Windows programmer. 3
Windows tekst program: 4
Kontrolstrukturen i applikationen: 6
Opgaver: 11
Opgaver: 12
Opgaver: 14
Windows indtastnings formular: Personer 15
Opgaver: 20
Sidespring: Demo af forskellige C# kontroller: 21
ListView: 24
Generelle problemer: Når formen ændrer størrelse og hvad der så sker: 26
Layout og paneler: 30
Opgaver: 32
Eksempel: TreeView kontrollen: 32
Opgaver: 34
Anvendelse af exe.config filer til konfiguration af programmet: 34
Config utility: 36
Opgaver: 36
Windows projekt i SharpDevelop: 37
Delegater, events og event handlers 43
Opgaver: 48
Timer, Threads eller tråde: 48
Threads: 51
ThreadPool – anonyme tråde: 57
Opgaver: 58
Skriv dine egne Windows kontroller: 58
Opgaver: 61
FlatStyle i ButtonBase: 61
MDI Multiple Documents programmer: 63
GDI programmer (Grafik programmer) i C#: 63
Tabeller og databaser – ADO .NET 67
Et Windows program som anvender databasen boghandel: 72
Opgaver: 77
Tabeller kan bruges til meget: 77
Opgaver: 79
Visning/konvertering af XML filer: 79
Opgaver: 83
At koble op til en Access database: 83
DataReader Eksempel: 83
Opgaver: 86
Vis database i DataGrid med en DataAdapter og et DataSet: 87
En DataGrid er nem, men ikke altid velegnet: 89

Opgaver: 90
At anvende en tekst database: 91
At anvende en database på Internettet eller et netværk: 95
ODBC drivere: 95
Ressourcer: 97
Eksempel: ResourceWriter: 97
Skriv dine ressourcefiler i hånden: 100
Gem ressourcer som XML: 101
Opgaver: 104
Lokalisering og ressourcer: 105
En alternativ metode til lokalisering: 110
Opgaver: 113
Hjælp! 113
Opgaver: 121
Interop eksempler: MS Word: 121
Internet og netværks programmer: 123
Microsoft ASP: 123
Statisk HTML: 124
Dynamiske sider: 125
Eksempel ASP og ODBC Database (Access): 127
Klient side scripts: 129
Microsoft ASPX eller ASP .NET: 131
En simpel server i C#: 131
Opgaver: 134
En simpel netværks klient i C#: 134
Unicode: 137
Opgaver: 138
En anden server: 138
Opgaver: 140
IEEXEC: 140
Brug af ActiveX WebBrowser i C# applikation: 141
Sammenfatning af databaser og netværksprogrammer: 141
Remoting i Csharp: 144
Klient: 144
Klientens rpc (DLL) stub: 146
Serverens Service: 148
Serveren: 149
Remoting uden config filer: 152
Code Access Security (CAS): 153
caspol: 160
Profilering af C# programmer: 161
Kryptering: 165
Krypterings program eksempel: 169
Opgaver: 174
Attributter: 174
Ny attribut klasse: DataAttribute: 178
Finde og bruge attribut værdier: 180
UsageAttribute: 185
Opgaver 185

Konvertere fra C# til COM: 186
Dynamisk kald af COM komponent: 189
En COM Person klasse: 190
COM+ eller ComPlus: 192

Windows programmer.

Alle hidtige programmer i dette kursus har været **'konsol** programmer' som er blevet afviklet i en kommando prompt, et DOS vindue.
Konsol programmer er langt fra 'forældede' og anvendes overalt – også selv om grafiske windows programmer har eksisteret siden ca 1990. Konsol programmer har mange **fordele** frem for windows programmer.

Alligevel produceres i dag først og fremmest grafiske windows programmer som præsenterer brugeren for et grafisk 'user interface' (GUI) dvs en flade med vinduer, knapper, tekstbokse osv.
Der er afgørende forskelle på konsol og windows programmer:

1. Windows programmer kører i en principielt evig løkke. Hvis jeg åbner Notesblok stopper programmet ikke af sig selv. Windows programmer startes (normalt) med en: Application.Run() som starter løkken.
2. Windows programmer er **event** styrede dvs de står og venter på at brugeren gør noget og reagerer så på det
3. Windows programmer fungerer ved at der bliver sendt **budskaber** eller **messages** rundt i systemet, når der sker en event: Når jeg klikker på en knap eller Button sender knappen en message til sin 'ejer' eller **parent** fx formen selv. Denne message bliver så databehandlet på en vis måde. Den der skriver programmet bestemmer **hvad** der sker når man klikker på en Button. Alt kan omprogrammeres – fx er det **ikke** givet eller nødvendigt at vinduet lukker fordi jeg klikker på det standard **luk kryds** der som regel er i formens øverste højre hjørne! Jeg **kan** programmere formens event handler til at gøre noget andet end at lukke formen!
4. Windows programmer styres i høj grad af musen eller lignende input mekanismer
5. Windows programmer tegner en grafisk flade af pixels med farver, kan vise billeder, skrift i forskelligt format osv
6. Windows programmer åbner for uanede muligheder og fri fantasi – næsten alt kan lade sig gøre
7. Windows programmer stiller **meget** større krav til programmets design, dets robusthed, dets brugervenlighed osv
8. Hvis Windows programmer skulle kodes 'fra start af' ville de være **urimeligt** besværlige at producere. Mange har prøvet i sin tid at skrive Windows programmer 'from scratch' men hvis projekterne skal være realistiske **skal** Windows programmer bygge på klasser og kode som ligger der på forhånd – på genbrug af kode. Basisklasserne i C# namespace System.Windows.Forms.dll har netop dette formål.

I C# kaldes et window eller et vindue for en **form** og defineres i klassen **Form**. I C# findes to forskellige slags Windows klasser: Windows forms og webforms. I det følgende vil vi **kun** tale om almindelige Windows forms hvis formål er at levere en 'desktop' applikation (og ikke en form til brug på et netværk).

Vi vil i det følgende gradvist opbygge 2 eksempler på Windows applikationer: et tekstbaseret program og et formular (indtastnings) program. Al kode vil her blive skrevet i hånden. Dels for bedre at kunne **forstå** C# programmeringen dels for at skabe mere **effektiv** kode.
Mange C# Windows programmer produceres ved hjælp af et **'builder** program' fordi de fleste finder det meget nemmere med drag-and-drop grafisk at designe et Windows program.

Problemet med denne fremgangsmåde **kan** dels være at programmøren ikke selv forstår den kode 'han' har produceret dels at koden fylder urimeligt meget og derfor ikke er effektiv eller let at overskue. Desuden kan builder programmer **aldrig** skrive den **væsentlige** kode alligevel (et builder program kan typisk skrive triviel primitiv kode) og ofte er det lige så **hurtigt** at skrive koden i hånden!
En tredie eller fjerde fordel er, at hvis man skriver koden i hånden er det muligt at være meget mere **præcis** med den grafiske udformning af et vindue – fx hvis en tekstboks skal anbringes 2 pixels fra vinduets kant – det er ikke særligt nemt at gennemføre i et grafisk drag and drop værktøj!

Vi skal senere vende tilbage til anvendelsen af sådanne builder programmer.

Windows tekst program:

Det simplest mulige Windows program i C# vil nok se sådan ud:

```
//fil:x1.cs

//det simplest mulige C# Windows program

//postcondition: viser et vindue

using System.Windows.Forms;

//vores egen definerede form: XForm

public class XForm : Form{}

//applikation - i samme fil - som anvender klassen XForm:
class app{
                    public static void Main(string[] args){
                    Application.Run(new XForm());
                    }
}
```

Programmet kompileres således:

csc /r:System.Windows.Forms /t:winexe x1.cs

Det er ikke sikkert at referencen r:/System.Windows.Forms er nødvendig men hvis der kommer en fejl meddelelse kan det være referencen er problemet.

/t:winexe eller /target:winexe

sikrer at programmet kompileres som et Windows program – hvilket betyder at hvis exe filen x1 åbnes fx i Windows Stifinder vil der ikke komme et DOS vindue. Ellers er /t:winexe ikke væsentlig. Programmet erklærer først en Xform som arver fra C# klassen Form – uden nogen ændringer. Xform er altså identisk med Form!

Alle de kommende udgaver af XForm vil også blive kaldt for 'XForm'. **Hvis** en klasse skal kunne bruges som basis klasse til nedarving er det selvfølgeligt nødvendigt at den har et **eget** navn! Den skal kunne refereres til med eget navn og i en bestemt DLL fil. Når alle her bliver kaldt XForm er det blot fordi det er det enkleste.

Programmet er – lige som de kommende eksempler – delt i to klasser – selve den nye Windows klasse og en applikations klasse app som instantierer XForm.
En sådan adskillelse gør det nemmere at genbruge klasser.

App klassen anvender Application.Run(): Application er hele programmet som sådan. Metoden Run() er nødvendig for at starte den Windows loop (løkke) som blev nævnt tidligere. Hvis vi blot valgte at vise vores Xform fx med en sætning som enXForm.Show() (det er muligt) ville vinduet forsvinde lige så hurtigt som det kom frem! (Prøv det).

Når x1 køres bliver resultatet følgende:

Vinduet/formen kan minimeres, maximeres, lukkes. Den har en kontrol boks (klik på ikonet), et standard ikon og en standard størrelse. Og en titel linje uden titel. Men i forhold til hvor besværligt Windows programmeringen en gang var – er vi kommet MEGET let til dette resultat!

Klassen **Form** indeholder et utal af forskellige metoder, events og egenskaber. Kun en lille del heraf vil blive gennemgået her. Form klassen arver de fleste af sine egenskaber fra klassen **Control** lige som en lang række af Windows **komponenter** så som tekst boxe, lister, knapper, statuslinjer, check boxe osv osv. P.g.a. denne nedarving er de fleste egenskaber, events og metoder **fælles** for klassen Form og **alle** de forskellige komponenter – heldigvis!

Med lidt øvelse kan man derfor hurtigt få en vis ide om hvad der kan gøres med de forskellige komponenter eller 'kontroller'.
Et eksempel på denne 'polymorfisme': Alle kontroller har en properties som Font og Width og det er altså muligt at skrive fx:

```
enform.Font=new Font("Arial",14);
enform.Width=300;
enbutton.Font=new Font("Arial",14);
enbutton.Width=300;
entekstboks.Font=new Font("Arial",14);
entekstboks.Width=300;
```

NB Det er vigtigt at bruge nogle af de metoder vi gennemgik i forbindelse med C# reflection ellers er det svært at finde rundt i Windows klasserne! Der findes mellem 40 og 50 forskellige Windows kontroller og mellem 4000 og 5000 forskellige metoder og egenskaber for disse kontroller!

Kontrolstrukturen i applikationen:

Hvis vi **analyserer** ovenstående program kan vi se hvordan kontrol strukturen er:

1. Når programmet starter gås **straks** til linjen med Main() i app (Al koden før Main() linjen læses **slet ikke** af operativ systemet i første omgang!!).
2. I Main() gås derefter hen til **parentesen**
3. Der instantieres et nyt anonymt form objekt dvs kontrolstrukturen **kalder** (hopper op i) constructormetoden oppe i klassen og **alle** dens linjer gennemløbes
4. Derefter – nu har vi det som står inde i parentesen – gås til **Run** og derefter til **Application**!
5. Populært sagt læses Main() linjen **bagfra**!
6. NB i Main() kunne opstartes **mange** forms efter hinanden. De udgør så tilsammen det som kaldes **Application**. En Application kan indeholde mange forms (eller andet).
7. Der er derfor **forskel** på at lukke en form (form.Close()) og at stoppe applikationen (Application.Exit())!

I det følgende eksempel er der indsat en tekst boks i vinduet og der er etableret lidt mere kontrol over vores 'default' vindue:

```
//fil:x2.cs

//mere kontrol over formen:

//postcondition: viser et vindue med en rich tekstbox

using System.Windows.Forms;
using System.Drawing;

//vores egen definerede form: XForm

public class XForm : Form {
```

```
            //klassens instans variable/egenskaber/komponenter:
            RichTextBox tekstbox;

            //constructor til XForm:
            public XForm(){

            Size=new Size(600,400);
            Text="C# Tekst program.";
            CenterToScreen();
            tekstbox=new RichTextBox();
            tekstbox.Size=new Size(Width-10,Height-50);
            Controls.Add(tekstbox);

            }




}

//applikation - i samme fil - som anvender klassen XForm:
class app{
            public static void Main(string[] args){
            Application.Run(new XForm());
            }
}
```

Kommentar:
En række metoder i Form kræver klasser fra System.Drawing.dll – derfor er der tilføjet en using sætning.
Form har en liste **Controls** som rummer alle de komponenter som sættes på formen. For at en komponent vises skal den tilføjes samlingen **Controls –** det sker med:
Controls.Add();
Som det vil fremgå gælder denne procedure alle de komponenter som ønskes sat på formen.
Alle komponenter har egenskaber som Size, Location, Left, Top, Width og Height.
Formens koordinat system starter i øverste venstre hjørne med punktet 0,0. Første tal angiver pixel rækken og 2. tal angiver pixel kolonnen.
Hvis en komponents Left sættes til 30 og Top til 100 betyder det: gå 30 pixels hen (x aksen) og derefter 100 pixels ned (y aksen).
I stedet kan angives en Location som fx:

tekstbox.Location=new Point(300,100);

Size og Point findes i **Drawing**.dll. Alle pukter på formen angives efter skemaet: new Point(hen,ned). Hvis en komponent ikke defineres med en størrelse sættes den ind med en default standard størrelse – TextBox indsættes som en lille inputbox. I C# er der ikke principielt forskel på en lille tekstboks og et stort tekst vindue (modsat andre sprog).
I dette tilfælde indsættes en RichTextBox (som kan rumme formatteret tekst) som blot fylder hele det tilgængelige område.

