceURI
- prefix
- localName
- hasChildNodes

Når vores script afvikles får vi dette billede hvor en række af DOM metoderne er anvendt:



Ved hjælp af DOM kan vi finde alle **elementer** af en vis type – her typen bil – og arbejde med disse objekter. Vi kan finde deres **sub** elementer, et evt **namespace** for elementet bil, objektets **parent** osv.

Vi kan teste om objektet har **attributter** eller **sub** elementer under sig. Det ses også at selve teksten i XML er en selvstændig specifik **tekst** node som er en **child** af det nærmeste element. I eksemplet er 'b54321' en **anonym** tekst node hvis **parent** er et 'almindeligt' navngivet element nemlig 'id'.

Selve id elementet har altså **INGEN** nodeValue – dets firstChild (tekst noden) har derimod. Denne firstChild – teksten – tilhører også en nodetype – nemlig 3. Type 1 er et element og type 2 er en attribut.

Metoderne childNodes og attributes returnerer en **collection** eller **liste** over en række node objekter. Derfor er det nødvendigt at anvende en løkke for at gennemgå disse objekter.

Vores script er skrevet således:

```
<script>

function parse_dom(){
//begge er OK:
//var doc=document.all("xml_document").XMLDocument;
var doc=xml_document.XMLDocument;
alert(doc.xml);

//alle elementer bil i dokumentet:
var data=doc.getElementsByTagName("bil");

//alle attributter i dokumentet:
//var data=doc.getElementsByTagName("@*");

//giver selve roden - kun eet element bil_lager!
//var data=doc.getElementsByTagName("bil_lager");

//giver alle elementer:
//var data=doc.getElementsByTagName("*");

var s="";

for(i=0;i<data.length;i++) {
s+="\n tagName: "+data(i).tagName;
s+="\n hasChildNodes: "+data(i).hasChildNodes;
s+="\n getAttribute: "+data(i).getAttribute('km');
s+="\n nodeType: "+data(i).nodeType;
s+="\n firstChild.tagName: "+data(i).firstChild.tagName;
s+="\n firstChild.firstChild.nodeValue: "+data(i).firstChild.firstChild.nodeValue;
s+="\n parentNode.tagName: "+data(i).parentNode.tagName;
s+="\n namespaceURI: "+data(i).namespaceURI;

var data1=data(i).getElementsByTagName('*');

for(j=0;j<data1.length;j++) {
s+="\n    tagName: "+data1(j).tagName;
s+="\n    hasChildNodes: "+data1(j).hasChildNodes;
s+="\n    nodeType: "+data1(j).nodeType;
s+="\n    firstChild.tagName: "+data1(j).firstChild.tagName;
s+="\n    firstChild.nodeType: "+data1(j).firstChild.nodeType;
s+="\n    firstChild.nodeValue: "+data1(j).firstChild.nodeValue;
```

```
s+="\n    parentNode.tagName: "+data1(j).parentNode.tagName;
s+="\n    namespaceURI: "+data1(j).namespaceURI;

}


}
alert(s);


}
parse_dom();


</script>
```

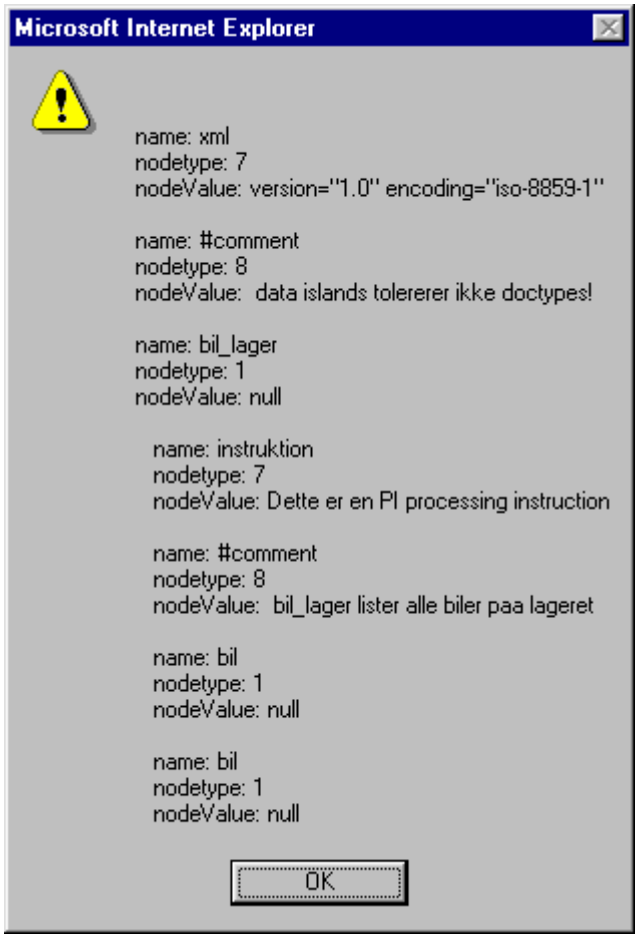
Selv om man ikke er verdensmester i scripting eller JavaScript kan man se at der er anvendt 2 **løkker** for at finde alle elementer **bil** og alle sub elementer under hver bil. På denne måde kan ethvert XML dokument parses og analyseres og man kan finde bestemte data!

```
var data1=data(i).getElementsByTagName('*');
```

