SAX – Simple API for XML.

En API (Application Programming Interface) et et sæt af funktioner eller metoder. SAX er et sådant bibliotek af faste 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 skrevet for sproget Java men er 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 – som mere er en **pull** model - det modsatte af SAX modellen. I DOM læses hele filen, der opbygges et DOM træ i RAM (i hukommelsen). I SAX læses filen simpelthen:

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 ingen 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 en meget stor XML fil 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 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.

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 mange 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 – 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 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!

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 alting!

Hvis det 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:

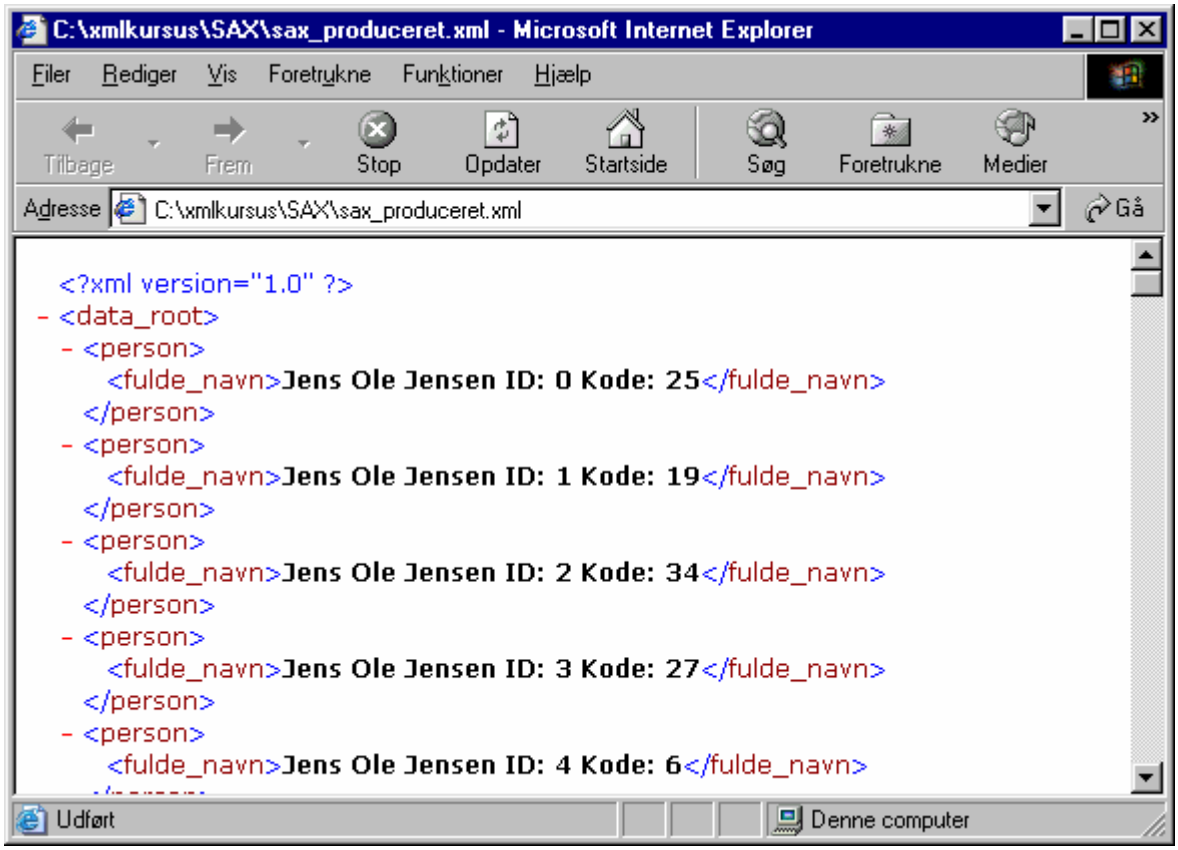
Vi kan producere evt. store test XML filer på denne måde:

```
<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 en ugyldig XML fil skal vi skrive en error_handler, en event handler som aktiveres hvis der findes fejl!

En SAX reader har en property **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. 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
```

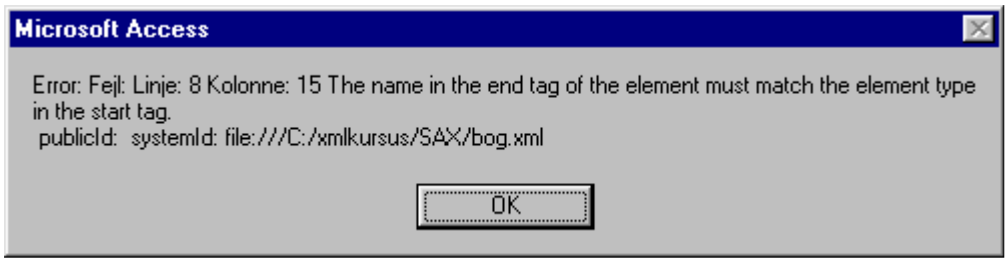
Hele dette system kan virke besværligt men i SAX er der stort set kun brug for en content handler og en error handler. Og disse eksempler kan man så blive ved med at genbruge – så arbejdet er ikke så stort!

En SAX Locator er et objekt som hele tiden registrerer i hvilken linje og kolonne reader'en 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().

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 (cache) og reader'en får så denne samling tilknyttet. En cache kan loades med en lang række af skemaer som så ligger klar! 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'.

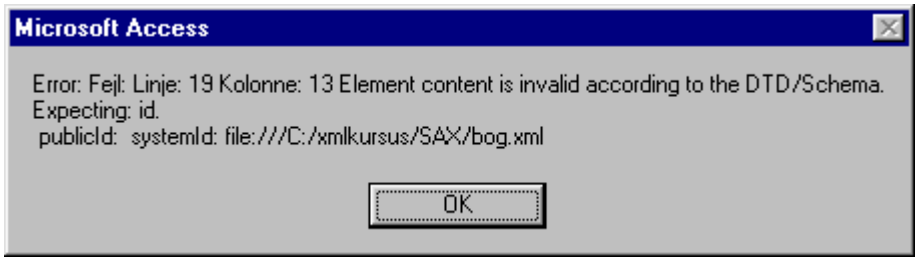
Vi bruger her den samme error_handler som tidligere. Som det ses har vi slet ikke brug for en content handler!

