SAX – Simple API for XML.

En API (Application Programming Interface) et bibliotek eller et sæt af funktioner eller metoder. SAX er et sådant bibliotek af abstrakte metoder som f. eks. startDocument() eller startElement(). SAX API er ikke noget program men en standard som så er blevet implementeret af mange forskellige SAX parsere f. eks. Microsofts MSXML eller Xerces.

SAX blev oprindeligt – i 1998 - skrevet for sproget Java men er siden implementeret i næsten alle de anerkendte programmerings sprog.

SAX er en **push** model: d.v.s. at når SAX parseren læser XML dokumentet læser den hele filen tegn for tegn og sender det hele videre. Man kan altså ikke i SAX vælge kun at læse dele af XML dokumentet. Det hele bliver pushet videre til den applikation som bruger SAX parseren. Dette er i modsætning til en såkaldt **reader** som normalt kan vælge at læse f. eks. kun de første 10 tegn i en fil!

SAX er en **event** baseret model på den måde at hver gang SAX parseren støder på et bestemt objekt affyrer den en event som så sendes videre til det program som bruger SAX parseren som et redskab. En event kan f. eks. være startElement som SAX affyrer når den f.eks. støder på disse tegn i filen (NB det som SAX parseren læser er jo simpelt hen karakter koder som #97 (’a’) eller #0A (linjeskift):

<parcel_hus>...

DOM modellen – en **pull** model – er det modsatte af SAX modellen. I DOM læses hele filen, der opbygges et binært DOM træ i RAM (i hukommelsen). I SAX læses eller parses dokumentet simpelthen tegn for tegn:

1. SAX kan ikke springe rundt i dokumentet, kan ikke gå tilbage
2. SAX kan ikke huske hvad der tidligere er læst
3. SAX opbygger intet træ eller nogen RAM struktur

Af disse grunde er det oftest meget **hurtigere** og mere effektivt at læse et XML dokument med SAX end med DOM. Især hvis XML dokumentet er stort – som på 25 MegaBytes! – fungerer SAX meget hurtigere end DOM. Hvis man skal finde nogle få data i et meget stort XML dokument er SAX meget hurtigere end DOM.

I SAX kan man begynde at bruge data som ER afsendt fra SAX parseren – også selv om den endnu ikke har læst hele dokumentet!

SAX – modsat DOM – er ikke en W3C standard men et privat initiativ – skabt af amerikaneren Megginson - som siden er blevet bredt anerkendt.

Men SAX har som nævnt sine begrænsninger. SAX affyrer sine evnts i ’real time’ mens DOM arbejder med en opbygget træ struktur som kan anvendes igen og igen når den først er opbygget.

Hvis der skal foretages lidt mere indviklede søgninger i dokumentet er DOM modellen klart overlegen. Dette gælder også hvis der skal foretages mere omfattende redigeringer i dokumentet!

Funktioner:

Alle SAX parsere **skal** implementere (definere) disse funktioner eller metoder:

1. characters() som resturnerer nodens tekst
2. startDocument() og endDocument()
3. startElement() og endElement()
4. ignorableWhitespace() som definerer hvad der skal ske med mellemrum, tabulator og linjeskift
5. processingInstruction()
6. startPrefixMapping() og endPrefixMapping() som drejer sig om namespaces f. eks. i et element
7. skippedEntity()

En konkret SAX parser kan så definere flere supplerende metoder.

Selve SAX definitionen (standarden) kan downloades fra http://sax.sourceforge.net. Der findes både en SAX1 og en SAX2, men kun den sidste version fra år 2000 – med namespaces – bruges nu.

Vi kan i nogen grad anvende SAX parsing i scripts. Her er et eksempel på et script som anvender SAX og læser et meget stort dokument og viser det på siden:

```
<body>
<div id="resultat" name="resultat" style="background-color:#dedede"></div>

<script>
function f(){
var reader = new ActiveXObject("MSXML2.SAXXMLReader.4.0");
var writer = new ActiveXObject("MSXML2.MXXMLWriter.4.0");
reader.contentHandler=writer;
try {

   //fra harddisken:
   //reader.parseURL ("c:/xmlkursus/SAX/meget_lang1.xml");

   //SAX kan hente en URL:
   reader.parseURL ("http://localhost/xml/meget_lang1.xml");

   //alert("SAX OK");
   //alert(writer.output);
   document.all("resultat").innerText=writer.output;
}
catch(e) {
   alert("Parsing med SAX Ikke OK: " + e.description);
}
}
f();
</script>
```

```
</body>
```

Vi anvender – instantierer – her en SAX reader og en SAX writer (nemlig de to klasser fra Microsofts MSXML2, altså Microsofts SAX parser).