NB Med hensun til formens **size**: Det er vigtigt at **skærmopløsningen** er ret forskellig på forskellige maskiner! Hvis du har sat formens bredde til 1100 **pixels** – vil der være en del

maskiner der ikke vil kunne se hele formen!! Bredde og højde bør derfor – som et primitivt **udgangspunkt** - være max 640/480 eller 800/600! Vi skal senere se hvordan dette problem kan løses mere fikst!

Desuden sættes en tekst i formens titelinje (**Text –** alle komponenter har en Text fx TextBox, Button, ListBox, Label) og formen sættes til en start størrelse.
Som det ses er al denne 'initialisering' anbragt i en **constructor**.
Initialiseringen kunne være anbragt i en selvstændig metode Initialize() som så blev kaldt af constructoren. Builder værktøjer gør som regel dette.
Bemærk at hvis formen maximeres – ændrer tekst boksens størrelse sig **ikke**!

Prøv at eksperimentere med formen som tekst behandling – de fleste almindelige mekanismer fra tekst behandlingen virker allerede! Fx Control-X, Control-V osv. Du kan kopiere fra anden tekst og indsætte den med Control+V!

Et tekst program kan dårligt klare sig uden en **menu** – og det kommende eksempel viser hvordan en **'minimums** menu' ret nemt kan etableres. Bemærk at det i første omgang er nemmest at lave hele menuen og først siden tilføre menuen funktionalitet. Menu punkterne er derfor bevidst IKKE implementeret i dette eksempel:

```
//fil:x3.cs

//en hovedmenu er sat på formen:

//postcondition: viser et vindue med en tekstbox og en menu

using System.Windows.Forms;
using System.Drawing;
using System;

//vores egen definerede form: XForm

public class XForm : Form {

        //klassens instans variable/egenskaber/komponenter:
        //alle private - eksempel på OOP indkapsling:
        private RichTextBox tekstbox;
```

```
        private MainMenu menu;
        private MenuItem filer;
        private MenuItem ny;
        private MenuItem aaben;
        private MenuItem gemsom;
        private MenuItem udskriv;
        private MenuItem exit;

        //constructor til XForm:
        public XForm(){

        Size=new Size(600,400);
        Text="C# Tekst program.";
        CenterToScreen();
        tekstbox=new RichTextBox();
        tekstbox.Size=new Size(Width-10,Height-50);
        Controls.Add(tekstbox);

        //constructor kalder hjælper metoder opret_menu():
        opret_menu();
        }

        //opret_menu() er anbragt i sin egen metode
        //for at gøre koden modulær/overskuelig:

        private void opret_menu(){
        menu=new MainMenu();
        filer=new MenuItem("&Filer");
        menu.MenuItems.Add(filer);

        //opret de enkelte menu punkter og tilføj dem til menuen 'Filer':

        ny=new MenuItem("&Ny",new EventHandler(klik_menu),Shortcut.CtrlN);
        filer.MenuItems.Add(ny);
        aaben=new MenuItem("Åbn",new EventHandler(klik_menu),Shortcut.CtrlO);
        filer.MenuItems.Add(aaben);
        gemsom=new MenuItem("Gem som",new EventHandler(klik_menu),Shortcut.CtrlS);
        filer.MenuItems.Add(gemsom);
        udskriv=new MenuItem("Udskriv",new EventHandler(klik_menu),Shortcut.CtrlP);
        filer.MenuItems.Add(udskriv);
        exit=new MenuItem("E&xit",new EventHandler(klik_menu),Shortcut.CtrlX);
        filer.MenuItems.Add(exit);

        Menu=menu;
        }

        //en event handler der lige nu ikke gør noget som helst:
        private void klik_menu(object o,EventArgs a){}



}

//applikation - i samme fil - som anvender klassen XForm:
class app{
        public static void Main(string[] args){
```

```
        Application.Run(new XForm());
        }
}
```

Kommentar:
Som det kan ses opstår megen kode men det meste er trivielle gentagelser:

```
ny=new MenuItem("&Ny",new EventHandler(klik_menu),Shortcut.CtrlN);
filer.MenuItems.Add(ny);
```

gentages for alle menupunkter. Hvert menupunkt er et nyt objekt, som kan oprettes med 3 paramtre: 1)teksten i menupunktet (& tegnet giver en shortcut med fx Alt+F), 2)en eventhandler og en 3)genvejs tast.
Som genvejstaster kan vælges alle kombinationer med Control og Alt og funktionstasterne. Det er vigtigt at være konservativ og vælge genveje som i forvejen er kendte som det er gjort i koden ovenfor. Fx skal F1 helst være Hjælp og Control+X helst være exit (programmet lukker).
Der er foreløbigt skrevet den samme **eventhandler** til alle menupunkt-objekter nemlig:

```
private void klik_menu(object o,EventArgs a){}
```

Metoden er så efterhånden at implementere metoden for de enkelte menu punkter.
At klikke på et menu punkt udløser en **event**. En event gør det muligt for programmet at gøre noget når en bestemt event forekommer. En **eventhandler** er en **metode** som definerer hvad der skal ske. Et mere generelt begreb for en event handler er i C# en **delegat**. En delegat delegerer arbejdet videre til en konkret metode. Delegater bruges overalt i Windows programmeringen i C#.

Alle event handlere skal basalt set have de samme parametre som i eksemplet ovenfor: Nogle har en subklasse af EventArgs som 2.argument - fx en muse event handler der tager disse parametre: klik_mus(object o, **MouseEventArgs** a). Alle eventhandleres 1. argument er **object** o – dvs det objekt som er 'afsender' af event'en (hvordan object kan udnyttes vil siden blive vist). Det første argument skrives derfor ofte i C# som 'object **sender' – hvad** jeg kalder parameteren er helt ligegyldigt! Begge argumenter gemmer **information** om den skete event som kan udnyttes af programmet. I eksemplet med en muse event gemmer MouseEventArgs fx x og y koordinaterne for **hvor** på formen musen er klikket.

Resultat af at køre programmet x3:

Opgaver:

1. en 'use case' bruges som en beskrivelse af en bestemt handling som en bruger kan udføre med et program: skriv en liste over mulige use cases i et tekstbehandlings program – hvilke funktioner vil en bruger ønske implementeret
2. hvilke events kan forekomme i et program som i eksemplerne her - (et tekst baseret program)?
3. skriv kode til at oprette flere menuer og flere menupunkter
4. opret en samlet menu der minder om den man finder i professionelle programmer af denne type
5. implementer disse menupunkter ved at der vises en MessageBox når brugeren klikker på menupunktet. I disse bokse kan stå korte beskeder om hvad brugeren har gjort eller der kan stå at punktet snart bliver implementeret eller lignende! Se tidligere eksempler på hvordan man instantierer MessageBoxe - hvis du er i tvivl!
6. Tildel menupunkterne 'konservative' shortcuts/genvejstaster
7. Giv formen en anden størrelse, find frem til en ideel størrelse
8. Korriger tekstboksen tilsvarende
9. Prøv at skifte ikon på formen med formlen:
   Icon=new Icon("mit_ikon.ico");

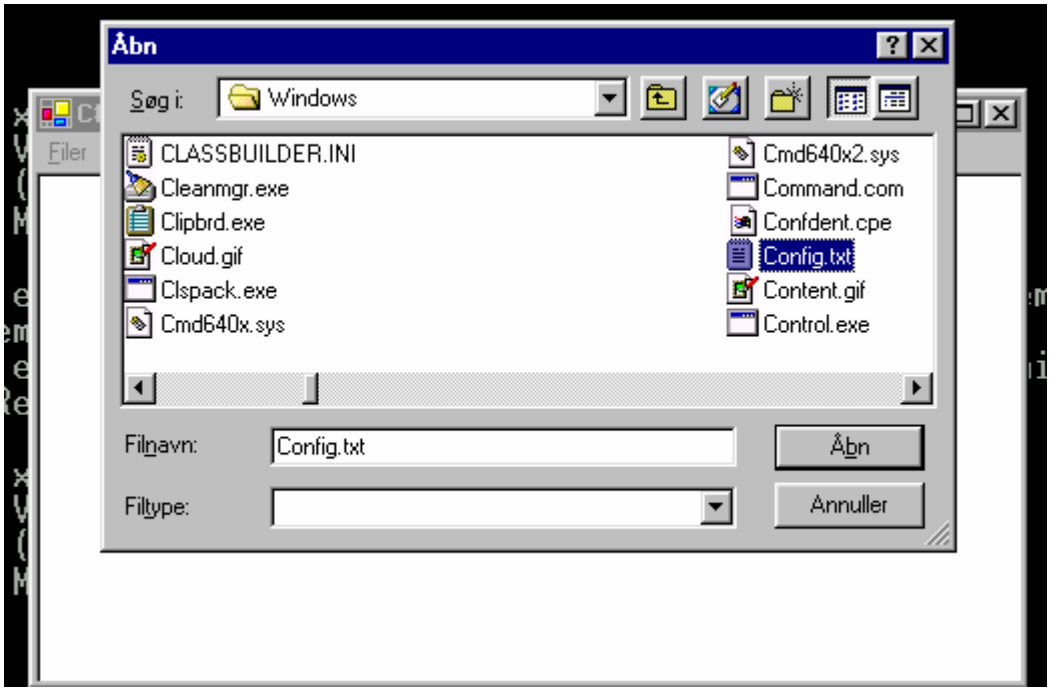
Vi vil nu implementere to af menupunkternes funktionalitet, så de laver noget fornuftigt. Det enkleste er at implementere exit menuen men vi er også interesseret i menu punktet Åbn:

```
//event handlere til menuen Filer:
private void klik_menu(object o,EventArgs a){}//tom eventhandler!

private void klik_menu_aaben(object o,EventArgs a){
OpenFileDialog dialog=new OpenFileDialog();

//Hvis bruger klikker OK efter at have valgt en fil:
if((dialog.ShowDialog())==DialogResult.OK){
Stream stream=dialog.OpenFile();
StreamReader reader=new StreamReader(stream);
string fil=reader.ReadToEnd();
tekstbox.Text=fil;
}
```

```
}

private void klik_menu_exit(object o,EventArgs a){Application.Exit();}
```

Der er skrevet to nye eventhandlere: klik_menu_aaben() og klik_menu_exit(). Application.Exit() betyder at hele applikationen lukker. I stedet kunne bruges: Close(), som ville lukke denne form. Resultatet er ofte det samme.

Den anden eventhandler instantierer en af basisklasserne i C# - OpenFileDialog. (Der findes helt tilsvarende en SaveFileDialog, ColorDialog, FontDialog og en PrintPreviewDialog). Hvis brugeren klikker Annuller (Cancel) sker der intet. Hvis der klikkes Åbn/OK **efter** at en fil er valgt vises filen i formen. Metoderne til at læse en fil er omtalt tidligere i afsnittet om filer.



En kørsel med x4 kan give dette resultat.

Opgaver:

1. Skriv en implementering dvs en eventhandler til menupunktet Ny
2. Skriv en eventhandler til menupunktet Gem som – brug de ting der stod i afsnittet om filer. Du kan bruge basisklassen SaveFileDialog, men du kan også gøre det på anden måde.
3. Skriv en eventhandler til punktet Udskriv
4. Skriv eventhandlere til andre menuer og menupunkter som du synes skal med i programmet
5. Lav en menu Hjælp og et menupunkt 'Om Programmet' og vis en MessageBox der beskriver programmet (en såkaldt About box). Undersøg hvad der plejer at stå i sådanne bokse!
6. Lav et menupunkt under Hjælp der viser en **ny** form (med forskellig slags hjælp) hvis der klikkes. Opret en **ny** klasse der arver fra Form, instantier den når der klikkes i menuen og vis den således:

   Nyform.Show();

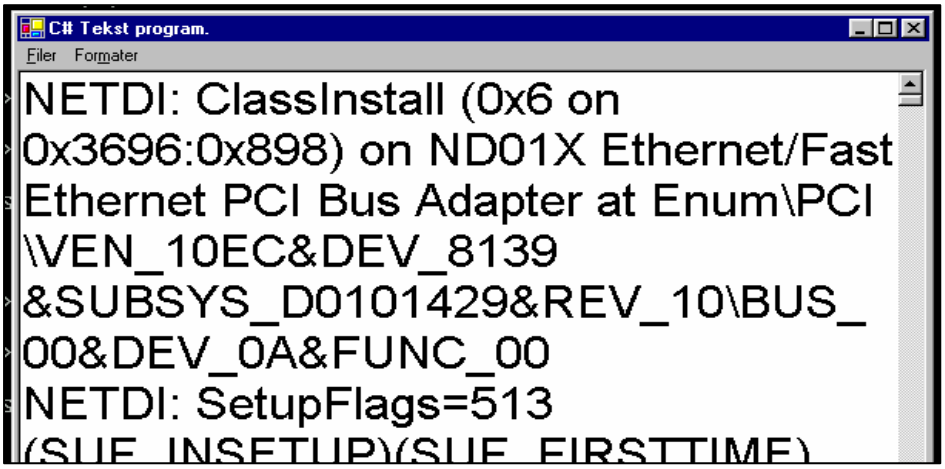
Eller:
Nyform.ShowDialog();

Hvad er forskellen på resultatet af de to metoder!? Hvordan opfører de to vinduer sig?