Denne sætning går ud fra en bestemt **kontekst** node nemlig den **aktuelle** bil! Derfor virker metoden på denne **kontekst** node **ikke** på hele dokumentet!

En vigtig del af DOM er at man hele tiden arbejder ud fra en **kontekst** node som kan være hele dokumentet (som også er en node fordi alt er noder i XML!) eller som kan være et sub træ!

Hvis vi ser på dokumentet som et træ kan vi se at dokumentet her har 3 børn under root:



De tre **børn** er **xml** erklæringen, **kommentar** noden og **documentElement** nemlig noden bil**_lager**!

Neden under objektet bil**_lager** finder vi 4 børn: **instruktions** noden (en processing instruction), en **kommentar** node og 2 **bil** noder! Hver bil har så igen noder under sig - som vi så tidligere!

Det ses at **alt** er noder eller node-objekter i DOM!

Vores script til at vise dette hierarki er:

```
var data=doc.childNodes;

var s="";

for(i=0;i<data.length;i++) {
   s+="\n\n name: "+data(i).nodeName;
   s+="\n nodetype: "+data(i).nodeType;
   s+="\n nodeValue: "+data(i).nodeValue;

      for(j=0;j<data(i).childNodes.length;j++) {
      s+="\n\n    name: "+data(i).childNodes(j).nodeName;
      s+="\n    nodetype: "+data(i).childNodes(j).nodeType;
      s+="\n    nodeValue: "+data(i).childNodes(j).nodeValue;
```

```
        }

    }
alert(s);
```

Hele dokumentet – som alle DOM dokumenter - er opbygget af **childNodes** og **relationer** parent – child. Den træ struktur som DOM opbygger i hukommelsen rummer netop en samling af links eller såkaldt linkede lister parent-child. En linked liste har en pointer (en pegepind) til next og previous i en liste. Dette ses tydeligt i en linked liste af siblings eller søskende. For hvert element kan man springe direkte til den næste søskende (og til den forrige)!

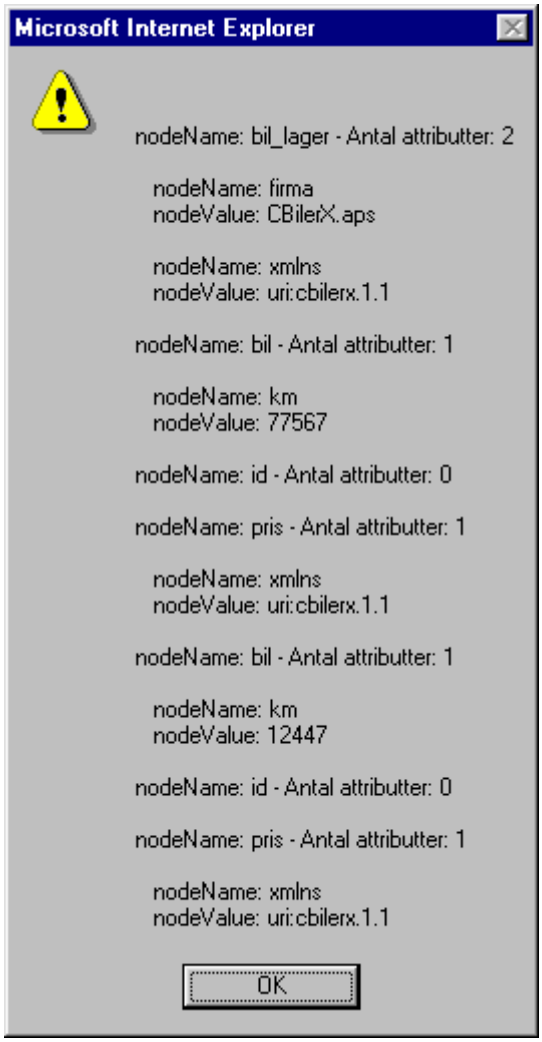
attributter:

Et elements **attributter** findes **ikke** ved childNodes metoden. Et elements attributter er **ikke** børn af elementet men data som beskriver elementet og er en del af elementet. Man kan også forestille sig en attribut i et element som en supernode som ligger oven over elementet og gælder for hele sub træet – ligesom en namespace node gælder for hele sub træet!

Bemærk også at attributter kan skrives i en **vilkårlig** rækkefølge – modsat elementerne normalt i et sub træ! Sættet af et elements attributter kaldes en N**amed** N**ode** M**ap** og er en **hash** tabel som består af par af **name** og **value:** f.eks. opdateret=”2003-03-15”. Name eller **key** er her ’opdateret’ og value er ’2003-03-15’!

Attributter er **noder**, men **ikke** elementer!

I vores eksempel har elementet bil_lager f.eks. 2 attributter.



parser fejl:

OBS: Hvis der er formelle fejl i XML dokumentet (ikke vel formet) i en data **island** kommer der INGEN fejl melding! Metoderne returnerer blot en tom streng! Prøv fx at ændre et 'id' til et 'idd'!

XPATH:

Stier i XPATH:

Når vi **navigerer** rundt i et DOM dokument bruges et simpelt og ikke XML baseret **sprog** XPATH. XPATH specifikationen kan også hentes på adressen http://www.w3.org.

XPATH minder om de **stier** som bruges om fil og mappe systemet på en computer! Hvis vi forestiller os denne sti på computeren:

c:\windows\notepad.exe

har vi en klar fornemmelse af at vi taler om **objektet** notepad.exe som ligger i en **mappe** (en **container**) der hedder **windows** som igen ligger på C drevet!

Stier i XPATH fungerer på samme måde – bortset fra anvendelsen af nogle finesser som **prædikater** - som vi skal se!

Nogle vil måske se at XPATH minder om **SQL**: I SQL kan man anvende et udtryk som: select * from tabel where fornavn='erik'! Her sættes først en **kontekst** nemlig alle felter i tabellen tabel og derefter undersøges det punkt for punkt hvilke poster der tilfredsstiller kravet: at fornavn skal være 'erik'!

XPATH arbejder på den måde hele tiden med et nodesæt eller 'set' og med om en betingelse er opfyldt eller ikke opfyldt – sand eller falsk!

. (et punktum) betyder det nuværende objekt – lige som på computeren hvor punktum betyder den nuværende mappe
.. (to punktummer) betyder **parent** eller 'gå et niveau op i træet'!

XPATH **stier** anvender overalt i XML – ikke mindst i **stylesheets** eller **typografi** ark i XSLT. Vi skal se mange eksempler på XPATH stier i dette kursus – så her kommer en kort sammenfatning:

En sti kan være **relativ** eller **absolut**. En **relativ** sti er f. eks.:

person/fornavn

Denne sti består af 2 **trin** – trin bliver altid adskilt af '/'. XPATH finder – ud fra den givne **kontekst**! – alle de sub elementer eller børn – under den aktuelle kontekst node! – som hedder person og i dette sæt går XPATH så videre og finder alle de sub elementer som hedder fornavn!

En sti som denne er **relativ** fordi den afhænger af konteksten! Måske er der slet ingen børn der hedder person! Dette svarer fuldstændigt til at jeg f. eks. står i mappen Windows og vil skifte til en under mappe ved navn olenyborg – men denne mappe eksisterer måske slet ikke som undermappe under kontekst mappen!

XPATH tager trin for trin – som forklaret.

Et eksempel på en **absolut** sti ville være:

/telefonliste/person/fornavn

En absolut sti starter altid med '/' som betyder **roden** af dokumentet. Derefter følger en relativ sti. En absolut sti er altså **uafhængig** af enhver kontekst!