En SAX **reader** skal have en såkaldt **content** handler som er en **event** handler. En event handler er en klasse som implementerer det abstrakte interface ContentHandler og definerer de ovennævnte obligatoriske metoder! Hvis der skal ske noget når vi parser et XML dokument med SAX er det altså nødvendigt at SAX parseren (reader) har en event handler, en content handler! Ellers sker der ingen ting – ingen events bliver (reelt) affyret – i al fald er der **ingen** som **lytter** til disse events! - under læsningen.

Ovennævnte eksempel bruger det **trick** at writer'en sættes til at være **content handler** til reader'en! D.v.s. at de events som reader'en affyrer **modtages** og bearbejdes af writer'en – dvs. reelt udskrives! Dette kan lade sig gøre fordi en SAX writer implementerer det abstrakte interface ContentHandler.

Bagefter kan vi så bruge writer'ens output - som er produceret af writer'en ud fra de inputs den får fra reader'en!

Denne metode er den **hurtigste** måde man kan læse et evt. meget stort XML dokument på! Meget hurtigere end at anvende DOM metoder.

På denne måde kontrolleres **også** om dokumentet er vel formet! SAX parseren er en ægte **parser** lige som en DOM parser: Den kontrollerer om XML dokumentet er velformet – ellers stopper læsningen!

Hvis dokumentet ikke er vel formet kommer en fejl meddelelse:

Dette kan lade sig gøre ved at anvende en try ... catch konstruktion i scriptet. Parseren stopper så **første** gang den støder på et element som ikke er velformet! Den kaster en Exception!

Skrive XML dokumenter med SAX writer:

Vi kan producere – f. eks. store - test XML filer på denne måde:

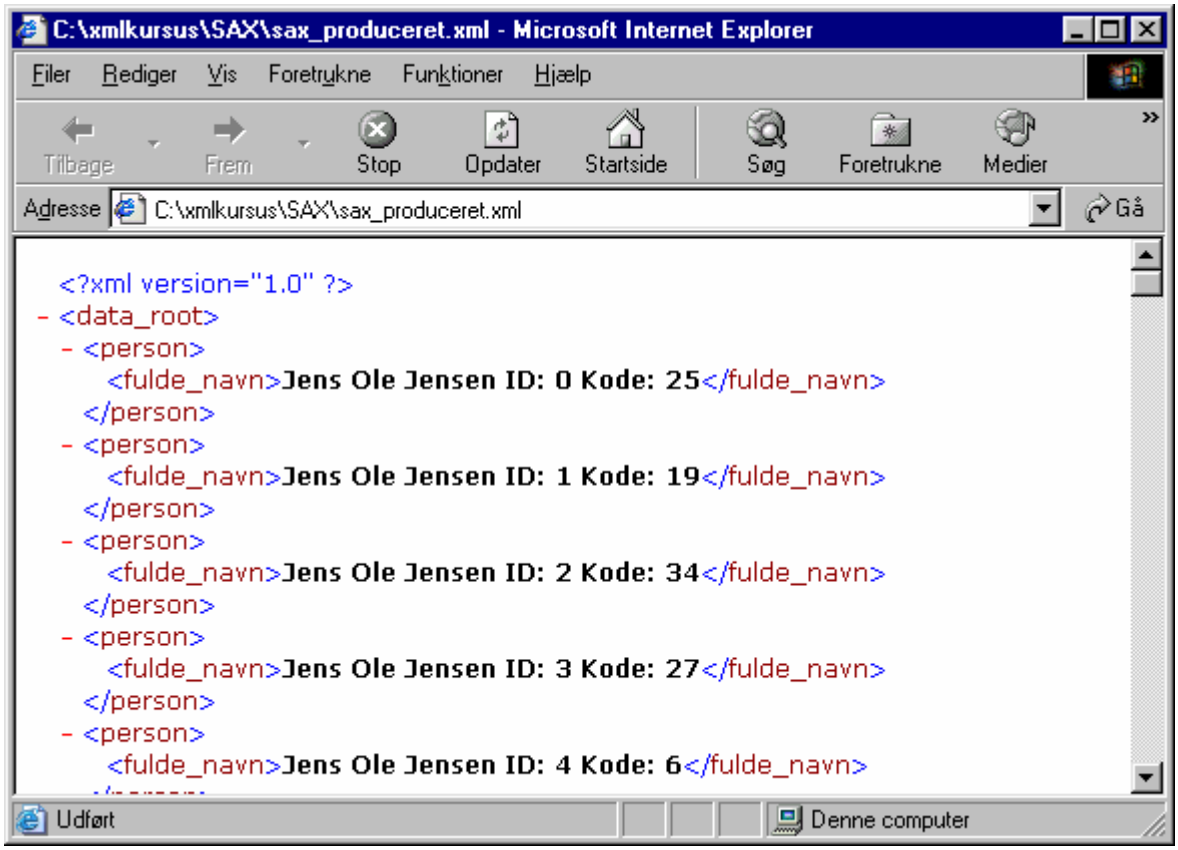
```
<script language="javascript">

function f(){
var writer = new ActiveXObject("MSXML2.MXXMLWriter.4.0");
```

```
var attr= new ActiveXObject("MSXML2.SAXAttributes.4.0");
var fso = new ActiveXObject("Scripting.FileSystemObject");
var fil=fso.createTextFile("c:/xmlkursus/SAX/sax_produceret.xml");
writer.indent=true;
writer.standalone=true;

//writer.startDocument(); - giver problemer med encoding!
writer.startElement ("","","data_root",attr);

for(i=0;i<100;i++) {
var rnd1=parseInt(Math.random()*100);
   writer.startElement ("","","person",attr);
   writer.startElement ("","","fulde_navn",attr);
   writer.characters("Jens Ole Jensen ID: "+i+" Kode: "+rnd1);
   writer.endElement ("","","fulde_navn");
   writer.endElement ("","","person");
}
writer.endElement ("","","data_root");

//writer.endDocument();
fil.write(writer.output);
fil.close();
}
f();
</script>
```