I dette sidste eksempel er tilføjet en ny menu Formater som kan bruges til at ændre skriftstørrelsen i tekst boksen.
Læg mærke til at tekstboksen har en automatisk linje brydning og automatisk visning af rullepaneler (scrolling).
Som det ses er det rimeligt nemt at arbejde videre med menuer når først grunden er lagt. Det meste består blot i kopier og sæt ind og så ændre nogle få værdier!

```
//opret_menu() er anbragt i sin egen metode
//for at gøre koden modulær/overskuelig:

private void opret_menu(){
menu=new MainMenu();
filer=new MenuItem("&Filer");
menu.MenuItems.Add(filer);
formater=new MenuItem("For&mater");
menu.MenuItems.Add(formater);


//opret de enkelte menu punkter og tilføj dem til menuen 'Filer':

ny=new MenuItem("&Ny",new EventHandler(klik_menu),Shortcut.CtrlN);
filer.MenuItems.Add(ny);
aaben=new MenuItem("Åbn",new EventHandler(klik_menu_aaben),Shortcut.CtrlO);
filer.MenuItems.Add(aaben);
gemsom=new MenuItem("Gem som",new EventHandler(klik_menu),Shortcut.CtrlS);
filer.MenuItems.Add(gemsom);
udskriv=new MenuItem("Udskriv",new EventHandler(klik_menu),Shortcut.CtrlP);
filer.MenuItems.Add(udskriv);
exit=new MenuItem("E&xit",new EventHandler(klik_menu_exit),Shortcut.CtrlX);
filer.MenuItems.Add(exit);

stor=new MenuItem("Stor skrift",new EventHandler(klik_stor));
formater.MenuItems.Add(stor);
mellem=new MenuItem("Mellem skrift",new EventHandler(klik_mellem));
formater.MenuItems.Add(mellem);
lille=new MenuItem("Lille skrift",new EventHandler(klik_lille));
formater.MenuItems.Add(lille);


Menu=menu;
}

//event handlere til menuen Filer:
private void klik_menu(object o,EventArgs a){}//tom eventhandler!

private void klik_menu_aaben(object o,EventArgs a){
OpenFileDialog dialog=new OpenFileDialog();

//Hvis bruger klikker OK efter at have valgt en fil:
```

```
            if((dialog.ShowDialog())==DialogResult.OK){
            Stream stream=dialog.OpenFile();
            StreamReader reader=new StreamReader(stream);
            string fil=reader.ReadToEnd();
            tekstbox.Text=fil;
            }
            }

            private void klik_menu_exit(object o,EventArgs a){Application.Exit();}

            private void klik_stor(object o,EventArgs a){tekstbox.Font=new Font("Arial",24);}
            private void klik_mellem(object o,EventArgs a){tekstbox.Font=new Font("Arial",12);}
            private void klik_lille(object o,EventArgs a){tekstbox.Font=new Font("Arial",8);}
}
```



En kørsel med x5:

Opgaver:

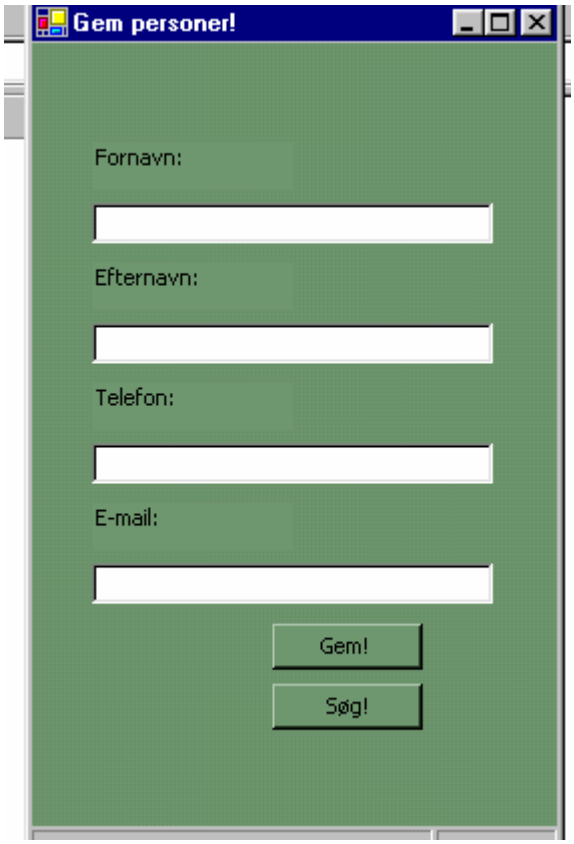
1. Opret flere menupunkter under Formater: fx valg af en skriftfarve eller skrift type **uden** at anvende FontDialog eller ColorDialog.
2. Prøv at skrive menupunkter der **bruger** en **FontDialog** og en **ColorDialog**. Når brugeren lukker disse bokse findes brugerens valg i fontdialog.**Font** og farvedialog.**Color**. Brug ellers den kode som vises i eksemplerne.
3. Skriv kode til at titelinjen viser filnavnet (som det ofte ses).
4. Du indsætter en separator streg mellem to menupunkter ved at indsætte et nyt MenuItem med strengen ”-”. Prøv at gøre dine menuer mere overskuelige ved at indsætte sådanne adskille streger.
5. Skriv kode til en helt ny Windows applikation som indsætter et baggrundsbillede på hele formens baggrund:
   Filens navn fås igennem et kommandolinje argument (jvf string[] args)
   Billedet indsættes med koden:

   BackgroundImage=Image.FromFile(…her står filens navn…);

   Skriv kode så at filens navn sættes i titelinjen.

Windows indtastnings formular: Personer

En anden almindelig type af Windows programmer er **formularer** dvs. en form som kan **vise** data og hvor der kan **indtastes** og **gemmes** nye data. Formen ser sådan ud når programmet kører:



Den mest interessante kode knytter sig til de to Buttons. Når vi tilføjer vores gemknap til formen med metoden Controls.Add() knytter vi samtidigt en event handler til knappen efter denne formel:

knap.Click+= new EventHandler(metodenavn);

Det som sker her er følgende: Vi definerer at når der klikkes på denne knap kaldes metoden 'metodenavn'. Næsten alle kontroller Button, PictureBox, ComboBox, Label osv en en indbygget event der hedder Click. Det gælder også selve vinduet – som altså kan reagere hvis der klikkes på det. Formlen += betyder at **mange** forskellige event handlers kan tilføjes til den samme kontrol.

Metoden i dette tilfælde ser sådan ud:

```
private void gemBtn_clicked(object o,EventArgs a){

   panel1.Text="Personen gemmes i databasen: personer.txt.";//skriv tekst i statuslinjen som information
   FileInfo fil=new FileInfo("personer.txt");

   //nye personer appendes til filen:

   StreamWriter writer=fil.AppendText();
   writer.WriteLine(fornavn.Text);
   writer.WriteLine(efternavn.Text);
```

```
   writer.WriteLine(telefon.Text);
   writer.WriteLine(email.Text);
   writer.Close();

   //Nulstil tekst bokse:
   fornavn.Text="";
   efternavn.Text="";
   telefon.Text="";
   email.Text="";

}
```

Alle event handlers har den samme **struktur**: De har 2 parametre et **objekt** som er den kontrol eller andet som er afsender af **message** (budskabet) og et **EventArgs** som indeholder forskellige data om event'en.
Her gør vi det vi åbner en **fil** personer.txt som append (dvs hver gang vi gemmer, gemmer vi i **forlængelse** af det som allerede er gemt i filen) og skriver indholdet af tekstboksene ind i filen linje for linje. Dette er **ikke** nogen optimal måde at gemme data på (en database havde været væsentligt bedre og mere 'sikker') – men her er det vist som et enkelt eksempel på hvordan man kan gemme data. (Vi skal senere se hvordan man kan gemme data i en database).

Derefter resettes (nul stilles) tekstboksene for at gøre det hele mere brugervenligt!

Den anden event handler – **søg** – ser sådan ud:

```
private void soegBtn_clicked(object o,EventArgs a){

   panel1.Text="Søger efter: "+fornavn.Text+" i databasen.";

                  //I stedet kunne anvendes static metode: File.OpenText():
                  //Her burde være exception handling!:

   FileInfo fil=new FileInfo("personer.txt");
   StreamReader reader=fil.OpenText();
   string ind=null;

                  //Læs hele filen:
   while((ind=reader.ReadLine())!=null){
      if(ind==fornavn.Text){
         efternavn.Text=reader.ReadLine();
         telefon.Text=reader.ReadLine();
         email.Text=reader.ReadLine();

      }
      else fornavn.Text="Personen blev ikke fundet!";
   }
   reader.Close();

}
```

NB Programmet er lavet enkelt på den måde at bruger **indtaster** et fornavn i den øverste tekstboks og klikker Søg!

Både denne og den anden event handler er gjort **private** – de er ikke ment som 'offentlige' metoder men som interne hjælpe metoder som programmet bruger!

Denne eventhandler åbner så den samme fil personer.txt og læser linje for linje. Hvis den finder en linje med det fornavn som brugeren har indtastet i øverste tekstboks udskriver den alle personens data.

Historisk er mange data blevet gemt på denne måde – men systemet er meget sårbart. Hvad sker der hvis fx. brugeren er kommet til at redigere i filen eller slette en linje i filen!?

Hele koden til programmet følger her (formen har også en statuslinje til informationer til bruger) – læg mærke til at det meste af koden er 'triviel' placering af kontrollerne osv:

```
//Fil: personer.cs

//Demo eksempel af form til indtastning
//og søgning af personer
//data gemmes i 'flad' fil:

using System;
using System.Windows.Forms;
using System.Drawing;
using System.IO;

public class Vindue : Form{

//constructor:

public Vindue(){

Text="Gem og Find Personer!";

//En farve kan oprettes med 3 værdier for rød, grøn og blå:

BackColor=System.Drawing.Color.FromArgb(110,150,110);
Size=new System.Drawing.Size(270,440);
CenterToScreen();

//En statuslinje bruges her til at informere brugeren:
status=new StatusBar();
status.Font=new Font("Times New Roman",10);

panel1=new StatusBarPanel();
panel2=new StatusBarPanel();

//OBS uden denne linje vises panelerne ikke!:
status.ShowPanels=true;

//OBS panelerne tilføjes statuslinjen ikke formen!:
status.Panels.Add(panel1);
status.Panels.Add(panel2);
panel1.Text="";
panel2.Text="";
panel1.Width=Width-70;
```

```
panel2.Width=70;

//OBS statuslinjen har ikke her nogen Size eller Location: dvs den sættes på som standard:
Controls.Add(status);

                    //Initialisering af formens komponenter:
   for_navn=new Label();
   for_navn.Text="Fornavn:";
   for_navn.Location=new Point(30,50);
   Controls.Add(for_navn);

   fornavn=new TextBox();
   fornavn.Location=new Point(30,80);
   fornavn.Size=new Size(200,30);
   Controls.Add(fornavn);

   efter_navn=new Label();
   efter_navn.Text="Efternavn:";
   efter_navn.Location=new Point(30,110);
   Controls.Add(efter_navn);

   efternavn=new TextBox();
   efternavn.Location=new Point(30,140);
   efternavn.Size=new Size(200,30);
   Controls.Add(efternavn);

   tlf=new Label();
   tlf.Text="Telefon:";
   tlf.Location=new Point(30,170);
   Controls.Add(tlf);

   telefon=new TextBox();
   telefon.Location=new Point(30,200);
   telefon.Size=new Size(200,30);
   Controls.Add(telefon);

   mail=new Label();
   mail.Text="E-mail:";
   mail.Location=new Point(30,230);
   Controls.Add(mail);

   email=new TextBox();
   email.Location=new Point(30,260);
   email.Size=new Size(200,30);
   Controls.Add(email);

   gemBtn=new Button();
   gemBtn.Text="Gem!";
   gemBtn.Location=new Point(120,290);
   gemBtn.Click+=new EventHandler(gemBtn_clicked);
   Controls.Add(gemBtn);

   soegBtn=new Button();
   soegBtn.Text="Søg!";
   soegBtn.Location=new Point(120,320);
   soegBtn.Click+=new EventHandler(soegBtn_clicked);
   Controls.Add(soegBtn);
```

```
}

//Event handler til knappen Søg:

private void soegBtn_clicked(object o,EventArgs a){
   panel1.Text="Søger efter: "+fornavn.Text+" i databasen.";

                  //I stedet for FileInfo kunne anvendes en static metode i klassen File: File.OpenText():
                  //(Resultatet er det samme!)

                  //Her burde være exception handling!:

   FileInfo fil=new FileInfo("personer.txt");
   StreamReader reader=fil.OpenText();
   string ind=null;

                  //Læs hele filen:
   while((ind=reader.ReadLine())!=null){
      if(ind==fornavn.Text){
         efternavn.Text=reader.ReadLine();
         telefon.Text=reader.ReadLine();
         email.Text=reader.ReadLine();

      }
   }
   reader.Close();

}

//Event handler til knappen Gem:

private void gemBtn_clicked(object o,EventArgs a){
   panel1.Text="Personen gemmes i databasen: personer.txt.";
   FileInfo fil=new FileInfo("personer.txt");

                  //nye personer appendes til filen:
   StreamWriter writer=fil.AppendText();
   writer.WriteLine(fornavn.Text);
   writer.WriteLine(efternavn.Text);
   writer.WriteLine(telefon.Text);
   writer.WriteLine(email.Text);
   writer.Close();

                  //Nulstil tekst bokse:
   fornavn.Text="";
   efternavn.Text="";
   telefon.Text="";
   email.Text="";

}

//Formens kontrol medlemmer:

private StatusBar status;
private StatusBarPanel panel1;
private StatusBarPanel panel2;
private TextBox fornavn,efternavn,telefon,email;
```

```
private Label for_navn,efter_navn,tlf,mail;
private Button gemBtn,soegBtn;


public static void Main(){
 Application.Run(new Vindue());
}
}
```

Alle formens kontroller er **private** – reglen i OOP er at hvis noget **kan** være private **skal** det være private for at sikre indkapsling og black box princippet!