Prædikater:

Et prædikat er et yderligere søge kriterium – f. eks. kan vi søge efter en bestemt person:

/telefonliste/person/fornavn[.=’Marie-Louise’]

Her gennemløbes samtlige personer og deres sub elementer fornavn indtil systemet finder (eller ikke finder) en person med fornavnet ’Marie-Louise’!

. (punktum) bruges altid som det nuværende objekts tekst(node).

Hvis man vil finde det første eller sidste eller 7. objekt kan man bruge en index operator:

/telefonliste/person[1]
/telefonliste/person[position()=last()]
/telefonliste/person[7]
/telefonliste/person[last() - 1]

Operatorer:

I øvrigt anvendes en række operatorer i XPATH til at definere en bestemt sti til et objekt:

// bruges til at finde en node **uanset** hvor den findes i træet. F. eks. stien //* betyder find alle elementer (og kun elementer!) - idet * bruges om alle uanset hvad de hedder!

//fornavn finder alle fornavne uanset hvor de findes.

//@* betyder: Find alle attributter – idet @ bruges om en attribut.

//@opdateret finder alle attributter i dokumentet som hedder opdateret.

Funktioner i XPATH:

Der findes en række **funktioner** i XPATH og vi vil bruge dem adskillige gange igennem kurset! Eksempler på funktioner som er forud definerede er:

1. position()
2. last()
3. first()
4. count()
5. id()
6. local-name()
7. name()
8. namespace-uri()
9. concat()
10. contains()
11. starts-with()
12. string-length()
13. substring()
14. substring-before()

15. substring-after()
16. translate()
17. number()
18. sum()
19. round()

De forskellige parsere har så implementeret endnu flere **extension** eller udvidede funktioner! Vi skal gennem kurset se de fleste af disse funktioner brugt - især i XSLT stylesheets!

Som et demo eksempel kan vi se på følgende eksempel der anvender nogle XPATH funktioner:

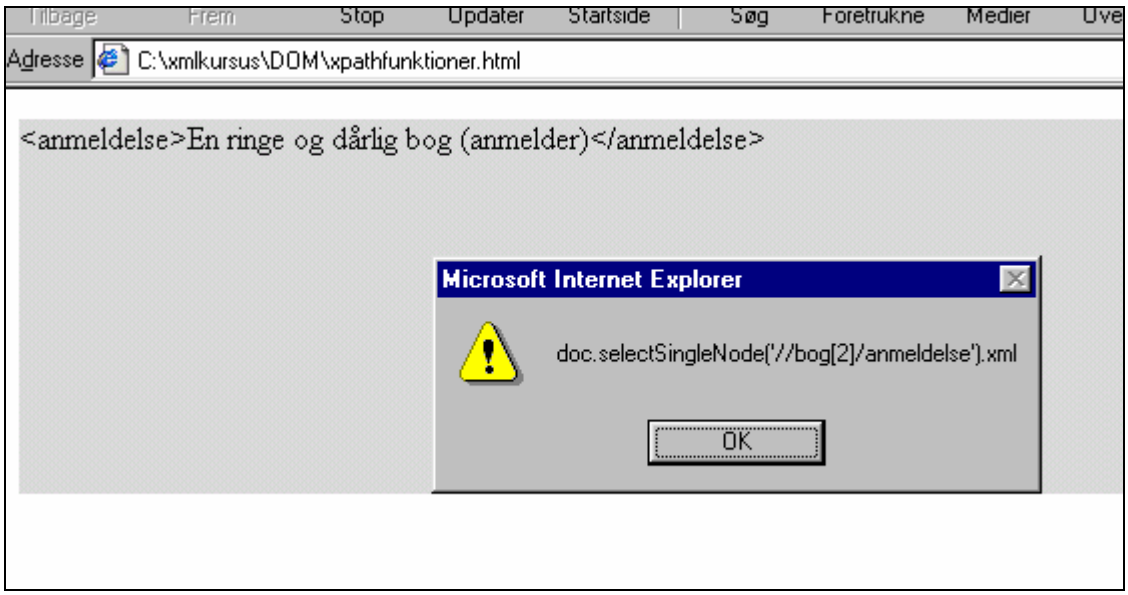
I dette script anvendes ikke en data island. XML dokumentet loades direkte fra en lokal fil på maskinen. Dette sker ved at vi instantierer et objekt **DOMDocument**.4.0!