Vi skal senere se på andre eksempler på hvordan man kan generere XML dokumenter med SAX. Her ses det at SAX metoden **characters**() blot tager en streng som parameter, mens **startElement**() tager disse parametre:

1. namespace URI
2. local name
3. qualified name
4. et attribut objekt

Denne tilfældigt genererede XML fil gemmes her på harddisken. Den kan vises således i en browser:



SAX metoder i Visual Basic:

For at kunne udnytte mulighederne i SAX i højere grad er det nødvendigt at skrive kode i et programmeringssprog som Java, C, C++ eller Visual Basic.

Vi vil i det følgende illustrere dette ved at skrive nogle enkle eksempler i Visual Basic (VBA) som – evt. kan køre i VBA – i Microsoft Office. Man kan starte MS Excel eller Access eller et andet Office program og vælge Funktioner -> Makro -> Visual Basic editor. Her kan man så oprette nye klasser og moduler (så kaldte subs) som kan udnytte SAX. Det er vigtigt at sætte en reference til MSXML ved at vælge Tools (Funktioner) -> References og vælge Microsoft XML version 4.

Vi kan skrive et nyt meget enkelt modul (en ny sub rutine) således:

```
Option Explicit

Public Sub parse_sax()
Dim reader

    Set reader = New SAXXMLReader40
    Set reader.errorHandler = New error_handler
    reader.parseURL "bog.xml"
    On Error Resume Next
    MsgBox "XML Dokumentet er parset OK af SAX!"
```

```
End Sub
```

De centrale linjer i denne kode er:

```
Set reader.errorHandler = New error_handler
reader.parseURL "bog.xml"
```

error handler:

I Visual Basic skal vi **instantiere** en SAX parser (reader) – en SAXXMLReader - på denne måde. Hvis vi ønsker at parseren skal gøre noget - hvis den støder på et ugyldigt XML dokument - skal vi skrive en error_handler, som er en event handler som aktiveres hvis der findes fejl!

En SAX reader har en property (egenskab) ved navn **errorHandler** som sættes lig med en ny instantiering af vores egen klasse error_handler!

Vores egen error_handler **skal** indeholde 3 metoder fordi den skal implementere et bestemt interface nemlig interfacet ErrorHandler. Alle error handlers skal altså have disse tre metoder!

Vores klasse skrives i et klasse modul ved navn 'error_handler':

```
Option Explicit
Implements IVBSAXErrorHandler

Private Sub IVBSAXErrorHandler_fatalError(ByVal locator As IVBSAXLocator, _
                        msg As String, ByVal errCode As Long)

MsgBox "Error: Fejl: Linje: " & locator.lineNumber & " Kolonne: " & _
locator.columnNumber & " " & msg & " publicId: " & locator.publicId _
& " systemId: " & locator.systemId


End Sub

Private Sub IVBSAXErrorHandler_error(ByVal locator As IVBSAXLocator, msg As _
                            String, ByVal errCode As Long)
MsgBox "Error: Fejl: Linje: " & locator.lineNumber & " Kolonne: " & _
locator.columnNumber & " " & msg & " publicId: " & locator.publicId _
& " systemId: " & locator.systemId


End Sub

Private Sub IVBSAXErrorHandler_ignorableWarning(ByVal oLocator As _
                MSXML2.IVBSAXLocator, strErrorMessage As String, _
                ByVal nErrorCode As Long)

End Sub
```

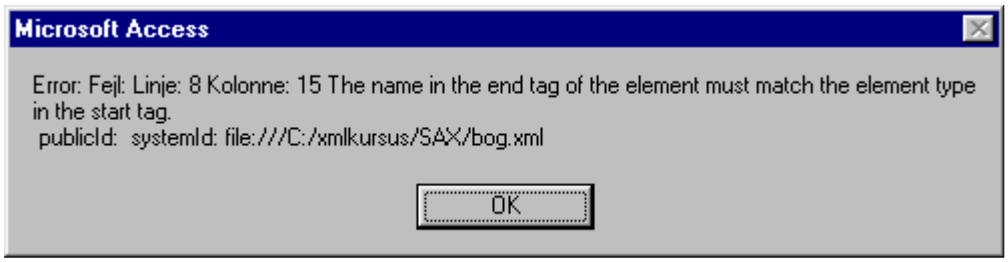
Hele dette system kan virke besværligt men i SAX er der stort set kun brug for en content handler og en error handler. Og disse eksempler kan man så blive ved med at genbruge – så arbejdet er ikke så stort!