Statuslinjen er tilføjet 2 paneler – dette er ikke strengt nødvendigt men giver et pænere billede og muliggør forskellige informationer i de to paneler.

Husk sætningen:

```
status.ShowPanels=true;
```

Ellers vises ingen af panelerne!

Som det ses er kodningen af **kontrollerne** rimelig enkel og kan klares med kopier – sæt ind!

Panelernes size kan sættes med en **AutoSize** f.eks. således:

```
pnlStatus.Text = "Ready";
pnlStatus.Icon = new Icon(Application.StartupPath + "\\active.ico");
pnlStatus.AutoSize = StatusBarPanelAutoSize.Contents;
StatusBarPanel pnlConnection = new StatusBarPanel();
pnlConnection.Text = "Connected to localhost";
pnlConnection.AutoSize = StatusBarPanelAutoSize.Spring;

statusBar.SizingGrip = false;
```

Her er også vist hvordan man kan sætte et ikon i et panel. I det viste eksempel får det 1. panel en min størrelse, mens det 2. panel udfylder resten af statuslinjen. Desuden er fjernet den 'SizingGrip' som normalt sættes på statuslinjen – for at give et pænere resultat.

Opgaver:

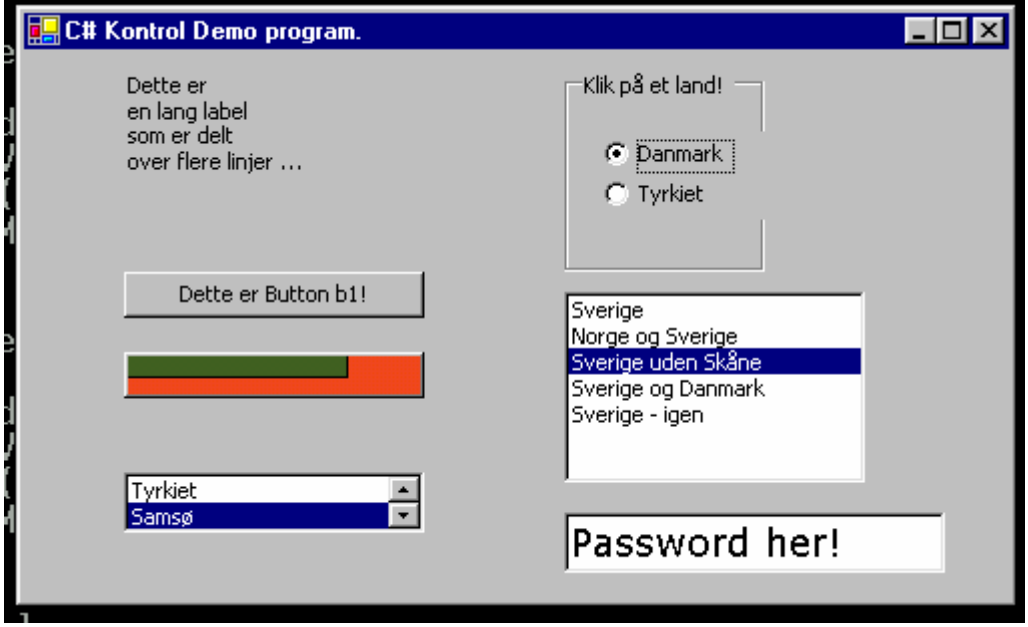
1. Hvordan man gøre gemme processen i filen mere **'sikker'** så evt fejl i filen ikke fik så stor betydning?
2. Hvordan kunne der skrives en **exception** handling til programmet? Hvad er **problemet** uden en sådan exception handling?
3. Skriv programmet om så der gives mere information i status **panelerne**!
4. Skriv et formular program som kan gemme udlån på et **bibliotek**!

Sidespring: Demo af forskellige C# kontroller:

Følgende kode viser hvordan en række af de vigtigste Windows **kontroller** kan anvendes på en form. Som et lille side spring vil vi derfor kort se på de vigtigste funktioner omkring disse komponenter eller kontroller. Programmet er gemt i filen 'dethele.cs' og har samme struktur som de øvrige Windows programmer – **dels** en app klasse **dels** en ny form klasse som arver fra Form. Som det kan ses, kan det meste af koden klares med kopier og sæt ind – så koden er væsentligt nemmere at producere end man umiddelbart får indtryk af:

```
//fil:dethele.cs

//demo af forskellige Windows kontroller

//postcondition: viser et vindue med mange komponenter
//der er ingen event handlere så kontrollerne 'gør ingenting'

using System.Windows.Forms;
using System.Drawing;

//vores egen definerede form: XForm

public class XForm : Form {

        //klassens instans variable/egenskaber/komponenter:
        private Label label;
        private Button b1,b2;
        private TextBox tekst;
        private RadioButton radio1,radio2;
        private GroupBox gruppe;
        private ListBox lille,stor;


        //constructor til XForm:
        public XForm(){

        Size=new Size(500,300);
        Text="C# Kontrol Demo program.";
        CenterToScreen();

        //komponenter:
        label=new Label();
        label.Text="Dette er \nen lang label \nsom er delt \nover flere linjer ...";
        label.Location=new Point(50,10);
        label.Size=new Size(100,100);
        Controls.Add(label);

        tekst=new TextBox();
        tekst.Text="Password her!";
        tekst.Location=new Point(270,230);
        tekst.Width=190;
        tekst.Font=new Font("Verdana",14);
        Controls.Add(tekst);
```

```
b1=new Button();
b1.Text="Dette er Button b1!";
b1.Width=150;
b1.Location=new Point(50,110);
Controls.Add(b1);

b2=new Button();
//b2.Text="Dette er Button b1!";
b2.Width=150;
b2.Location=new Point(50,150);
b2.Image=Image.FromFile("nr1.jpg");
Controls.Add(b2);

gruppe=new GroupBox();
gruppe.Location=new Point(270,10);
gruppe.Size=new Size(100,100);
gruppe.Text="Klik på et land!";
Controls.Add(gruppe);

radio1=new RadioButton();
radio1.Location=new Point(20,30);
radio1.Text="Danmark";
gruppe.Controls.Add(radio1);

radio2=new RadioButton();
radio2.Location=new Point(20,50);
radio2.Text="Tyrkiet";
gruppe.Controls.Add(radio2);

lille=new ListBox();
lille.Location=new Point(50,210);
lille.Size=new Size(150,30);
lille.SelectionMode=SelectionMode.One;
lille.BeginUpdate();
lille.Items.Add("Danmark");
lille.Items.Add("Tyrkiet");
lille.Items.Add("Samsø");
lille.EndUpdate();
lille.SelectedIndex=2;
Controls.Add(lille);

stor=new ListBox();
stor.Location=new Point(270,120);
stor.Size=new Size(150,100);
stor.SelectionMode=SelectionMode.One;
stor.BeginUpdate();
stor.Items.Add("Sverige");
stor.Items.Add("Norge og Sverige");
stor.Items.Add("Sverige uden Skåne");
stor.Items.Add("Sverige og Danmark");
stor.Items.Add("Sverige - igen");

stor.EndUpdate();
stor.SelectedIndex=2;
Controls.Add(stor);

}
```

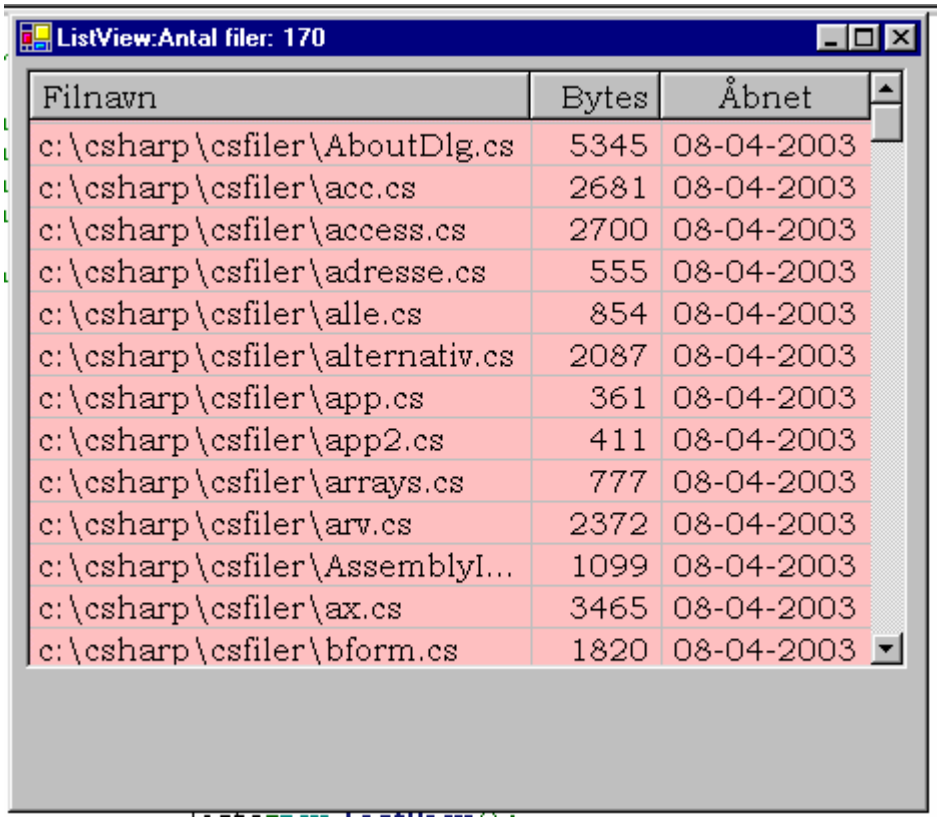



```
}

//applikation - i samme fil - som anvender klassen XForm:
class app{
                public static void Main(string[] args){
                Application.Run(new XForm());
                }
}
```

Når programmet køres fås følgende resultat:

ListView:

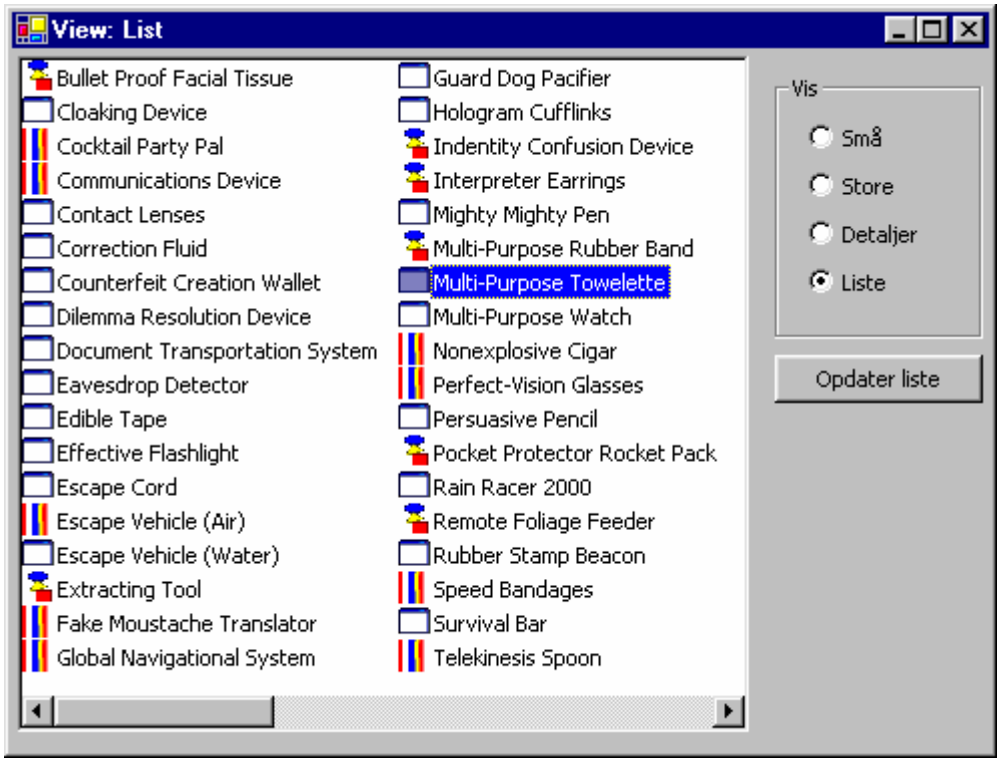
En System.Windows.Forms.**ListView** kan vise data i kolonner og rækker på en langt mere fleksibel måde end en almindelig liste. F.eks. kan den vise data fra en mappe på computeren således:



En ListView kan tilpasses på et utal af måder.

På http://csharpkursus.subnet.dk ligger to eksempler på hvordan man kan anvende en ListView kontrol – dels et enkelt, dels et mere omfattende eksempel.