Derefter kan vi sætte dette objekts egenskaber (**properties**) og anvende dets **metoder**. Hvis vi sætter **validateOnParse** til true vil parseren validere dokumentet – hvis der er et skema! Hvis vi sætter **async** til false betyder det at XML dokumentet loades synkront d.v.s. at vi 'venter' til at det er fuldt loadet inden vi går videre!

DOMDocument er en Microsofts DOM **parser**!

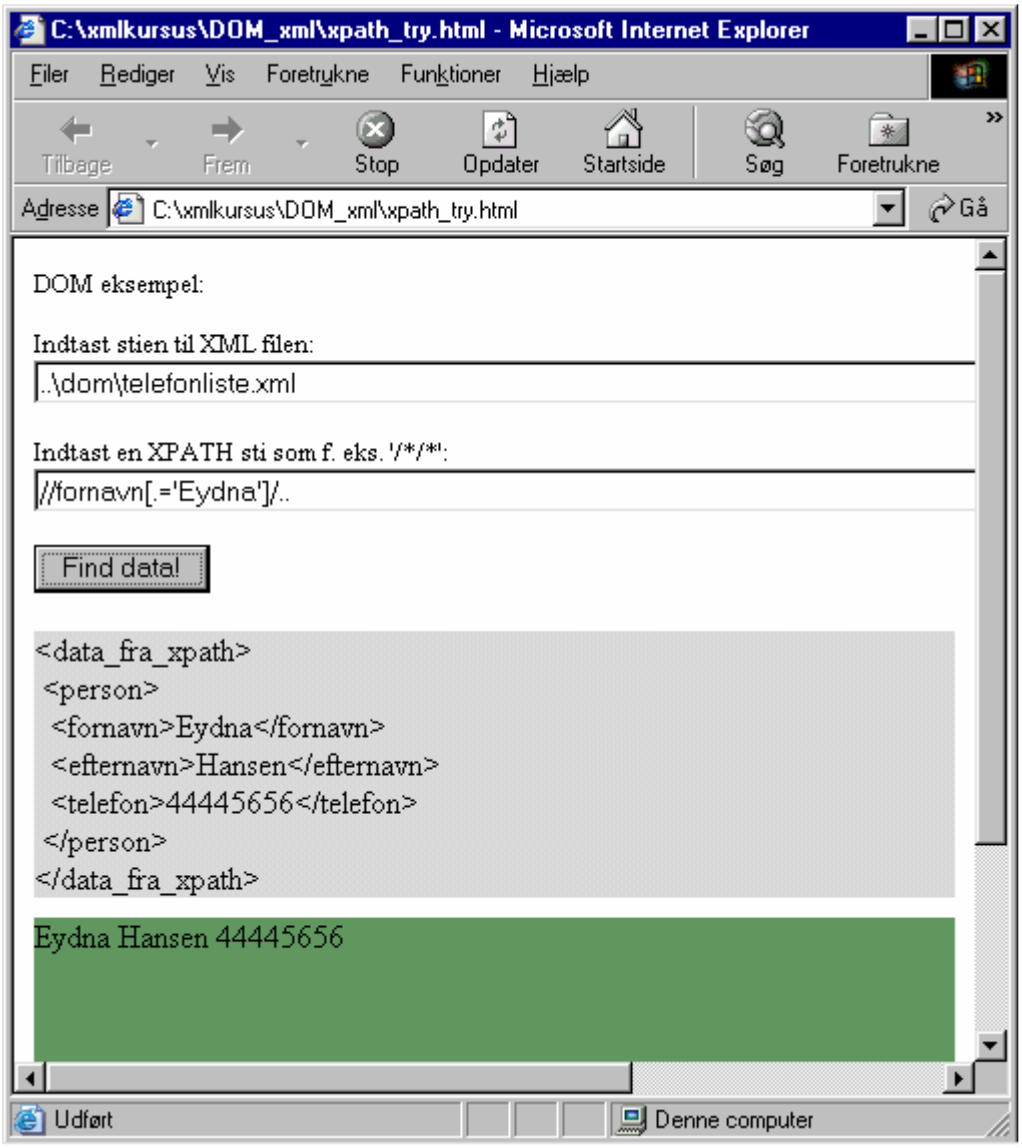
```
<body >
<div id="resultat" style="margin-bottom:10px;background-color:#dddddd;font-size:12pt;color:black;height:140pt">
</div>
</body>

<script>
load_doc();

function load_doc(){
//nulstil div:
resultat.innerText="";
try{
var doc=new ActiveXObject("MSXML2.DOMDocument.4.0");
doc.async=false;
doc.load("boghandel.xml");

if(doc.parseError.errorCode==0){
                    resultat.innerText=doc.xml;
                    alert("Forskellige XPATH eksempler.");

                    //XPATH funktioner eksempler:

                    resultat.innerText=doc.selectSingleNode("//bog/fornavn").xml;
                    alert("doc.selectSingleNode('//bog/fornavn').xml");

                    resultat.innerText=doc.selectSingleNode("//bog[2]/anmeldelse").xml;
                    alert("doc.selectSingleNode('//bog[2]/anmeldelse').xml");

                    resultat.innerText=doc.selectSingleNode("//bog/efternavn[.='Olsen']/..").xml;
                    alert("doc.selectSingleNode(//bog/efternavn[.='Olsen']/..).xml");

                    resultat.innerText=doc.selectSingleNode("//@*").xml;
                    alert("doc.selectSingleNode('//@*').xml");
```

```
                resultat.innerText=doc.selectSingleNode("/NewDataSet/*[3]").xml;
                alert("doc.selectSingleNode('/NewDataSet/*[3]').xml");

}
else alert ("Fejl: "+doc.parseError.reason);


}
catch(e) {alert("Fejl: "+e);}
}
</script>
```