En SAX Locator er et objekt som hele tiden registrerer i hvilken linje og kolonne reader'en eller parseren befinder sig!

systemId er den fil (adresse) som XML dokumentet stammer fra.

Vi kan se at vores error_handler implementerer de 3 metoder. Hvis der opstår fejl i XML dokumentet skal der vises en message box med alle oplysninger om fejlen! I dette tilfælde har vi lavet en bevidst fejl:

```
<T>
   <id>1</idd>
   <fornavn>jens</fornavn>
   <efternavn>jensen</efternavn>
   <titel>sommer i vejle</titel>
   <anmeldelse>en rigtig god bog - forfatteren har et stort potentiale og vil sikkert komme mere og
mere frem i de kommende år - hvem ved hvad det skal blive til - det bliver i al fald spændende at
følge hans udvikling - der vil komme mange gode bøger fra den mand
</anmeldelse>
  </T>
```



En content handler:

På samme måde som en SAX parser eller reader har en errorHandler egenskab har den en contentHandler egenskab (property).

Det egentlige arbejde er at definere en content handler til reader'en! Her definerer vi hvad vi vil med vores SAX program. En content handler er en event handler som **skal** rumme de efterfølgende metoder eller events:

```
Option Explicit
Implements IVBSAXContentHandler


Public s As String

Private Sub IVBSAXContentHandler_characters(strChars As String)
```

```
'den eneste event som implementeres her:
s = s & strChars

End Sub
Private Sub IVBSAXContentHandler_endDocument()

MsgBox "Tekster:" & vbNewLine & s

End Sub




Private Sub IVBSAXContentHandler_startElement(strNamespaceURI As _
            String, strLocalName As String, strQName As String, _
            ByVal oAttributes As MSXML2.IVBSAXAttributes)




End Sub

Private Sub IVBSAXContentHandler_endElement(strNamespaceURI As String, _
                        strLocalName As String, strQName As String)



End Sub


Private Property Set IVBSAXContentHandler_documentLocator(ByVal RHS _
                                    As MSXML2.IVBSAXLocator)

End Property


Private Sub IVBSAXContentHandler_endPrefixMapping(strPrefix As String)

End Sub

Private Sub IVBSAXContentHandler_ignorableWhitespace(strChars As String)

End Sub
```

```
Private Sub IVBSAXContentHandler_processingInstruction(strTarget As _
                                String, strData As String)


End Sub

Private Sub IVBSAXContentHandler_skippedEntity(strName As String)

End Sub

Private Sub IVBSAXContentHandler_startDocument()

End Sub

Private Sub IVBSAXContentHandler_startPrefixMapping(strPrefix As String, _
                                strURI As String)

End Sub
```

Vi kan her se **alle** de metoder som skal indgå i SAX! Den eneste metode vi har defineret er characters() som kaldes hver gang parseren støder på en tekstnode.

Vi kan nu sætte en content handler på vores reader. Klassen ovenfor kalder vi for ’c1’ f.eks. Vi kan så skrive denne sub:

```
Option Explicit

Public Sub parse_sax()
Dim reader

   Set reader = New SAXXMLReader40
   Set reader.errorHandler = New error_handler
   Set reader.contentHandler = New c1
   reader.parseURL "personliste.xml"
   On Error Resume Next
   MsgBox "XML Dokumentet er parset OK af SAX!"

End Sub
```

Vi har nu fået vores reader til at lave noget fornuftigt nemlig udskrive de tekst noder som findes i dokumentet. Vores XML eksempel ser således ud:

```
<?xml version='1.0'?>
<personliste>
<person>
```

<fornavn>Ole</fornavn>
<efternavn>Jensen</efternavn>
<telefon>38882222</telefon>
</person>
<person>
<fornavn>Lise</fornavn>
<efternavn>Jensen</efternavn>
<telefon>23233445</telefon>
</person>
</personliste>

Hvis vi afvikler vores sub rutine får vi følgende:

Vi kan også søge og finde data med SAX og det er netop her SAX har sine fordele frem for DOM. Hvis vi ønsker at finde alle fornavne kan vi omskrive vores c1 content handler således:

```
Option Explicit
Implements IVBSAXContentHandler


Public s As String
Dim fornavn As Boolean


Private Sub IVBSAXContentHandler_characters(strChars As String)

'den eneste event som implementeres her:
If fornavn = True Then
s = s & strChars
End If


End Sub
Private Sub IVBSAXContentHandler_endDocument()
```

```
MsgBox "Resultat:" & vbNewLine & vbNewLine & s

End Sub




Private Sub IVBSAXContentHandler_startElement(strNamespaceURI As _
        String, strLocalName As String, strQName As String, _
        ByVal oAttributes As MSXML2.IVBSAXAttributes)

        fornavn = False

        If strLocalName = "fornavn" Then
        fornavn = True
        End If




End Sub

Private Sub IVBSAXContentHandler_endElement(strNamespaceURI As String, _
                strLocalName As String, strQName As String)



End Sub


Private Property Set IVBSAXContentHandler_documentLocator(ByVal RHS _
                        As MSXML2.IVBSAXLocator)

End Property


Private Sub IVBSAXContentHandler_endPrefixMapping(strPrefix As String)

End Sub

Private Sub IVBSAXContentHandler_ignorableWhitespace(strChars As String)
```

```
End Sub

Private Sub IVBSAXContentHandler_processingInstruction(strTarget As _
                                String, strData As String)


End Sub

Private Sub IVBSAXContentHandler_skippedEntity(strName As String)

End Sub

Private Sub IVBSAXContentHandler_startDocument()

End Sub

Private Sub IVBSAXContentHandler_startPrefixMapping(strPrefix As String, _
                                strURI As String)

End Sub
```

Vi opretter en variabel en **boolean** fornavn som sættes til **true** hvis elementets navn er 'fornavn' – dette sker i den event som hedder startElement!

Herefter kan vi opsamle tekst noderne lige så længe fornavn er true!

Vi kan også tælle antallet af elementer i dokumentet meget enkelt således idet vi kun skriver en startElement definition:

```
Private Sub IVBSAXContentHandler_startElement(strNamespaceURI As _
        String, strLocalName As String, strQName As String, _
        ByVal oAttributes As MSXML2.IVBSAXAttributes)

        antal = antal + 1

End Sub
```

Private Sub IVBSAXContentHandler_endDocument()

MsgBox "XML dokumentet indeholder " & antal & " elementer."

End Sub

At validere et XML dokument med SAX:

Vi kan validere XML dokumenter i forhold til et skema på denne måde:

```
Option Explicit
Dim reader, skema_cache

Public Sub valider_dokument()

    Set reader = New SAXXMLReader40
    Set skema_cache = New MSXML2.XMLSchemaCache40
    skema_cache.Add "", "bog.xsd"
    reader.putFeature "schema-validation", True
    reader.putProperty "schemas", skema_cache
    reader.putFeature "exhaustive-errors", True
    On Error Resume Next
    reader.errorHandler = New error_handler
    reader.parseURL "bog.xml"

End Sub
```

Vi opretter en skema samling (en skema cache) og reader'en får så denne samling tilknyttet. En cache kan loades med en lang række af skemaer som så ligger klar! De er som navnet siger cachet i hukommelsen. Den første parameter til Add er det namespace som skemaet tilhører – her er der intet targetNamespace i skemaet!

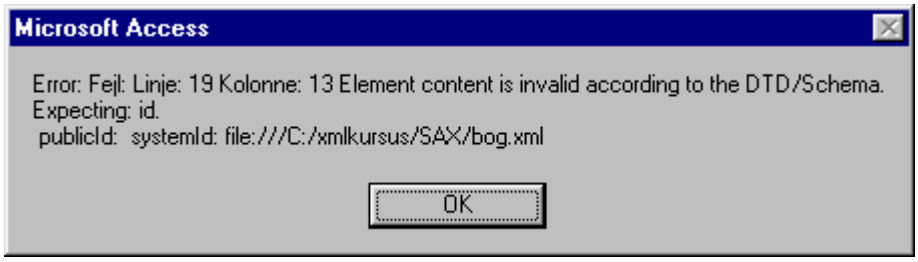
En reader har en property **schemas** som kan sættes til en skema samling. For at få reader'en til at validere er det også nødvendigt at vi sætter en **feature schema-validation** til true! **exhaustive**-errors betyder at parseren fortsætter – også efter den første fejl i forhold til skemaet! Ellers ville en SAX parser normalt stoppe og 'kaste en exception'.

Vi bruger her den samme error_handler som tidligere. Som det ses har vi slet ikke brug for en content handler!