Det andet ListView eksempel (med sortering, ikoner, forskellige Views osv) viser denne form:



Dette eksempel udnytter en række indbyggede muligheder i ListView – **for eksempel**:

```
listview.Activation = ItemActivation.OneClick;
listview.AllowColumnReorder = true;
listview.GridLines = true;
listview.HoverSelection = true;
listview.Sorting = SortOrder.Ascending;

listview.ColumnClick += new ColumnClickEventHandler
(listview_ColumnClick);
listview.SmallImageList = smaa_ikoner;
listview.LargeImageList = store_ikoner;
```

Data hentes i en **XML** fil 'store.xml' (meget vittig!) på denne måde:

```
public class Database
{
        public static DataTable GetProducts()
        {
                DataSet dataset = new DataSet();
                dataset.ReadXmlSchema(Application.StartupPath +
"\\store.xsd");
                dataset.ReadXml(Application.StartupPath + "\\store.xml");
                return dataset.Tables["Products"];
        }
}
```

En ListView har en property **View,** som kan sættes fx sådan:

```
private void skift_view(object sender, System.EventArgs e)
{
        //kontrollerne har også en Tag som frit kan defineres!:
        listview.View = (View)(((Control)sender).Tag);
}
```

Hvis vi så har defineret to radio buttons med en Tag property således:

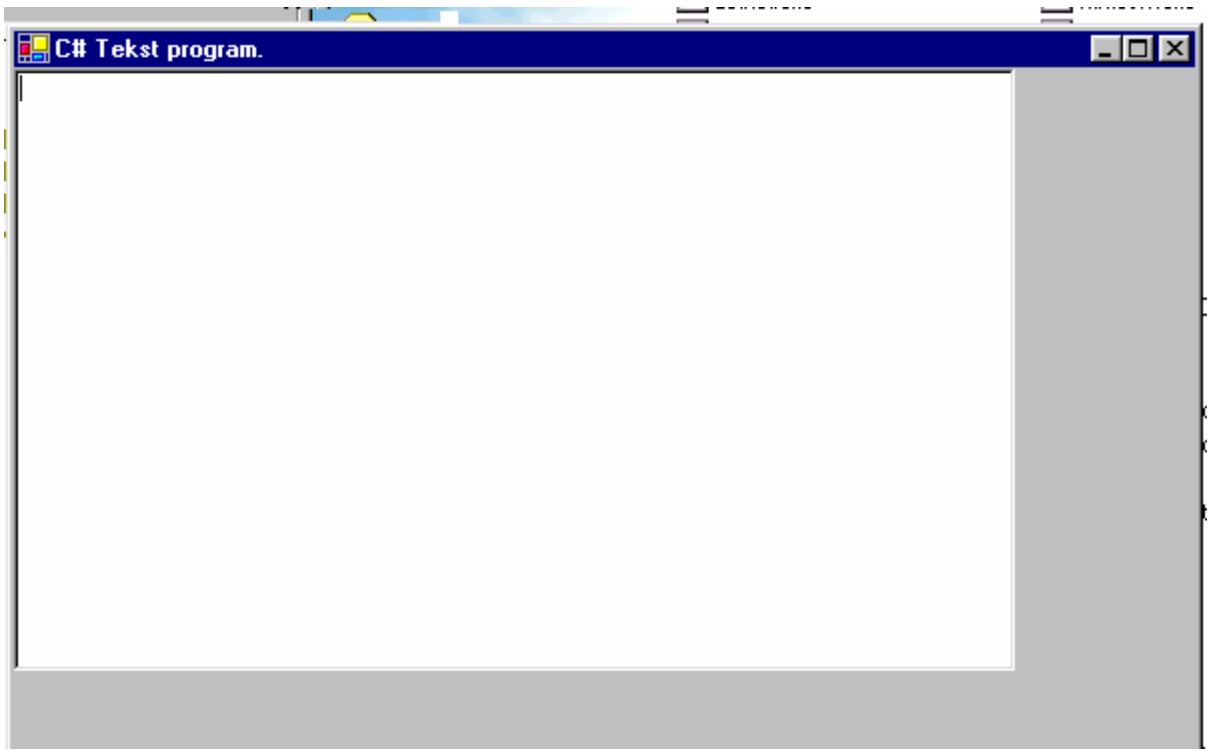
```
radio_store.Tag = View.LargeIcon;
radio_smaa.Tag = View.SmallIcon;
```

kan vi skifte View ved at klikke i disse.

Som det ses, er ListView forsynet med rigtig mange indbyggede metoder, som gør en ListView til et **meget** anvendeligt, fleksibelt værktøj!

Generelle problemer: Når formen ændrer størrelse og hvad der så sker:

De hidtil omtale Windows projekter lider under en generel svaghed. Formen 'fungerer' rimeligt nok lige når den bliver vist – men hvad sker når den maksimeres eller resizes?



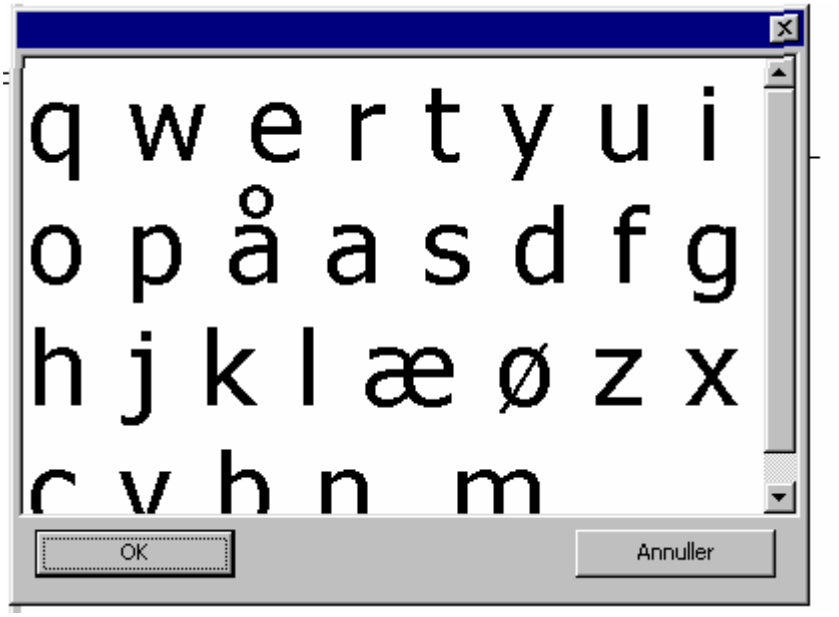
Som det ses, sker der det, at tekstboksen (der egentligt 'skulle' fylde det hele) kommer til at hænge i det øverste venstre hjørne og resten af formen bliver 'meningsløs'! Hvis vi havde anbragt andre kontroller på formen ville resultatet have været endnu værre!

Vi skal derfor i første omgang afgøre om vores form overhovedet skal **kunne** ændre størrelse! En 'dialogboks' kan fx sjældent ændre størrelse. Der er i C# en række muligheder - illustreret med dette kode fragment:

```
//kan minimeres og maximeres, men ikke resizes, 3D kant:
//FormBorderStyle=FormBorderStyle.Fixed3D;

//kan også resizes i kanten - single linje kant:
FormBorderStyle=FormBorderStyle.Sizable;

//kan ikke resizes i kanten, men max og min, enkelt linje kant:
//FormBorderStyle=FormBorderStyle.FixedDialog;

//bruges med en Timer som lukker formen, splash screen: Luk Alt F4
evt
//FormBorderStyle=FormBorderStyle.None;

//kun lille Luk knap, kan trækkes i kanten
//FormBorderStyle=FormBorderStyle.SizableToolWindow;

//eks på dialog boks i C#:

//FormBorderStyle=FormBorderStyle.Fixed3D;//evt - med ikon
//FormBorderStyle=FormBorderStyle.FixedDialog;//normalt
//MaximizeBox=false;
//MinimizeBox=false;
```

En form har altså en værdi hentet fra en **enumeration** FormBorderStyle, som bestemmer de overordnede egenskaber ved formen. Endvidere kan de to 'standardknapper' (Maksimer, Minimer) evt. fjernes.

Et eksempel på en **'dialogboks'** (med den kode som stod ovenfor) i C# ville være:



Her er valgt **FixedDialog**. Hvis der vælges **Fixed3D** får boksen også et ikon. Begge er hvad man plejer at kalde 'dialogbokse' – selv om der ikke i C# er noget som hedder en dialogboks: En form er en form!

Som det ses er der ingen Min eller Max knap og formen kan IKKE resizes ved at trække i kanterne! På denne måde kan vores resize problem altså løses.

En mere **fleksibel** løsning kan også findes fordi alle kontroller har et 'anker' eller **Anchor**. Dette er illustreret i dette kodestykke:

```
ok=new Button();
ok.Text="OK";
ok.Size=new Size(100,24);
ok.Location=new Point(10,240);

//anker egenskaben:
//ok.Anchor=AnchorStyles.Right|AnchorStyles.Bottom;
//uanset resize holder knappen samme afstand til FORMENS venstre og
bund kant!!
ok.Anchor=AnchorStyles.Left|AnchorStyles.Bottom;
//ok.Anchor=AnchorStyles.Left|AnchorStyles.Top;
Controls.Add(ok);
```

Forklaring:

Knappen anbringes først 10 pixels hen og 240 pixels ned på formen. Derefter **ankres** den til **alle** sider (men man kunne selvfølgeligt nøjes med at ankre den til en eller to af formens sider!). Dette betyder at uanset om formen resizes vil denne knap altid beholde den samme afstand til formens 4

kanter. Værdierne findes i en enum **AnchorStyles** men man kan sikkert hurtigt eksperimentere sig frem til hvad der er ønskeligt.

Vi gør nu formen sizable og resultatet er:

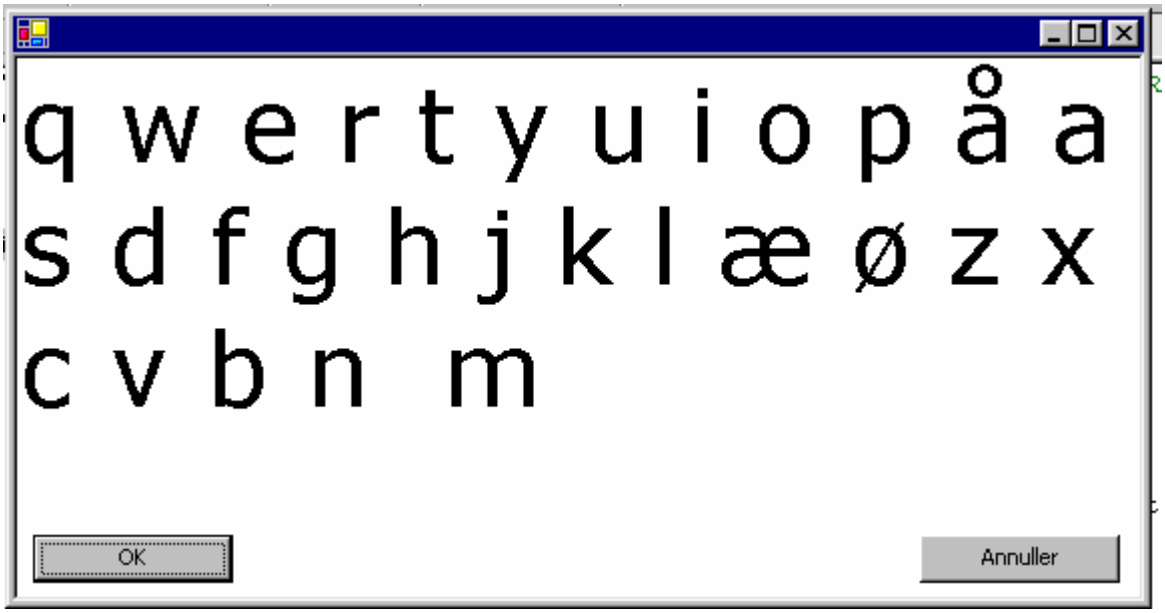
Formen kan nu resizes og maksimeres og kontrollerne **holder** deres indbyrdes relationer.Ok og Annuller knappen 'forsvinder' ikke – som det ellers **ville** ske – hvis formen gøres mindre af bruger!!

Hvis et program skal fungere er det fx **helt afgørende** at en OK eller Luk knap ikke pludselig forsvinder ud af brugerens vindue!!

Et andet redskab er at sætte en kontrols property **Dock**. I nedenstående eksempel er tekstboksen sat til:

tekst.Dock=DockStyle.**Fill**;

Dette bevirker at hele fladen fyldes med tekstboksen således:



Hvis formen resizes bliver tekstboksen **ved** med at fylde hele formen ud: Dette løser altså også på en vis måde vores resize problem!

Andre værdier for DockStyle er:

DockStyle.Top, DockStyle.Bottom, DockStyle.Left, DockStyle.Right.

Layout og paneler:

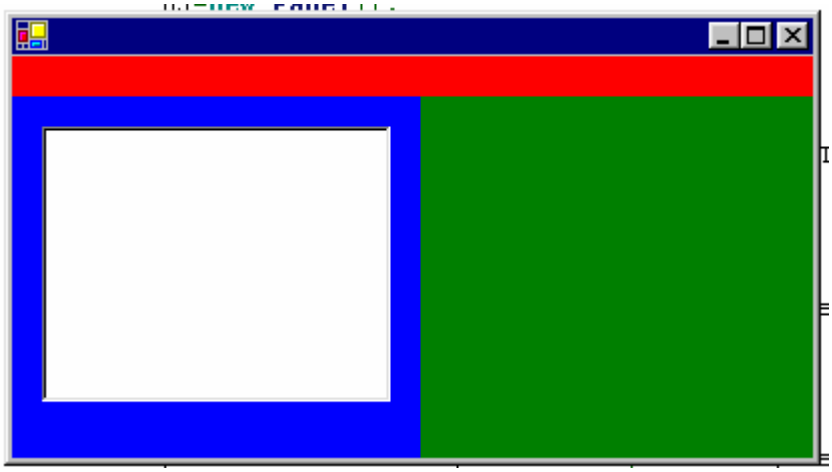
Det sikreste måde at styre formens **layout** på er at opdele formens flade i **paneler**. Metoden er at man starter med at tegne formens layout med blyant og papir!