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 (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
```

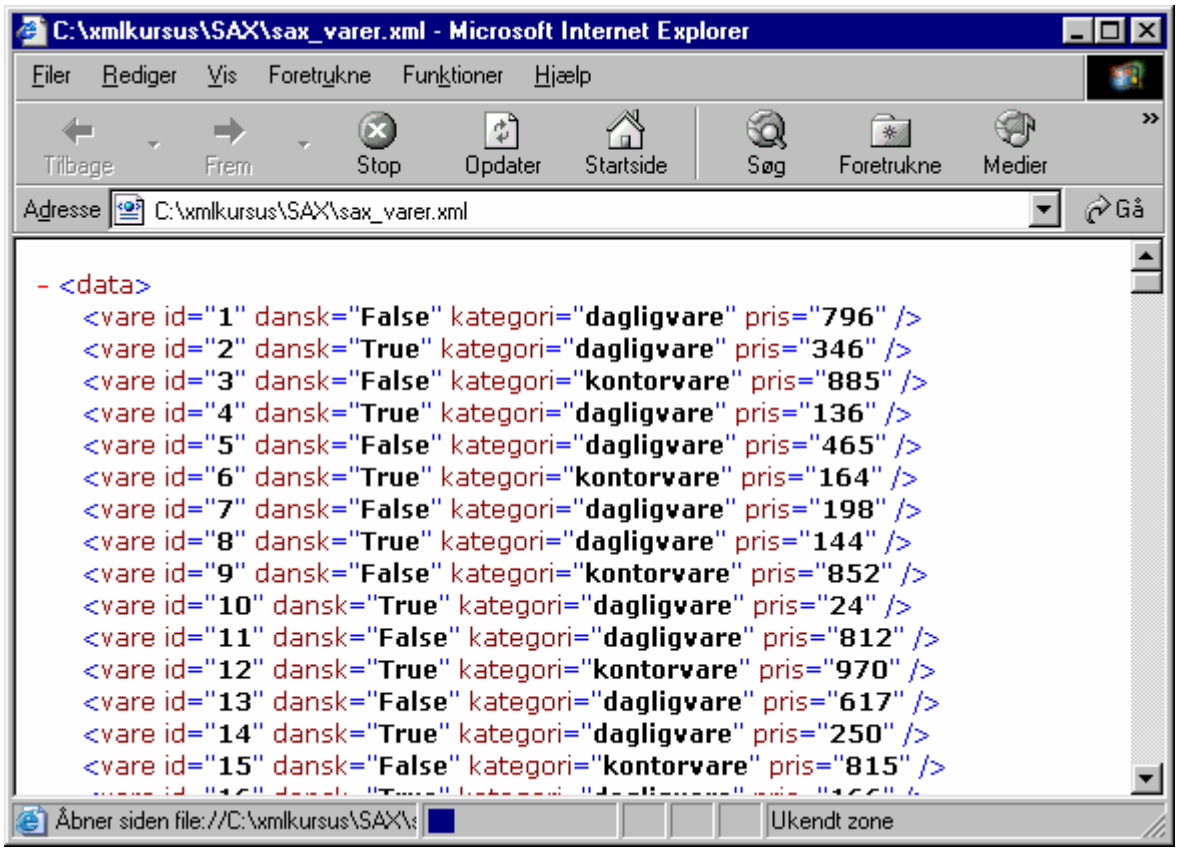
Forskellen er altså 0,17 sekunder over for 0,05 sekunder for SAX parseren – tre gange så hurtig! En SAX parser har også den fordel at den kan søge et bestemt stykke data og stoppe læsningen når den har fundet hvad den søgte! Dette kan slet ikke lade sig gøre med DOM modellen. Eksempel dokumentet her '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 ca 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 kan gemmes på harddiske f. eks. 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 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 kr 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 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 på vores store vare fil giver SAX dette resultat:

og DOM dette resultat:

Forskellen er altså mærkbar – SAX er 5 gange hurtigere!

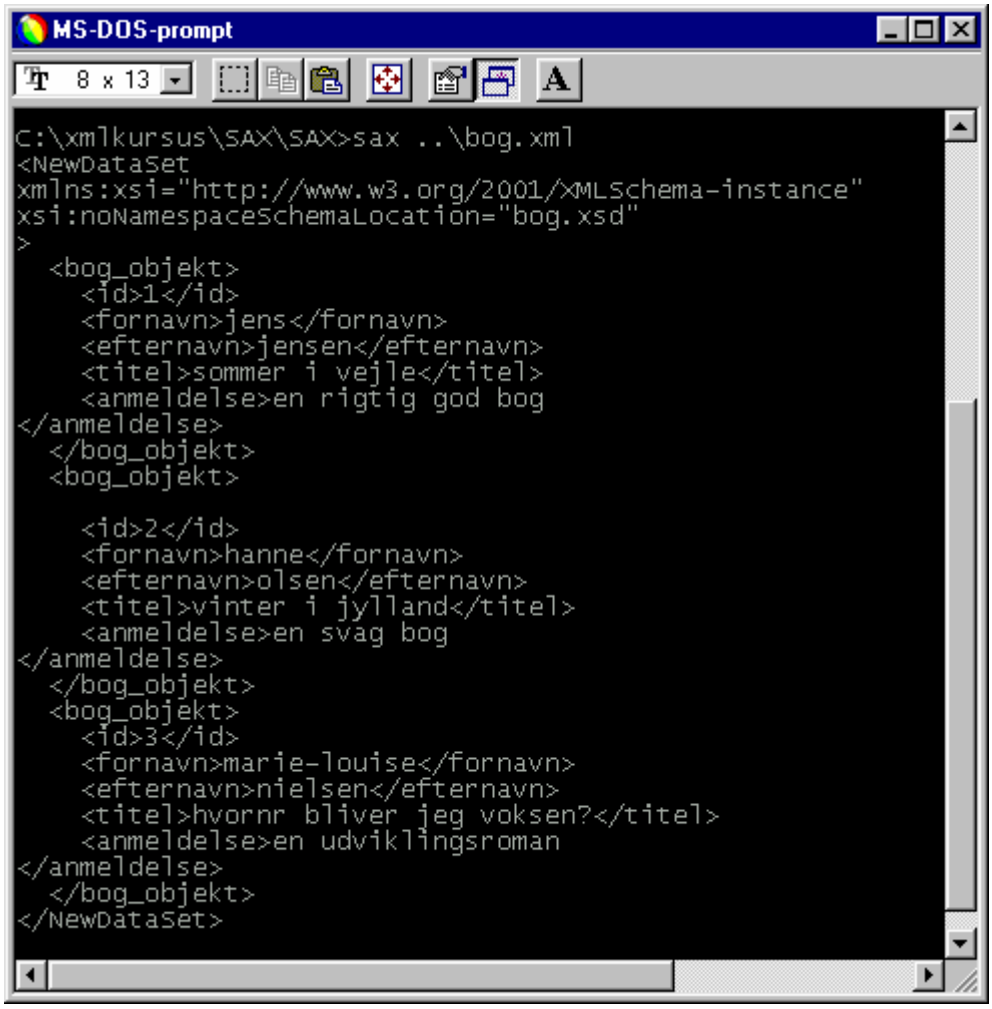
CSharp: Reader:

I Microsoft .NET er SAX ikke direkte implementeret men klassen XMLTextReader er reelt en SAX reader. Den cacher ikke (husker ikke) hvad den har læst, den kan kun gå fremad og den er kun read-only, kan altså ikke ændre XML dokumentet!

Her er nogle simple eksempler på en sådan reader:

```
using System;
using System.Xml;

class MainClass
{
		public static void Main(string[] args)
		{
			XmlTextReader reader=new XmlTextReader(args[0]);
			try{
			reader.MoveToContent();
			Console.WriteLine(reader.ReadOuterXml());
			}
			catch (XmlException e) {
				Console.WriteLine(args[0]+ " blev ikke loadet! \n\n{0}",e.Message);

			}
		}
}
```

Programmet er en simpel SAX reader som viser selve XML dokumentet som gives som kommando linje parameter:

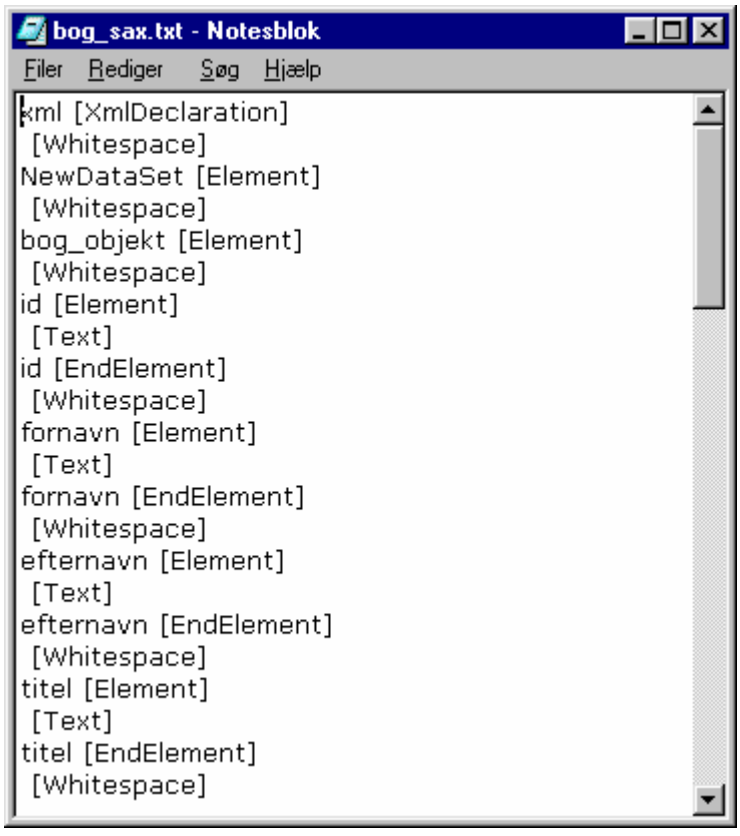


C# programmer kontrollerer om dokumentet er velformet men ikke om det overholder et skema.

Vi kan selvfølgeligt også læse tegn for tegn og analysere (men ikke modificere!) dokumentet. Dette eksempel viser hvordan vi kan læse hvert objekt i dokumentet og udskrive dets navn og type:

```
using System;
using System.Xml;

class MainClass
{
		public static void Main(string[] args)
		{
			XmlTextReader reader=new XmlTextReader(args[0]);
			int antal_noder=1;
			try{
			while(reader.Read()){
			    Console.WriteLine("{0} [{1}]\n",reader.Name, reader.NodeType);
			}
			}
			catch (XmlException e) {
			Console.WriteLine(args[0]+ " blev ikke loadet! \n\n{0}",e.Message);

			}
		}
}
```

Dette program resulterer så i følgende output:



Vi kan her se alle **noderne** i dokumentet! Og vi kan se alle **tekst** noderne – som ikke har noget navn – som er anonyme børn af deres parent – et element eller en attribut. Vi kan også se alle de mellemrum/linjeskift/tabulator – såkaldt whitespace – som parseren støder på! (Og som parseren grundlæggende ignorerer hvis whitespace optræder mellem elementerne).

C#: Et eksempel på en 'SAX' writer:

Hvis vi bruger et ægte programmerings sprog som C# kan vi også skrive XML dokumenter mere effektivt og med større kontrol over resultatet. Hvis vi ønsker at skrive meget store test XMl filer er C# også en oplagt mulighed!

En SAX writer er en writer som hele tiden bevæger sig fremad, aldrig springer i dokumentet. I C# kan vi skrive specialiserede writere som skriver f. eks. til en database eller til et specielt format som SVG eller XHTML.

Fordelen ved at en bruge writer er at den producerer gyldige velformede XML dokumenter. En writer som XmlTextWriter kan dog ikke kontrollere ugyldige tegn som f. eks. et 'Æ' eller at der erklæres to attributter med samme navn i et element! Den kan ikke – som sådan – kontrollere teksten mod et skema. Men den kan selv skrive et internt DTD skema subset!

I C# er klassen **XmlTextWriter** en slags SAX writer. Denne klasse har omkring 100 forskellige metoder, properties og events! Vi kan altså kontrollere vores output ret præcist med en XmlTextWriter!

Et basalt eksempel på en sådan writer kunne være dette:

```
using System;
using System.Xml;
using System.Text;

class MainClass
{
		static XmlTextWriter writer;

		public static void Main(string[] args)
		{
				writer=new XmlTextWriter("produceret.xml",Encoding.UTF8);
				writer.Formatting=Formatting.Indented;
				writer.WriteStartDocument();
				writer.WriteStartElement("personer");
				for(int i=0;i<10;i++) {
						writer.WriteStartElement("person");
						writer.WriteAttributeString("id",""+i);
						writer.WriteElementString("fulde_navn","fulde_navn nr "+i);
						writer.WriteEndElement();
				}
				writer.WriteEndElement();
				writer.WriteEndDocument();
				writer.Close();
```

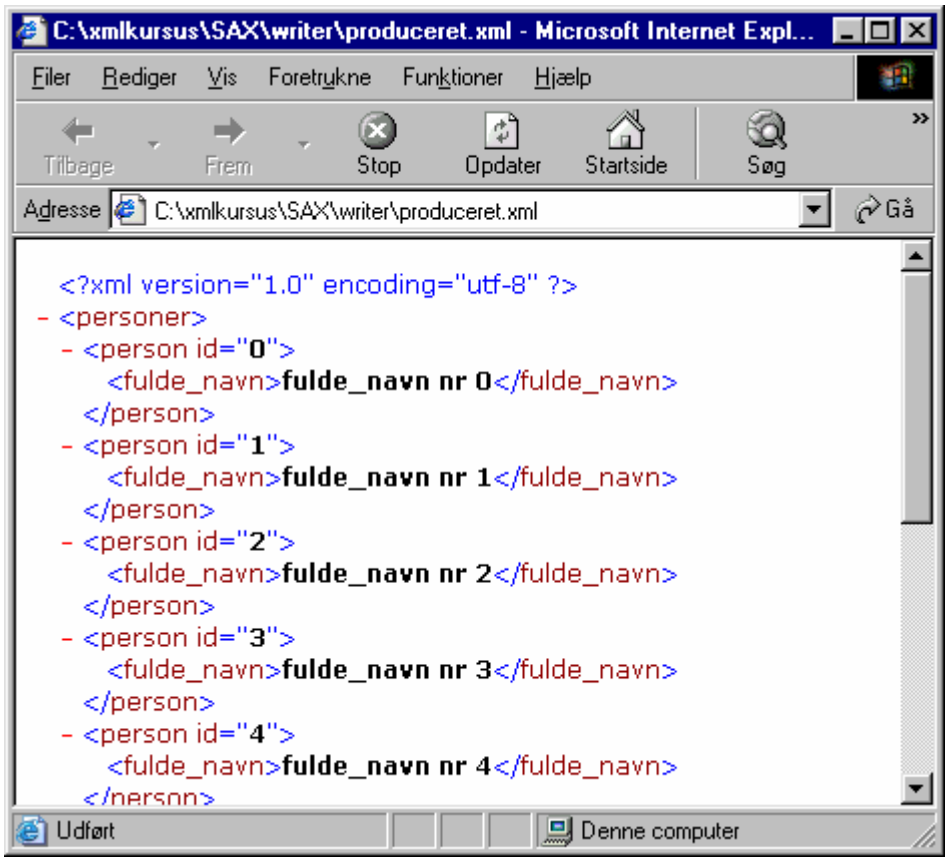
```
                }
}
```

Klassen har metoder som **WriteElementString** som skriver HELE elementet inkl. slut mærke. Tilsvarende med Write **AttributeString**. Og klassen har metoder som **WriteStartElement** som skal bruges til 'complexType' elementer som indeholder attributter og/eller under elementer.

Encoding:

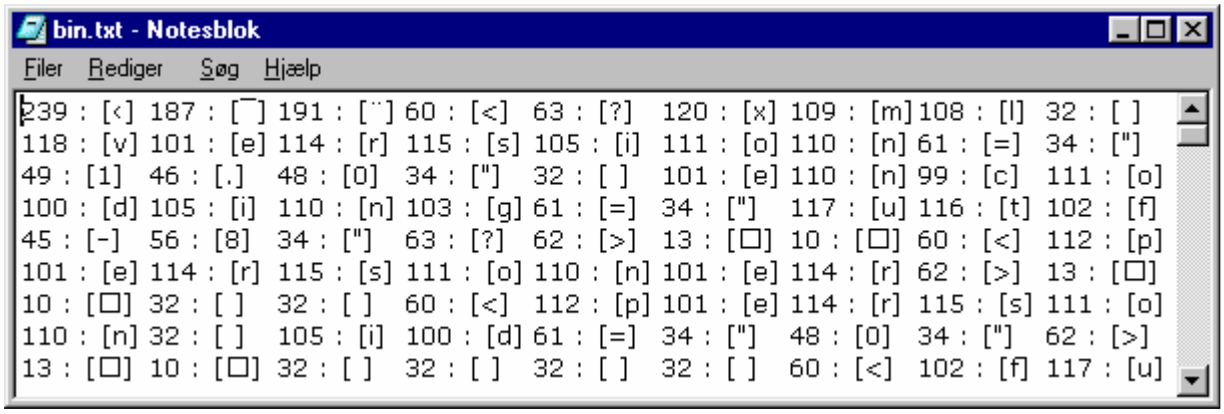
XmlTextWriter kan arbejde med forskellige slags Encoding svarende til attributten i xml erklæringen.

Her produceres dette dokument:



XmlTextWriter producerer altså velformede dokumenter! Som **Encoding** har vi valgt standard encoding nemlig UTF-8 som gemmer tegnene i XML dokumentet i fra 8 bits til 32 bits afhængigt af hvilket tegn vi anvender. Et kinesisk skrifttegn kræver flere bits end et amerikansk 'a' (som altid gemmes i een byte med værdien: 97)!

Hvis vi analyserer den producerede fil binært kan vi se følgende:



Vi har **ikke** anvendt specielle tegn her og hvert tegn fylder derfor kun en byte – 'xml' er f. eks. 3 bytes: 120, 109 og 108!
Filen indledes derimod med 3 tegn vi IKKE har skrevet: 239, 187 og 191 – forud for det første '<' i XML dokumentet!

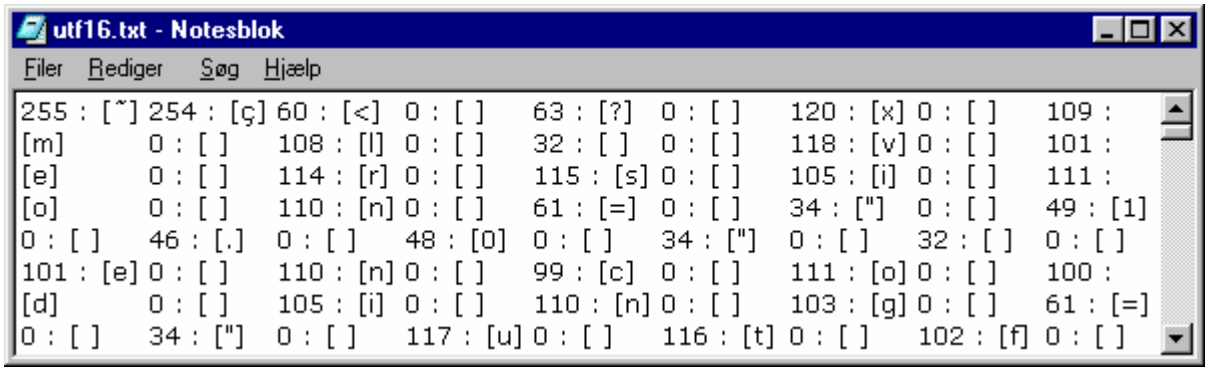
De hexadecimale tilsvarende værdier er: EF (239), BB (187) og BF (191). Disse tegn markerer den kodning som dokumentet anvender og anvendes af parseren når den starter på at læse dokumentet og ikke helt er klar over hvordan dokumentet er kodet!

Hvis vi anvender en anden encoding således:

```
writer=new XmlTextWriter("produceret.xml",Encoding.Unicode);
```

bliver dokumentet gemt i UTF-16 således at hvert tegn altid fylder 16 bits eller to bytes! 'Unicode' bruges oftest om UTF-16 – således også hvis man gemmer et XML dokument i MS Word eller Wordpad! Det er altså OFTE sådan at hvis vi gemmer et XML dokument i UTF-8 eller Unicode kommer det til at fylde dobbelt så meget som hvis vi koder det i UTF-8!

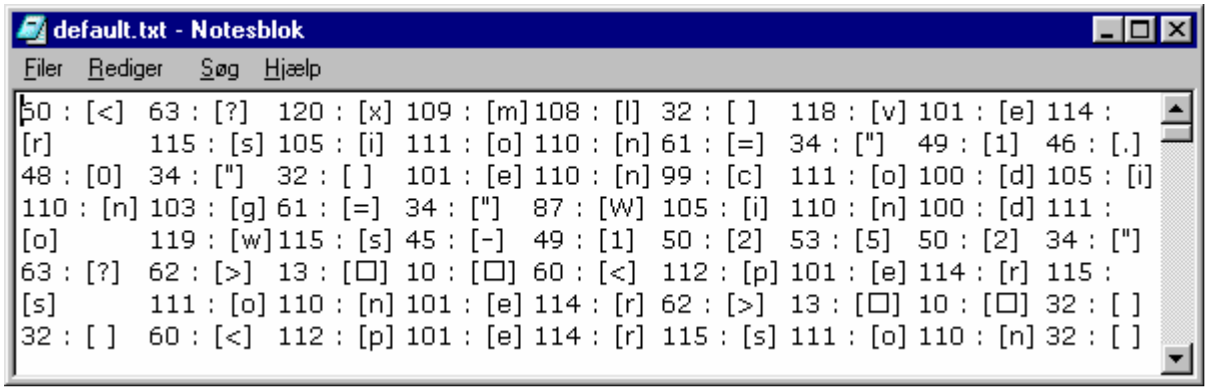
Vi kan se dette i funktioner hvis vi læser det sidste dokument binært:



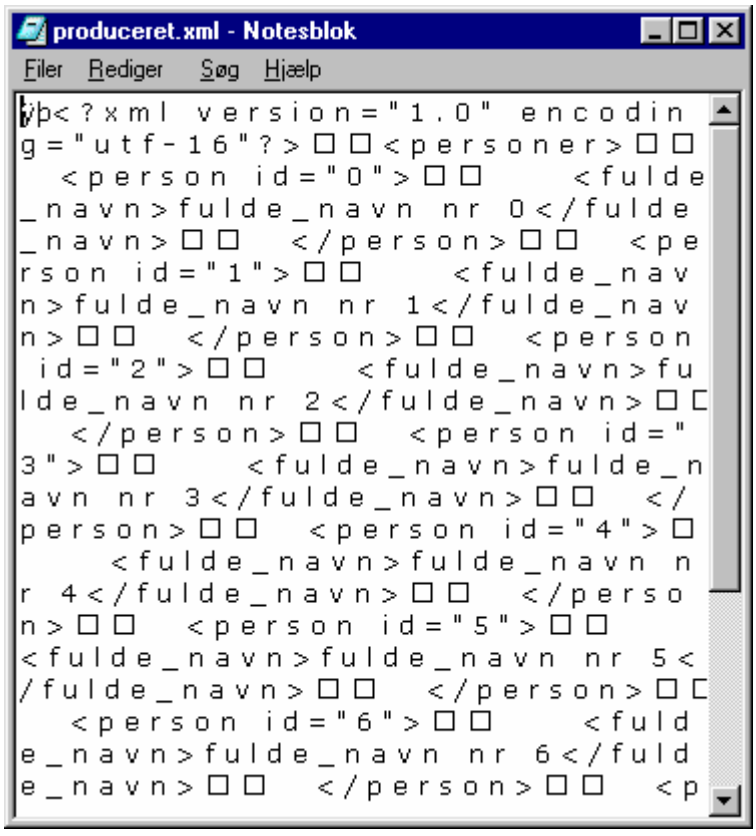
Vi kan se at filen er fyldt op med tomme bytes (0) fordi hvert tegn skal fylde to bytes eller 16 bits. Tegnet 'm' kommer nu til at bestå af først en byte: 0 og så en efterfølgende byte: 109!

Vi kan også se at dette dokument indledes med: FF (255) og FE (254) som viser Unicode kodningen.

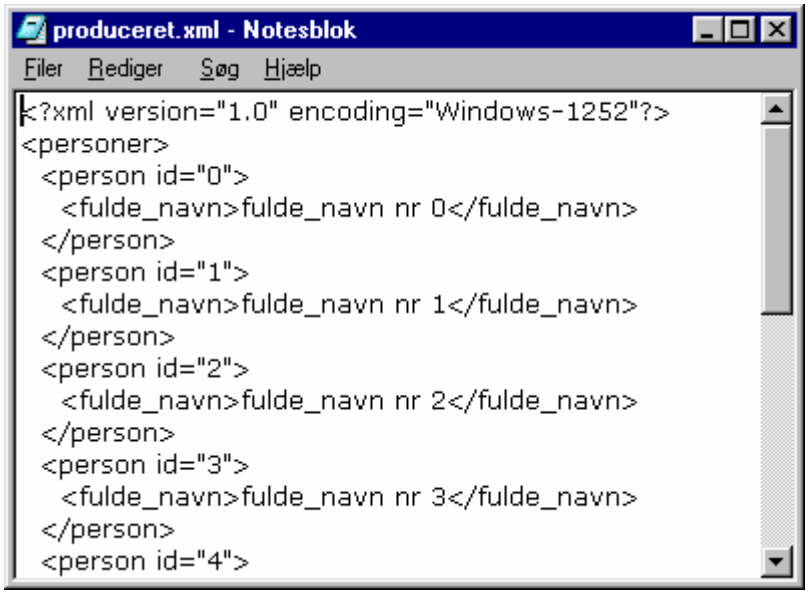
Hvis vi anvender Encoding.Default sættes **ingen** kodnings tegn ind foran i filen!:



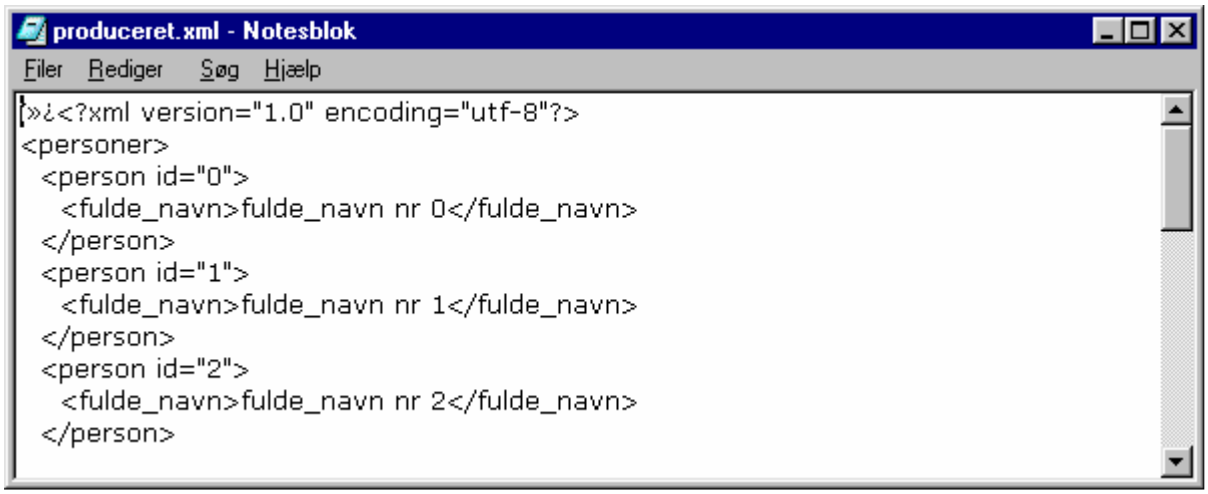
Hvis vi ønsker selve kideteksten vist får vi dette lidt kedelige resultat når vi bruger UTF-16:



Hvis i stedet bruger Encoding.Default får vi denne tekst fil:



Hvis koder med Encoding.UTF8 og ser på den producerede tekst fil får vi dette:



Hvis vi bruger Encoding.ASCII kan XML dokumentet slet IKKE vises i Internet Explorer! Dokumentet indledes ikke af kodnings bytes. (NB browserens tegnsæt kan sættes konkret til et bestemt tegn sæt fx ISO europæisk).

Men selve dokumentet er selvfølgeligt helt ok:

```
<?xml version="1.0" encoding="us-ascii"?>
<personer>
  <person id="0">
    <fulde_navn>fulde_navn nr 0</fulde_navn>
  </person>
  <person id="1">
    <fulde_navn>fulde_navn nr 1</fulde_navn>
  </person>
  <person id="2">
    <fulde_navn>fulde_navn nr 2</fulde_navn>
  </person>
```

Konklusionen er at hvis man vil skrive tekster der f. eks. indeholder danske bogstaver som Æ, Ø eller Å så skal der vælges Encoding.Default! Også Encoding.UTF8 viser de f. eks. danske bogstaver korrekt – men den producerede tekst fil er ikke så anvendelig!

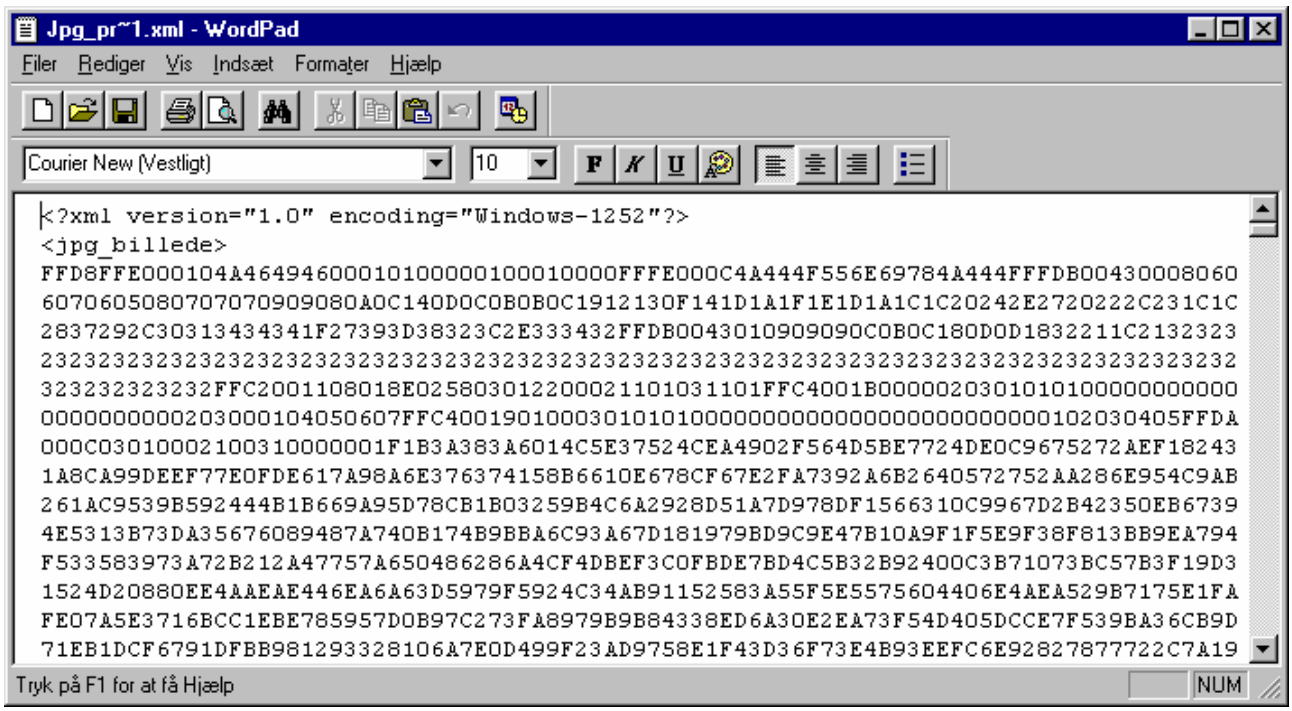
Skrive et JPG billede ind i et XML dokument:

XML dokumenter kan gemme binære data som f. eks. et JPG billede. En server kan på den måde sende et billede sammen med andre data til en klient i et XML dokument! Alle JPG billede – f. eks. – kan gemmes på harddisken som XML filer! (Men de kommer til at fylde noget mere i dette format!).

I følgende kode eksempel oversættes et JPG billede til en række af bytes som så lægges ind i et XML dokument:

```
using System;
using System.Xml;
using System.Text;
using System.IO;

class MainClass
{
		static XmlTextWriter writer;

		public static void Main(string[] args)
		{
				writer=new XmlTextWriter("jpg_produceret.xml",Encoding.Default);
				writer.Formatting=Formatting.Indented;
				writer.WriteStartDocument();
				writer.WriteStartElement("jpg_billede");

				//hent data fra filen på harddisken:
				FileInfo fil=new FileInfo("img.jpg");
				int size=(int)fil.Length;

				//byte tabellen skal have lige saa mange bytes som filen:
				byte[] bytes=new byte[size];
				FileStream stream=new FileStream("img.jpg",FileMode.Open);
				BinaryReader reader=new BinaryReader(stream);

				//byte tabellen fyldes op:
				bytes=reader.ReadBytes(size);
				stream.Close();

				//'SAX' metoden skriver en tabel fra plads 0 og hele tabellen:
				writer.WriteBinHex(bytes,0,size);

				writer.WriteEndElement();
				writer.WriteEndDocument();
				writer.Close();


		}
}
```

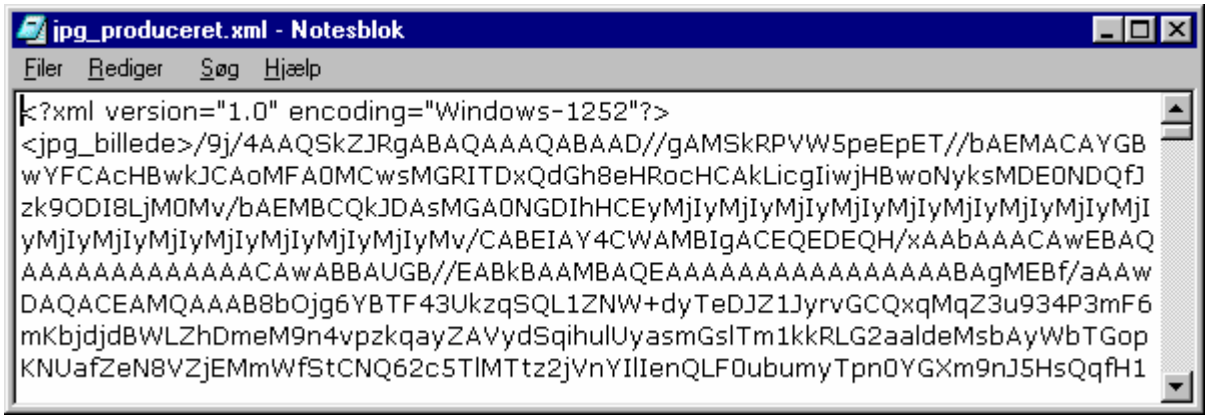
Denne kode producerer en meget lang XML fil:



Hver byte bliver kodet hexadecimalt med et tegn fra 0 til F! I dette tilfælde fyldte jpg filen 41,8 KB mens XML dokumentet kom til at fylde 83,7 KiloBytes! At gemme billedet på denne måde har dog flere fordele som nævnt.

En anden metode er at gemme data i formatet Base64, som koder bytes i et begrænset antal tegn som a til z og A til Z. Vi ændrer blot den ene linje:

```
//writer.WriteBinHex(bytes,0,size);
writer.WriteBase64(bytes,0,size);
```



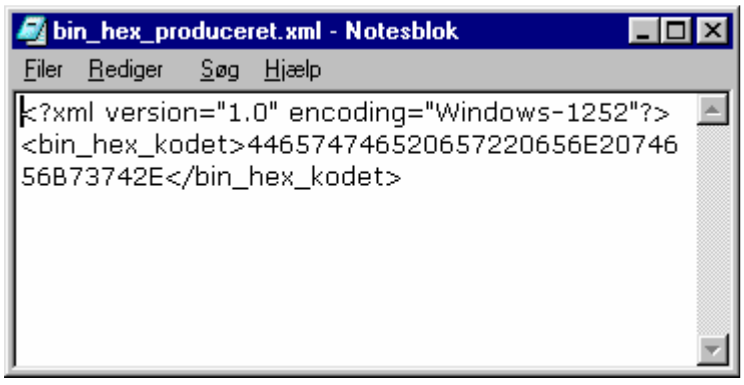
Den nye tekst ud fra Base64 kodningen fylder i alt 55,8 KiloBytes - altså væsentligt mindre end BinHex kodningen!

Kodning af tekster:

Alt kan kodes på denne måde. I XSD findes også en speciel type som markerer at indholdet er kodet. Vi kan som et eksempel kode en tekst på denne måde:

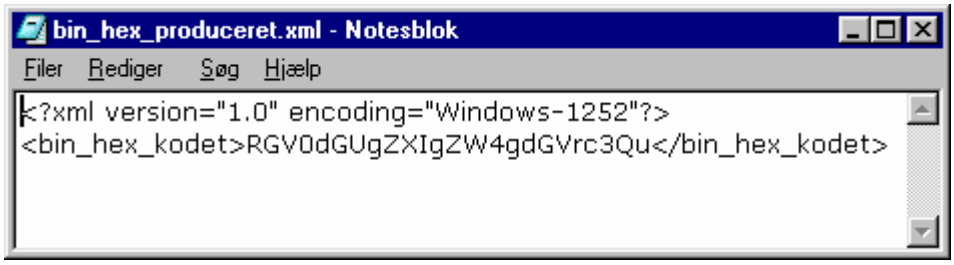
```
using System;
using System.Xml;
using System.Text;
using System.IO;

class MainClass
{
		static XmlTextWriter writer;

		public static void Main(string[] args)
		{
				writer=new XmlTextWriter("bin_hex_produceret.xml",Encoding.Default);
				writer.Formatting=Formatting.Indented;
				writer.WriteStartDocument();
				writer.WriteStartElement("bin_hex_kodet");
				string s="Dette er en tekst.";
				writer.WriteBinHex(Encoding.Default.GetBytes(s),0,s.Length);
				writer.WriteEndElement();

				writer.WriteEndDocument();
				writer.Close();


		}
}
```

Vi får så disse kodede data i XML dokumentet:



Vi kan se teksten er kodet sådan: D (hexadecimalt: 44), e (hexadecimalt: 65), t (hexadecimalt: 74) osv.

En **Base64** kodning af den lille tekst ville omvendt give et helt andet resultat:



Eksempel: En SAX writer som også skriver en DOCTYPE:

En SAXWriter som XmlTextWriter kan selvfølgeligt skrive et internt skema således at der kan anvendes entiteter eller så at dokumentet kan valideres. Følgende er et eksempel herpå:

```
using System;
using System.Xml;
using System.Text;
using System.IO;

class MainClass
{
		static XmlTextWriter writer;

		public static void Main(string[] args)
		{
				string[] fornavn={"Jens
Erik","Lone","Christoffer","Erik","Rosalina","Cecilie","Kai","Marie-Louise"};
	string[] efternavn={"Knudsen","Stadil Jensen","Hoysted","Hansen","Christensen","Nyborg","Jensen","Petersen"};

				writer=new XmlTextWriter("navne_produceret1.xml",Encoding.Default);
				writer.Formatting=Formatting.Indented;
				writer.WriteStartDocument();

				writer.WriteDocType(
				"personer",null,null,
				"<!ELEMENT personer (person)*><!ELEMENT person EMPTY ><!ATTLIST person
fornavn CDATA #REQUIRED efternavn CDATA #REQUIRED telefon CDATA #REQUIRED id CDATA
#REQUIRED>"

				);
				writer.WriteStartElement("personer");

				Random r=new Random();
				string forn="";
				for(int i=0;i<1000;i++){
				writer.WriteStartElement("person");
						writer.WriteAttributeString("id",""+i);

						int f=r.Next(8);
						writer.WriteAttributeString("fornavn",fornavn[f]);
						int f1=r.Next(8);
						writer.WriteAttributeString("efternavn",efternavn[f1]);
						int f2=r.Next(77777777);
						writer.WriteAttributeString("telefon",""+(f2+21212121));
```

```
                writer.WriteEndElement();
                }
                writer.WriteEndElement();


                writer.WriteEndDocument();
                writer.Close();


            }
}
```

Vi producerer her et test dokument med en række forskellige personer med tilfældige for og efternavne og telefonnumre!

Vi kan også skrive et internt DTD subset som vist. Herefter kan filen valideres når den er produceret og evt. modificeret.