Hvis vi bruger dette XML dokument (med en skema fejl: id mangler):

```
<?xml version="1.0" standalone="yes"?>
<NewDataSet
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="bog.xsd"

>
  <bog_objekt>
    <id>1</id>
    <fornavn>jens</fornavn>
    <efternavn>jensen</efternavn>
    <titel>sommer i vejle</titel>
    <anmeldelse>en rigtig god bog
</anmeldelse>
  </bog_objekt>
  <bog_objekt>
  <!--
    <id>2</id>
    -->
    <fornavn>hanne</fornavn>
    <efternavn>olsen</efternavn>
    <titel>vinter i jylland</titel>
    <anmeldelse>en svag bog
</anmeldelse>
  </bog_objekt>
  <bog_objekt>
    <id>3</id>
    <fornavn>marie-louise</fornavn>
    <efternavn>nielsen</efternavn>
    <titel>hvornr bliver jeg voksen?</titel>
    <anmeldelse>en udviklingsroman
</anmeldelse>
  </bog_objekt>
</NewDataSet>
```

Kan vi validere med vores SAX parser:



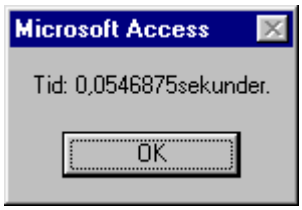
Man kan på denne måde validere mange dokumenter på en gang i en Visual Basic rutine!

Sammenligne DOM og SAX:

Vi kan sammenligne den **tid** det tager at indlæse et XML dokument med SAX og med DOM:

Denne sub rutine måler SAX's effektivitet (eller **performance**):

```
Public Sub load_sax_time()

Set reader = New SAXXMLReader40
starttime = Timer
reader.parseURL "meget_lang.xml"
MsgBox "Tid: " & Timer - starttime & "sekunder."
On Error Resume Next

End Sub
```



Denne procedure måler **performance** i DOM som jo opbygger et træ og en RAM struktur som kan fylde meget mere end selve dokumentet:

```
Public Sub load_dom()

Dim doc As New MSXML2.DOMDocument40
starttime = Timer
doc.Load "meget_lang.xml"
MsgBox "Tid: " & Timer - starttime & "sekunder."
On Error Resume Next

End Sub
```

Forskellen er altså 0,17 sekunder over for 0,05 sekunder for SAX parseren – tre gange så hurtig! En SAX parser har også den fordel at den kan søge et bestemt stykke data og stoppe læsningen når den

har fundet hvad den søgte! Dette kan ikke lade sig gøre med DOM modellen. Eksempel dokumentet her ’meget_lang.xml’ er på 1,86 MegaBytes.

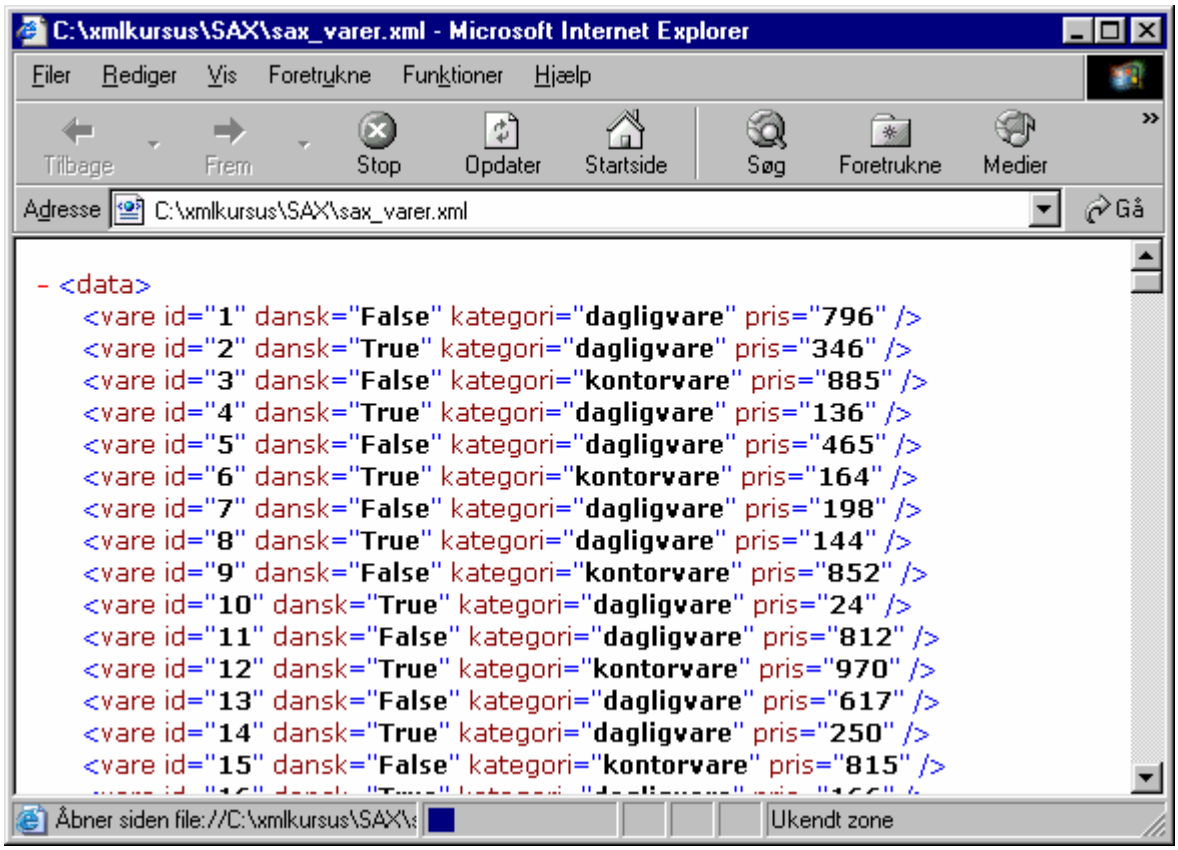
Skrive test XML dokumenter med SAX og Visual Basic:

Vi har allerede set et eksempel på hvordan man kan skrive evt. store filer med SAX. Her er et eksempel på hvordan det kan gøres i Visual Basic:

```
Option Explicit
Public Sub skriv_xml()

    Dim writer As New MXXMLWriter40
    Dim attr As New SAXAttributes40
    Dim doc As New DOMDocument40
    Dim i As Long

    writer.indent = True
    writer.standalone = True
    writer.output = doc
    Dim cont As IVBSAXContentHandler
    Set cont = writer
    cont.startDocument
    cont.startElement "", "", "data", attr
    Dim b As Boolean
    Dim streng As String

    For i = 1 To 200000
    b = False
    streng = "dagligvare"
    If i Mod 3 = 0 Then streng = "kontorvare"

    If i Mod 2 = 0 Then b = True

    attr.addAttribute "", "id", "id", "integer", i
    attr.addAttribute "", "dansk", "dansk", "boolean", b
    attr.addAttribute "", "kategori", "kategori", "", streng

    attr.addAttribute "", "pris", "pris", "", Int(1000 * Rnd())
    cont.startElement "", "", "vare", attr
    attr.Clear
    cont.endElement "", "", "vare"

    Next i

    cont.endElement "", "", "data"
    cont.endDocument
    doc.Save "c:/SAX/sax_varer.xml"
```

End Sub

Denne rutine producerer en XML fil på cirka 14 MegaByte med denne struktur:



Visual Basic funktionen vælger nogle 'tilfældige' værdier for hver vare. Resultatet af skrivningen gemmes i et DOM dokument som så eventuelt kan gemmes på harddisken

XML dokumenter af en vis størrelse – her er der f. eks. 200.000 forskellige vare-poster – er velegnede hvis visse procedurer skal testes. F. eks. kan man teste hvor lang tid det tager at finde visse data!

Søge og finde i vare eksemplet:

At søge og finde er lidt mere besværligt i SAX end i DOM modellen – men hastigheden er som regel meget højere!

Hvis vi f. eks. ønsker at finde alle daglig varer som koster mindre end 10 kroner kan vi gøre følgende:

Private Sub IVBSAXContentHandler_startElement(strNamespaceURI As _

```
String, strLocalName As String, strQName As String, _
ByVal oAttributes As MSXML2.IVBSAXAttributes)

If strLocalName = "vare" Then
  prisdata = CInt(oAttributes.getValueFromQName("pris"))
  If prisdata < 10 Then
     If oAttributes.getValueFromQName("kategori") = "dagligvare" Then

        pris = " pris: " & oAttributes.getValueFromQName("pris") & " "
        kategori = " kategori: " & oAttributes.getValueFromQName("kategori") & " "
        id = "id: " & oAttributes.getValueFromQName("id")
        MsgBox id & pris & kategori

     End If
  End If
End If


End Sub
```

Det skal understreges at hvis XML dokumentet er på 14 megabytes vil der komme rigtigt mange message bokse efter denne metode. I stedet kunne værdierne selvfølgeligt opsamles i en streng som så kan vises eller gemmes.

Det kan ses at søgeprocessen i SAX er fundamentalt anderledes end i DOM. I DOM modellen kan vi søge med et XPATH udtryk. I SAX kan vi kun læse et tegn ad gangen og må derfor søge på en helt anden måde.

Hvis vi prøver at måle tiden for indlæsningen af vores vare dokument giver SAX dette resultat:

og DOM dette resultat:

Forskellen er altså mærkbar – SAX er 5 gange hurtigere!

SAXON SAX parser:

Vi har tidligere omtalt SAX parseren fra **SAXON** skrevet af een mand i princippet Michael **Kay**. Her skal gives et **eksempel** på hvordan man kan skrive kode til SAXON.

Det heldige ved SAX er at der er bred **enighed** om hvordan kode skal skrives hvis man vælger selv at skrive kode til SAX. De **samme** metoder anvendes i script, Microsofts XML, Xerces, Oracle eller her SAXON. Så hvis man først har set metoderne i f. eks. Visual Basic har man set hvordan koden skrives og det hele bliver en del nemmere!