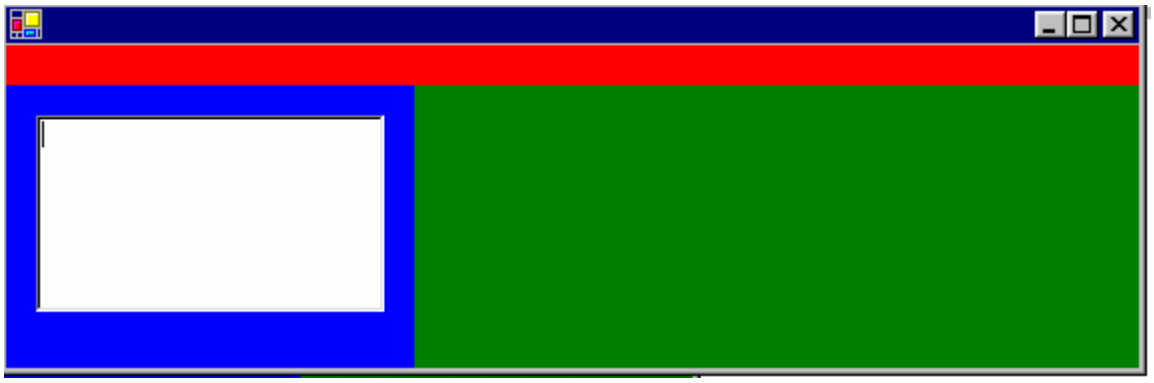
I det følgende eksempel er formen delt i 3 paneler: et **toppanel** (der uanset hvad beholder sin placering og højde), et **venstre** panel (som beholder sin bredde og 'klæber' til højre) og et højre panel (som udvides når formen gøres større). Når formen på den måde har fået en grundlæggende layout kan man tilføje kontroller - **ikke** til selve formen – men til et af **panelerne**!! Dette er vist med en tekstboks som er tilføjet venstrepanelet:

```
// Eksempel: Paneler til at styre layout paa form:


using System;
using System.Windows.Forms;
using System.Drawing;


        class PanelForm : Form
        {
                Panel p1,p2,p3;
                RichTextBox tekst;
```

```
		public PanelForm()
		{
			ClientSize=new Size(400,200);
			p1=new Panel();
			p2=new Panel();
			p3=new Panel();

			p1.Size=new Size(400,20);
			p1.BackColor=Color.Red;
			p1.Location=new Point(0,0);
			p1.Anchor=AnchorStyles.Left|AnchorStyles.Top|AnchorStyles.Right;

			p2.Size=new Size(Width/2,Height-20);
			p2.BackColor=Color.Blue;
			p2.Location=new Point(0,20);

	p2.Anchor=AnchorStyles.Bottom|AnchorStyles.Left|AnchorStyles.Top;

			p3.Size=new Size(Width/2,Height-20);
			p3.BackColor=Color.Green;
			p3.Location=new Point(Width/2,p1.Height);

	p3.Anchor=AnchorStyles.Bottom|AnchorStyles.Left|AnchorStyles.Top|AnchorStyles.Right;

			tekst=new RichTextBox();
			tekst.Size=new Size(p2.Width-30,p2.Height-70);
			tekst.Location=new Point(15,15);

	tekst.Anchor=AnchorStyles.Right|AnchorStyles.Bottom|AnchorStyles.Left|AnchorStyles.Top|AnchorSt
yles.Right;

			p2.Controls.Add(tekst);

			Controls.Add(p1);
			Controls.Add(p2);
			Controls.Add(p3);

			//I stedet kunne skrives:
			//Controls.AddRange(new Panel[]{p1,p2,p3});

		}


		[STAThread]
		public static void Main(string[] args)
		{
			Application.Run(new PanelForm());
		}
	}
```

Formen kan så se fx sådan ud:





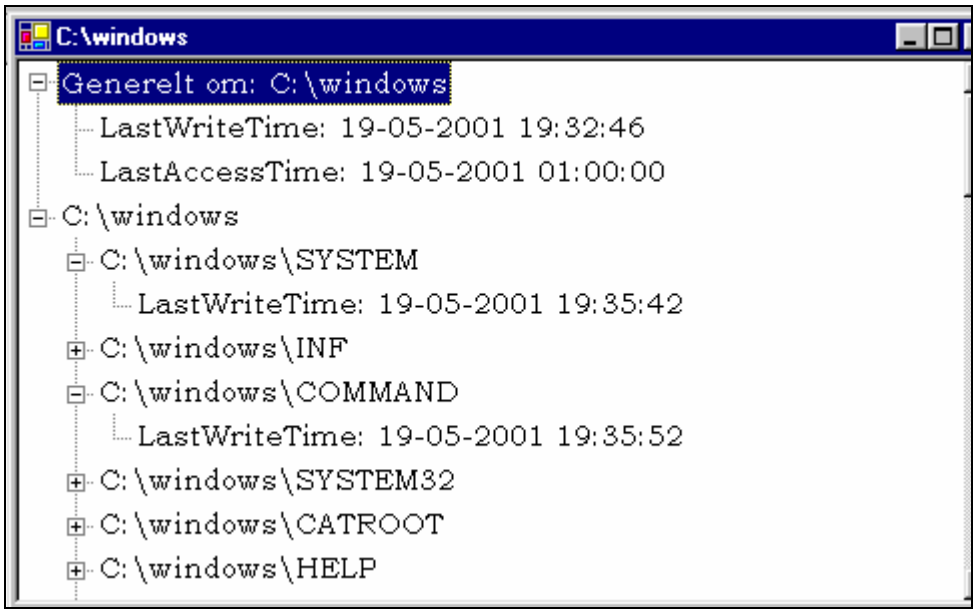
Opgaver:

1. Skriv et program med to rich tekstbokse som vises til højre og venstre på formen (del formen midt over) og som bevarer deres relationer når formen ændrer størrelse!
2. Skriv et program med en OK og Annuller knap **over** hinanden til højre (som er skik og brug) som bevarer deres positioner når formen resizes!
3. Der findes forskellige slags docking: knap.Dock=DockStyle.Top, knap.Dock=DockStyle.Left osv. Skriv et program hvor du docker 2 knapper og en tekstboks på denne måde! Hvad bliver resultatet!? Hvad kan det bruges til?
4. Skriv et program som åbner **forskellige** forms med forskellige borderstyles – jvf koden ovenfor!
5. Skriv et program hvor du opdeler formen i **paneler** og tilføjer kontroller til panelerne!

Eksempel: TreeView kontrollen:

System.Windows.Forms indeholder bl.a. en TreeView kontrol som kan vise **træ** strukturer som oprettes enten 'automatisk' eller manuelt. En TreeView kan således vise strukturerede data fra en XML fil.

I det følgende eksempel genbruger vi vores kode fra et tidligere program til at vise en mappe struktur i en TreeView således:

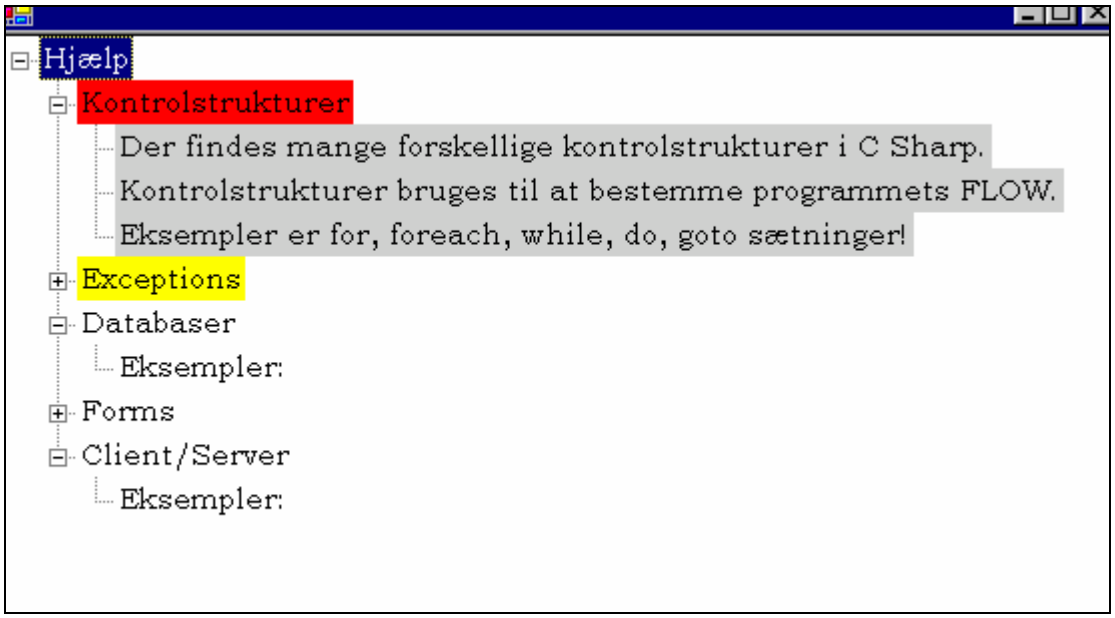


En **Node** i en Treeview er et **punkt** som kan åbnes hvis det har under punkter (jvf + og – knapperne). En Treeview kan have een root node **eller** mange. Træ strukturen oprettes ved at betegne noderne hierarkisk med tal som det fremgår af dette kode fragment:

```
                                        tree=new TreeView();
                                        tree.Font=new Font("Bookman Old Style",11);
                                        tree.Dock=DockStyle.Fill;
                                        dir=new DirectoryInfo("C:\\windows");

                                        //1. node
                                        tree.Nodes.Add(new TreeNode("Generelt om: "+dir.FullName));
                                        Text=dir.FullName;
                                        //2. node
                                        tree.Nodes.Add(new TreeNode(dir.FullName));
                                        DirectoryInfo[] subdirs=dir.GetDirectories();
                                        int i=0;

                                        foreach(DirectoryInfo d in subdirs){
                                                ny=new TreeNode(d.FullName.ToString());

                                                //sub noder under node 1:
                                                tree.Nodes[1].Nodes.Add(ny);

        tree.Nodes[1].Nodes[i++].Nodes.Add("LastWriteTime: "+d.LastWriteTime.ToString());
                                        }

                                        //Sub noder under node 0:
                                        tree.Nodes[0].Nodes.Add("LastWriteTime:
"+dir.LastWriteTime.ToString());
                                        tree.Nodes[0].Nodes.Add("LastAccessTime:
"+dir.LastAccessTime.ToString());


                                        Controls.Add(tree);
```

Opgaver:

1. Skriv en form med en TreeView som viser en form for hierarkisk præsentation af programmeringssproget C#! (Basis noder kan være 'Kontrolstrukturer', 'Exceptions' eller lignende).
2. Hvordan kunne du i øvrigt tænke dig en TreeView anvendt? Skriv et eksempel!
3. Skriv et program med en slags Hjælp (stikord) hvor basisnoderne er hovedemner i Hjælp! (F. eks. sådan):



NB Både den samlede TreeView og de enkelte TreeNode'r kan formatteres som vist her. Når du opretter en ny node skal du starte med at instantiere således:

```
TreeNode ny=new TreeNode();
ny.Text=" ... ";
```

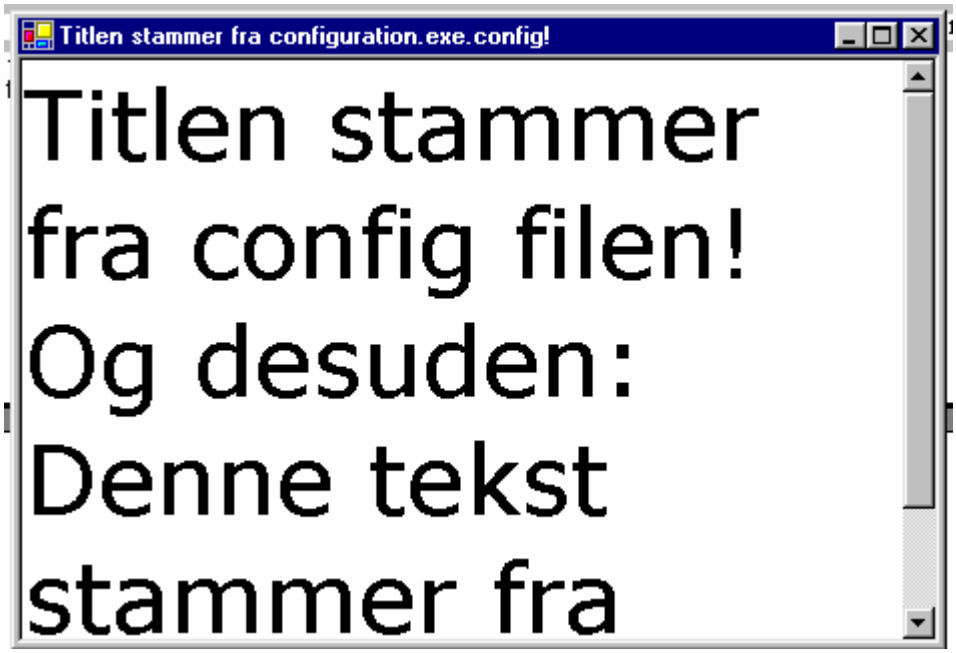
Derefter kan den nye node ny **formatteres** med farver, fonts mv.