XPATH test program:

Vi kan **teste** forskellige stier og udtryk i XPATH med et lille script program der ser sådan ud:



I programmet viser den øverste division selve XML teksten, den nederste viser denne tekst som HTML.

Det viste XPATH **eksempel** gør følgende:

1. Find alle fornavn elementer hvor de end befinder sig i træet
2. Find det **første** af dem hvis tekst node er lig med 'Eydna'
3. Gå derefter op til parent for denne node!

Koden ser således ud:

```
<body >

<form id="f" name="f">
DOM eksempel:<br>
<br>Indtast stien til XML filen:
```

```
<br><input type="text" value="..\dom\telefonliste.xml" size="100" id="fil" name="fil" />
<br>Indtast en XPATH sti som f. eks. '/*/*':
<br><input type="text" value="/child::*" size="100" id="path" name="path" />
<br><input type="button" value="Find data!" onClick="load_doc()" />
</form>

<div id="resultat" style="margin-bottom:10px;background-color:#dddddd;font-size:12pt;color:black;"> </div>
<div id="resultat1" style="background-color:#669966;font-size:12pt;color:black;height:140pt"> </div>

</body>

<script>

function load_doc(){
  //nulstil div:
  resultat.innerText="";
  try{
  var doc=new ActiveXObject("MSXML2.DOMDocument.4.0");
  var doc1=new ActiveXObject("MSXML2.DOMDocument.4.0");
  doc1.async=false;
  doc.async=false;
  var filsti=document.all("fil").value;
  doc.load(filsti);
  if(doc.parseError.errorCode==0){
      var sti=document.all("path").value;

      //opret et 'xml fragment' med en kunstig root:
      var xml_fragment=doc.selectSingleNode(sti).xml;
      doc1.loadXML("<data_fra_xpath>\n"+xml_fragment+"\n</data_fra_xpath>");
      resultat.innerText=doc1.xml;

//XSLT:

//var docxsl=new ActiveXObject("MSXML2.DOMDocument.4.0");
//docxsl.async=true;
//docxsl.load ("kun_node_txt1.xsl");
//(transformation her)

resultat1.innerHTML=doc1.xml;

  }
  else alert ("Fejl: "+doc.parseError.reason);


  }
  catch(e) {alert("Fejl: "+e);}
}
</script>
```

Som antydet kan man **transformere** eller formatere XML dokumentet (dvs. her: XML **fragmentet**!) med et XSLT typografiark, stylesheet. Det er kommenteret ud her og vi skal siden vende tilbage til hvordan det gøres.

Vi kan få vist elementernes tekstnoder (ikke attributternes!) ved simpelthen at kalde **innerHTML** som vist! **innerText** er derimod nødvendig når vi ønsker at vise selve XML teksten!

Man kan også finde lignende XPATH test programmer på Internettet! Et andet sted i kurset er omtalt at hvis man downloader Xerces parseren fra apache.org får man også et lille Java program der fungerer til test af stier i XPATH.