I SAXON – som skal kodes i Java - kan man skrive en Java klasse som parser et XML dokument, kontrollerer det for fejl og udskriver visse data fra dokumentet således:

```
import com.icl.saxon.aelfred.*;

public class Parser {

public static void main(String args[]) {
SAXDriver sax=new SAXDriver();
sax.setContentHandler(new H());
try {
sax.parse(args[0]);
}
catch (Exception e){System.out.print(e.toString());}
}

}
class H extends DefaultHandler {

private int antal=0;
private int elementer=0;

public H() {}

public void characters (char c[], int x, int y) {
antal+=y;
System.out.println("ANTAL TEGN LÆST: "+antal);
System.out.print("TEKST NODEN: ");

for(int i=0;i<y;i++){
System.out.print(c[i]);
}
System.out.println();
}

public void endElement(String x, String y, String z) {
System.out.println("End Element: "+x+" : "+y+" : "+z);
elementer++;
}

public void endDocument() {
System.out.println("ANTAL ELEMENTER I DOKUMENTET: "+elementer);
}
```

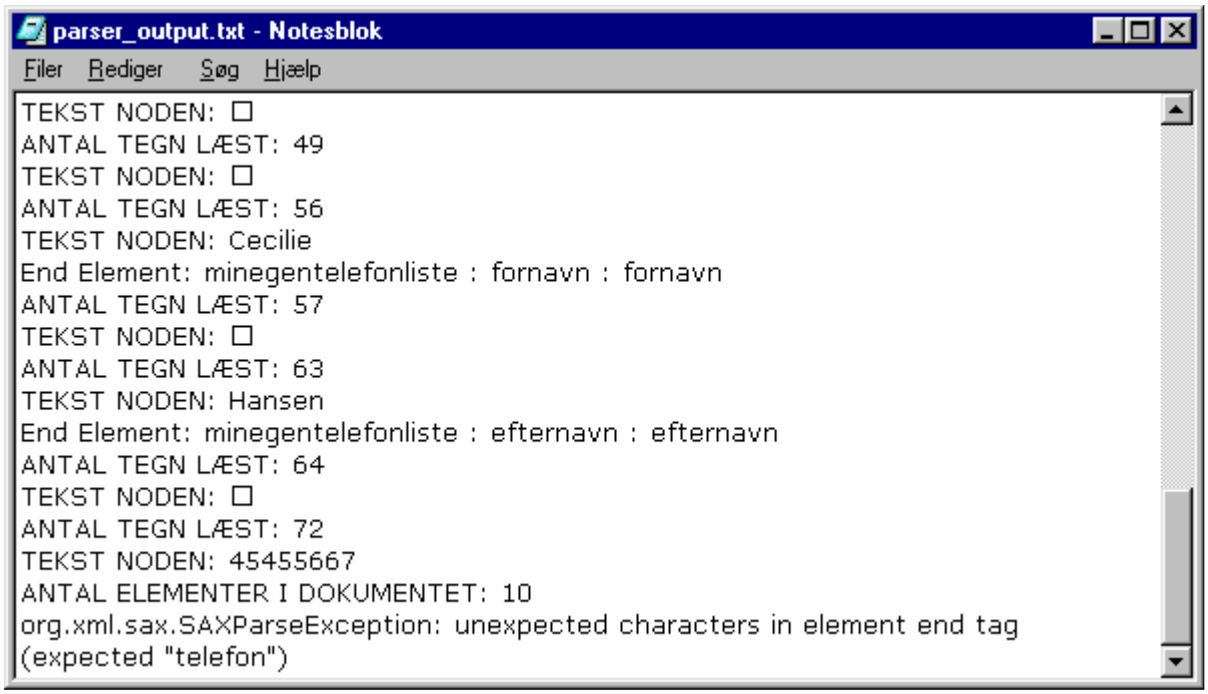
}

Denne kode gør nogenlunde hvad vi også har set tidligere! Programmet tæller hvor mange elementer der er i dokumentet, hvor mange tegn i tekst noderne, udskriver hver tekst node for sig!

Også her er vi nødt til at skrive en event **handler** klasse som definerer hvad vi vil med dokumentet! Vi kan også på denne måde søge efter et bestemt efternavn!

SAX parseren hedder her **SAXDriver** men der findes **mange** forskellige SAX parsere i Java systemet. Her er brugt klasser som følger med SAXON. Denne klasse kan læse dokumenter alle steder fra og kan udskrive om dokumentet ikke er vel formet. Det interessante ved en SAX parser er at den starter processen og **først** standser når den støder på en formel **fejl** - som vist i dette eksempel hvor vi bevidst har indført en fejl i XML dokumentet.

Dette viser tydeligt at vi kan begynde at **bruge** et XML dokument – med SAX i **'real** time' – **også** selv om det indeholder fejl!! Dette kan ikke lade sig gøre med metoderne i **DOM** som først skal indlæse hele dokumentet!:



Programmet udskriver elementets **namespace**, local **name** og **qualified** name. Alle elementerne har fået et namespace - fordi roden har fået et **default** namespace! Denne slags **analyse** er god hvis er i tvivl om hvorvidt et element er inde under et bestemt namespace!

Vi kan se at parseren forventer 'telefon' og har fundet 'elefon'!