Anvendelse af exe.config filer til konfiguration af programmet:

Ofte er man i den situation at man ønsker at kunne konfigurere et program uden at skrive koden om og uden at skulle kompilere programmet igen. I C# er der en mulighed for at konfigurere sit program forskelligt fra gang til gang.

Metoden er at der til hver EXE fil kan skrives en config fil (som er en ganske almindelige tekst fil i XML format) med samme navn (endelig!). Denne config fil definerer så værdier som programmet læser når det starter op.

Følgende eksempel viser en Windows form hvor **titellinjen** og **tekstboksens** indhold stammer fra config filen – er altså **ikke** 'hard' kodet ind i programmet!:



Programmet hedder **configuration.exe,** og derfor **skal** XML filen hedde **configuration.exe.config**:

```
<?xml version="1.0"?>
<configuration>
  <appSettings>
                <add key="titel" value="Titlen stammer fra configuration.exe.config!" />
                <add key="tekst" value="Titlen stammer fra config filen! Og desuden: Denne tekst
stammer fra config filen!" />

  </appSettings>
</configuration>
```

OBS: Filen **skal** have præcist disse elementer for at fungere som config fil, og den **skal** have det rigtige navn, og den **skal** ligge i samme mappe som EXE filen!

Filen opretter **keys** med tilhørende værdier. Dette kan så udnyttes i C# koden i programmet således (her er kun medtaget den centrale kode i programmet hvor kontrollerne oprettes):

```
void InitializeComponents()
{
this.SuspendLayout();
this.Name = "MainForm";
string s=System.Configuration.ConfigurationSettings.AppSettings["titel"];

this.Text = s;
```

```
this.Size = new System.Drawing.Size(500, 300);
tekst=new RichTextBox();
tekst.Dock=DockStyle.Fill;
tekst.Font=new Font("Verdana",36);
s=System.Configuration.ConfigurationSettings.AppSettings["tekst"];
tekst.Text=s;
Controls.Add(tekst);
this.ResumeLayout(false);
}
```

Config utility:

Det er meget vigtigt, at hvis data hentes fra en config fil, så **skal** programmet på en eller anden måde forudse **exceptions** – dvs at disse værdier af en eller anden grund **ikke** kan hentes i config filen! Brugeren kan jo fx have slettet config filen!!

På http://csharpkursus.subnet.dk ligger et lille program, som gør det nemmere at anvende data fra en konfigurations fil. Hvis der refereres til denne 'utility' (config.dll) kan man fx skrive kode på denne måde – idet der altid sættes en 'default' værdi, hvis data ikke kan hentes i XML config filen (!):

```
public class demo : Form {

        public demo(){
                BackColor=config.GetAppSetting("color",Color.Blue);
                Text=config.GetAppSetting("titel","INGEN TITEL");
                Width=(int)config.GetAppSetting("width",500);
                MaximizeBox=config.GetAppSetting("max",true);


        }
        public static void Main(){
                Application.Run(new demo());
        }

}
```

Opgaver:

1. Opret en form og en config fil til formen. Brug fantasien og prøv at gøre formen så fleksibel som mulig ved at konfigurere farver, tekster osv i en config fil.
2. Skriv config filen sådan at den bestemmer om bestemte kontroller fx knapper overhovedet skal vises eller ikke vises og med hvilken tekst!
3. Skriv et program som selv skriver en configfil hvis den ikke findes og som altid gemme visse værdier når det lukker!

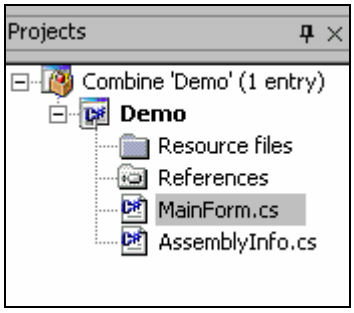
Windows projekt i SharpDevelop:

Meget kan gøres nemmere ved at bruge en IDE som SharpDevelop eller VS.NET til at udvikle Windows programmer. Meget af det manuelle arbejde på kommandolinjen kan en builder gøre 'automatisk'.

Start et projekt i Sharpdevelop med Filer->New Combine->Windows Form Project i C# - fx ved navn **Demo**.

Vælg auto create sub directories og programmet opretter en ny mappe 'Demo' til alle de nye projekt filer!

Vælg View->Projects og du vil se den struktur, der automatisk og altid produceres:



En 'Combine' kan indeholde en række filer, ressourcer (det gemmer vi til lidt senere) og hele projekter. Hvis du dobbelt klikker eller højre klikker på filer kan du få dem vist i vinduet til højre.

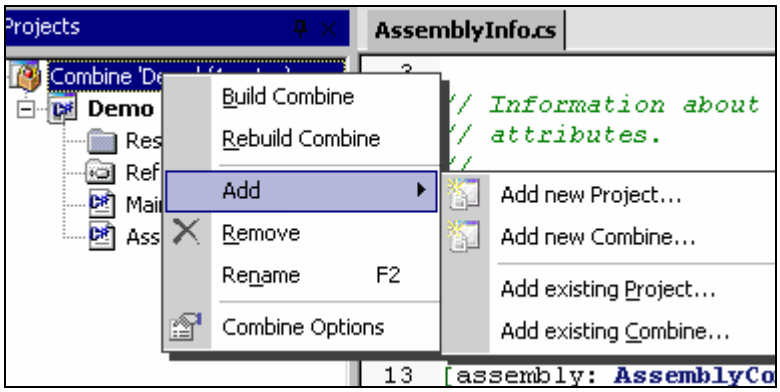
Interessant er også at der oprettes en **AssemblyInfo** fil. Heri kan du sætte en række egenskaber – attributter - for hele projektet fx version, kultur, firmanavn osv osv. Fx sådan:

```
7  // change them to the information which is associated with th
8  // you compile.
9
0  [assembly: AssemblyTitle("DEMO")]
1  [assembly: AssemblyDescription("Demo af SharpDevelop som IDE"
2  [assembly: AssemblyConfiguration("")]
3  [assembly: AssemblyCompany("")]
4  [assembly: AssemblyProduct("DEMO 4.4")]
5  [assembly: AssemblyCopyright("Computer APS 2003 DK")]
6  [assembly: AssemblyTrademark("CAXYZ")]
7  [assembly: AssemblyCulture("")]
8
```

Kun en DLL kan have en Culture. En CultureAttribute består af en værdi som "en" eller "dk" eller en kombineret sprog-geografi værdi som "en-US" eller "da-DK".

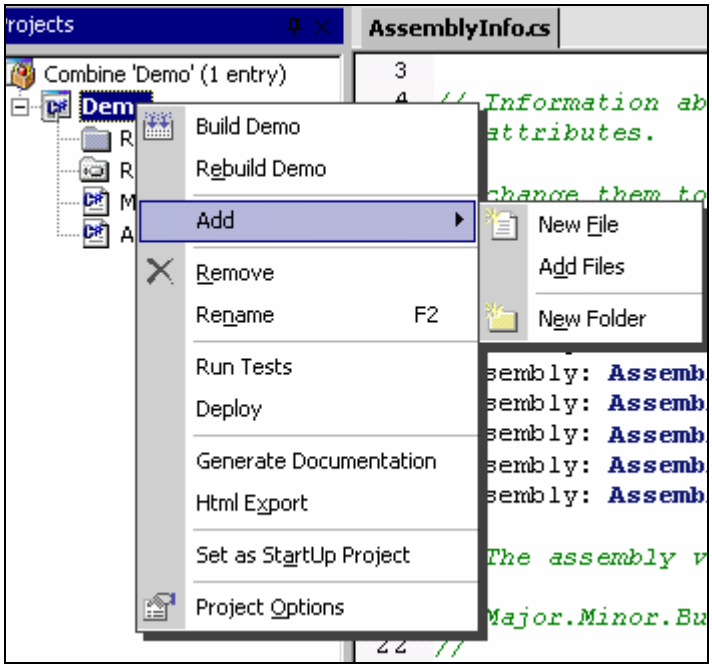
Når du kompilerer projektet bliver disse data gemt som meta data i selve filen/assembly'en!

Hvis man højreklikker på 'Combine' fås en række muligheder:

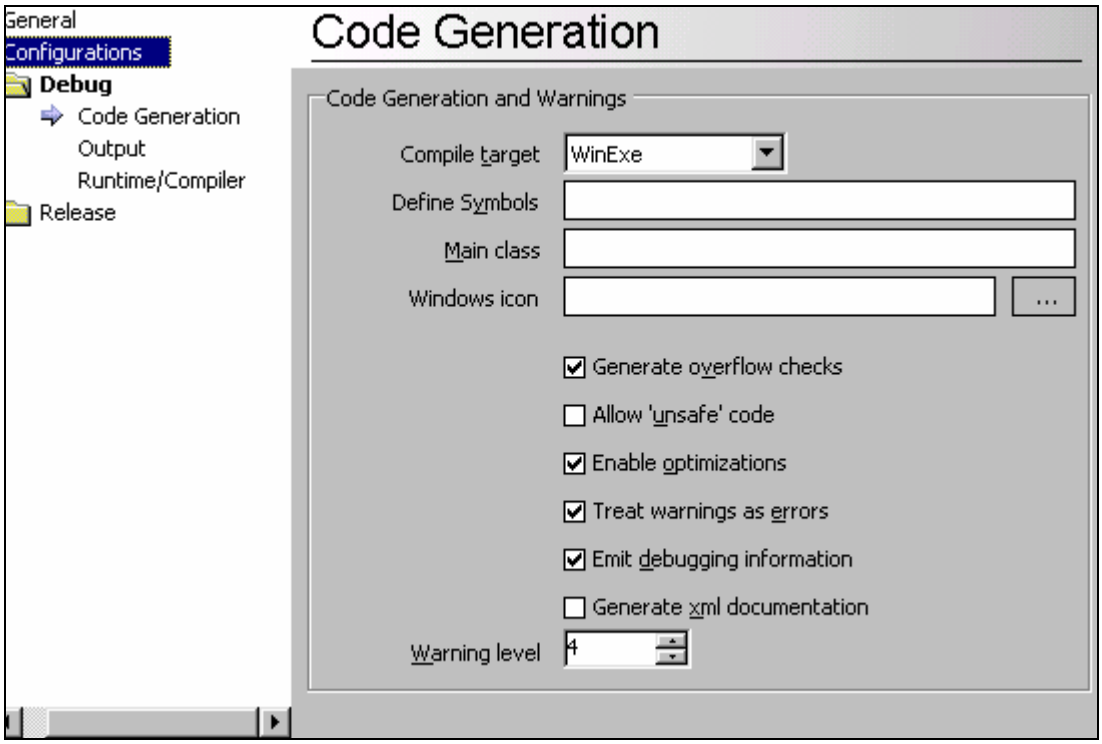


Her kan man altså tilføje hele nye projekter og builde Combine og dens ressourcer osv. Hvis man vælger Combine Options kan man vælge hvordan hele Combine'n skal opstarte hvis den indeholder flere projekter.

Hvis man højreklikker Demo (projektet) fås flere valg muligheder:



Der findes i Sharpdevelop en test enhed hvor et projekt kan testet. Der kan produceres HTML dokumentation (Java style) eller XML C# dokumentation. Hvis man vælger Project Options (ikke det samme som Combine Options!) fås en række muligheder for at konfigurere projektet:



Man kan her 'definere' symboler (se afsnittet om Debugging!), vælge et ikon til projektet eller at få produceret XML dokumentation samtidigt med kompileringen. Man kan vælge target – det som ellers sker på kommandolinjen med /target:winexe (f.eks).

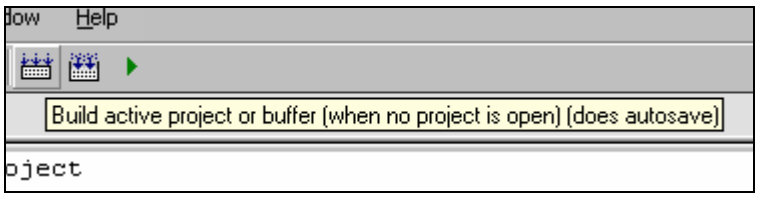
Hvis man vælge 'optimering' bliver koden optimeret dvs gjort kortere og mere effektiv af JIT kompileren.

Hvis man vælge at warnings skal tælle som errors kan man ikke køre programmet i Sharpdevelop hvis kompileringen har givet advarsler osv osv.

Den Main.cs som IDE'en producerer er **ALTID** ens:

```
namespace MyFormProject
{
    class MainForm : Form
    {
        public MainForm()
        {
            InitializeComponents();
        }

        // This method is used in the forms desi
        // Change this method on you own risk
        void InitializeComponents()
        {
            //
            //  Set up generated class MainForm
            //
            this.SuspendLayout();
            this.Name = "MainForm";
            this.Text = "This is my form";
            this.Size = new System.Drawing.Size(
            this.ResumeLayout(false);
        }

        [STAThread]
        public static void Main(string[] args)
        {
            Application.Run(new MainForm());
```

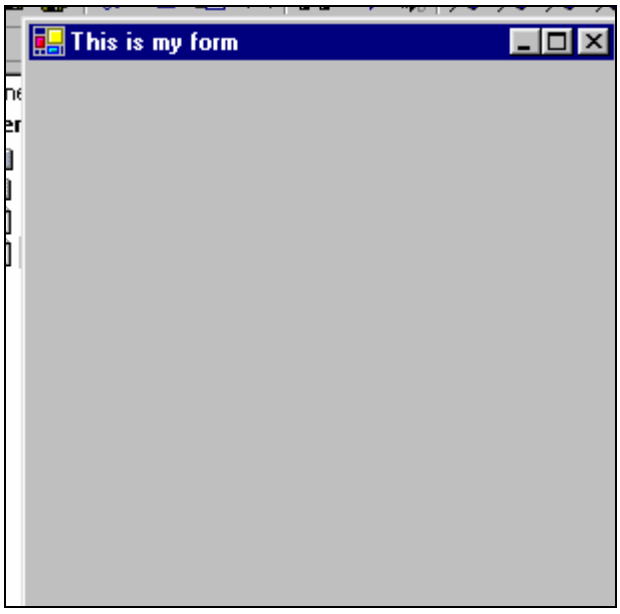
Formen kan kompileres og køre som den er med de 3 værktøjs linje knapper:



Som det fremgår builder/kompilerer den **venstre** knap samtidigt med at den gemmer den reviderede fil!

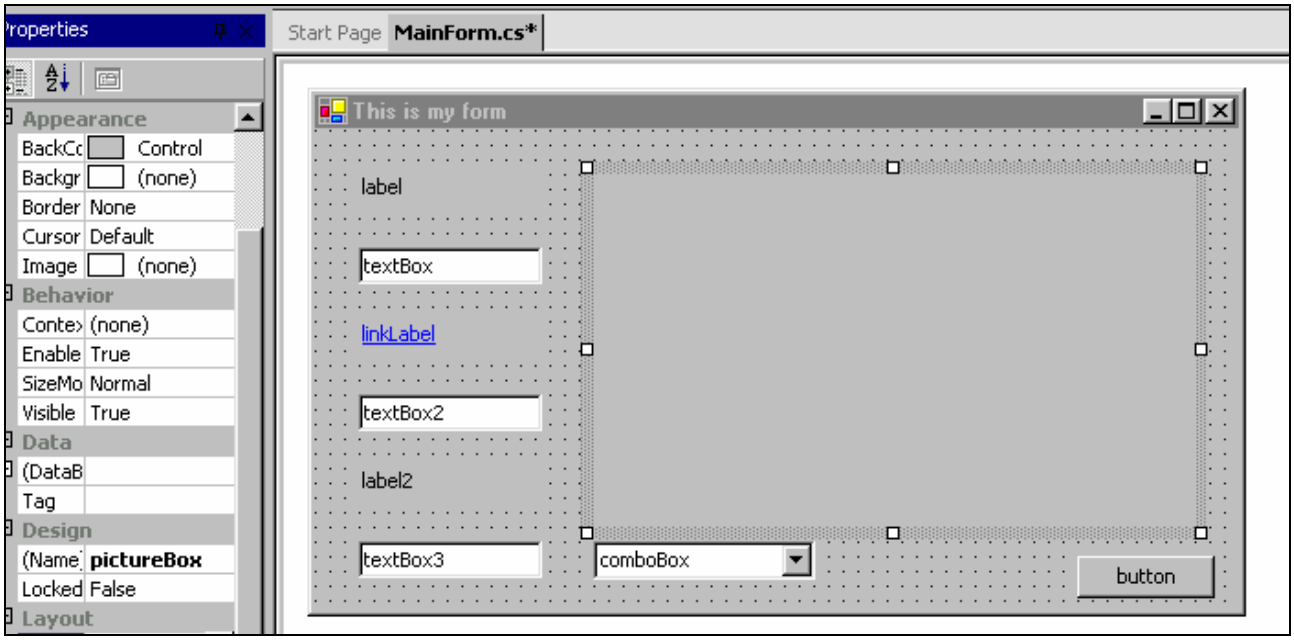
Vælg View->Output og du kan se evt. fejlmeldinger ved kompileringen. Disse vises også grafisk ved **røde** bølgestreger i koden!!

```
    }

    [STAThread]
    publi static void Main(string[] args)
    {
        Application.Run(new MainForm());
    }
}
```



Dette er grundlæggende hvad Sharpdevelop leverer i første omgang! Men klik Design i stedet for Source og der kommer en grafisk builder flade frem:

Vælg View->Tools og du kan nu **klikke** på kontroller og droppe dem på formen direkte!:

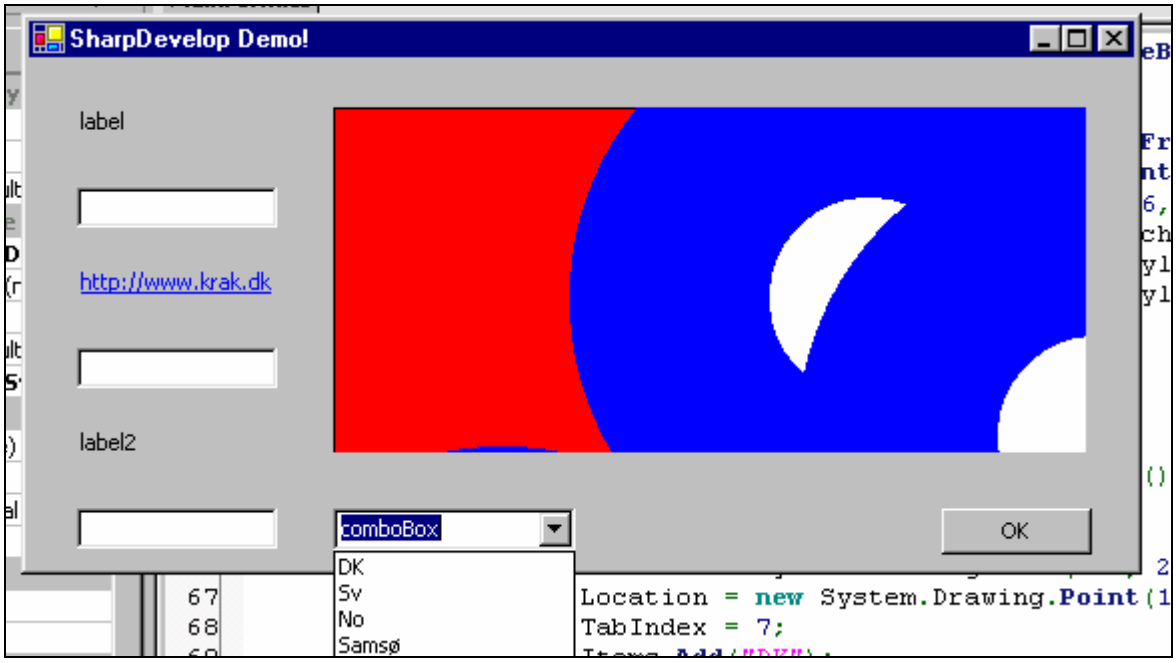


Hvis du fortryder kan du markere kontrollen og taste Delete. Du kan flytte kontrollerne rundt med musen.

Vælg View->**Properties** og marker en kontrol og sæt dens egenskaber! OBS: En del egenskaber kan slet **ikke** sættes direkte i builderen, men skal sættes manuelt direkte i kildekoden!!

Men man **sparer** arbejde i en del tilfælde og kan måske bedre overskue GUI Den grafiske brugergrænseflade. Man kan **direkte** kompilere og køre Windowsprogrammer og undgå besværlige ordrer på kommandolinjen! Man kan meget nemmere administrere et stort projekt.

Herudover kan man tilføje referencer til DLL filer ved at højre klikke på **references** i listen og **ressourcer** kan ligeledes direkte tilføjes.



SharpDevelop fungerer **ikke** altid lige optimalt som builder program og har det med at bryde sammen under byrderne - men kan bruges til begrænsede opgaver. Fordelen er at alle kontrollernes **egenskaber** er lette at finde. Og programmet er **gratis**.

SharpDevelop producerer også al den **'trivielle'** kode som f.eks. følgende **eksempler** – selv om navnene ikke ligefrem er 'sigende'!:

```
private System.Windows.Forms.Label label;
private System.Windows.Forms.TextBox textBox;
private System.Windows.Forms.LinkLabel linkLabel;
private System.Windows.Forms.TextBox textBox2;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.TextBox textBox3;
private System.Windows.Forms.ComboBox comboBox;
private System.Windows.Forms.PictureBox pictureBox;
private System.Windows.Forms.Button button;
```

```
// Set up member textBox3
//
textBox3 = new System.Windows.Forms.TextBox();
textBox3.Name = "textBox3";
textBox3.Text = "";
textBox3.Location = new System.Drawing.Point(24, 224);
textBox3.TabIndex = 6;
this.Controls.Add(textBox3);
```

```
//
//  Set up member label2
//
label2 = new System.Windows.Forms.Label();
label2.Name = "label2";
label2.Location = new System.Drawing.Point(24, 184);
label2.TabIndex = 5;
label2.Text = "label2";
this.Controls.Add(label2);
```

Microsofts foretrukne builder værktøj er **Visual Studio** .NET som bestemt **ikke** er gratis men som kan prøves på nettet i en 3 timers version! I visse perioder kan man også downloade en evaluation kopi af VS.NET. VS.NET er et langt mere professionelt og stabilt værktøj.

Delegater, events og event handlers

Vi har mange gange set eksempler på såkaldte **events** i Windows programmeringen. Hvis der klikkes på en knap er det en event i programmet og der kan skrives en passende event handler som definerer hvad der så skal ske.

Vi skal i dette afsnit se på hvad events, delegater og event handlers egentligt er for noget. Man kan skrive sine egne events og delegater – men i Windows programmeringen er det sjældent nødvendigt. De er som regel skrevet alle sammen på forhånd!!

Vi skal se et konsol program som anvender events. Vi vil starte 'bagfra' med selve applikationen som ser således ud:

```
class Baby_applikation
{
        public static void Main(string[] args)
        {
                Baby baby=new Baby();
                Baby baby1=new Baby();

                //EventHandler til event i klassen Baby:
                //Samme notation som i button.Click+=new EventHandler()!!

                Baby.For_stor+=new Baby.BabyHandler(er_blevet_for_stor);
                Baby.For_lille+=new Baby.BabyHandler(er_blevet_for_lille);

                baby.Alder=4;
                baby.fylder_aar();
                baby.fylder_aar();
                baby.fylder_aar();
                baby1.Alder=0;
                baby1.check_alder();

                //Disse skaber INGEN events!!:
                baby1.Alder=3;
                baby1.check_alder();
                baby1.fylder_aar();
```

```
                baby1.check_alder();
        }
```

Vi er simpelt hen **brugere** af en klasse **Baby**, som har 2 **events** For_stor (nemlig over 4 år!) og For_lille (nemlig under 1 år!).

Denne klasse kan vi så anvende og kalde dens metoder.

Vi ved også at denne klasse Baby har en event handler som hedder **BabyHandler.** Dette svarer fuldstændigt til, at en Button har en **EventHandler** – jvf også notationen += som tilføjer endnu en delegat/metode til eventhandleren.

+= betyder at hvis eventen For_stor forekommer, skal programmet udføre den kode som står i metoden er_blevet_for_stor – **og** i alle andre metoder som vi har tilføjet med += !! Det som vi gør med formlen += er at vi opretter en super delegat (en multi cast delegat) og tilføjer delegater til denne super delegat ('delegat = delegat + ny delegat') ! Et krav her er, at alle disse delegater skal returnere void!

I **Java** findes et system der er meget tilsvarende når man fx opretter en button og bruger formlen: button.AddActionListener(new ActionListener()) ...

Vi skriver selv metoden 'er_blevet_for_stor'. Fordi en eventhandler er konstrueret på en bestemt måde skal der **ikke** være parenteser efter metodenavnet!

På denne måde kan += anvendes mange gange og alle disse metoder vil så blive kaldt hvis event'en sker! Ingen ved i hvilken rækkefølge de forskellige metoder så ville blive kaldt!! (Dette er dog mest et teoretisk problem!). Hvis der anvendes -= fjernes en metode fra eventhandleren/delegaten. Vi kan altså dynamisk – runtime – aktivere og deaktivere bestemte eventhandlere. At aktivere en evnthandler kaldes også at registrere den. Eventhandleren er en 'listener' som kan aktiveres. Man kan også sige at eventhandleren 'abonnerer' på en event – den meddeler at den vil have besked hvis denne specielle event forekommer!

Applikationen tager alt dette for givet og **bruger** bare klassen Baby - akkurat som vi bruger klassen Button eller Form!

En applikation er 'bruger' eller **'consumer'** – ikke **'producer'**!

Delegater er 'call back' objekter idet de fungerer som et mellemled mellem fx Main() og en metode. Main kalder delegaten, som så kalder en metode, som så returnerer noget til delegaten, som så returnerer det til Main() blokken !

Delegater arbejder dynamisk – runtime – og anvender 'late binding' dvs at det først runtime afgøres hvilken metode som skal kaldes!

Hvis **applikationen** kører sker dette:

Samtidigt med at vi gør babyen ældre end 4 år affyres der en event – For_stor og tilsvarende hvis vi sætter en baby til 0 år – så affyres event'en For_lille!

Som det fremgår af kommentaren i koden sker der intet ved de sidste sætninger, fordi babyen her holder sig mellem 0 og 5 år! Så affyres ingen events!

For bedre at kunne forstå denne mekanik må vi se på klassen **Baby**:

```
public class Baby : Object {
                private int alder;

                //Disse metoder i Baby kan affyre en event:
                //Vi skal have noget i Baby som kan affyre en event!
                public void fylder_aar(){
                                alder++;
                                if(alder>4){
                                                For_stor(this,"FOR STOR: BABY NR: " +
this.GetHashCode().ToString() + " ALDER: "+alder);
                                }
                }

                public void check_alder(){

                                if(alder<1){
                                                For_lille(this,"FOR LILLE: BABY NR:
"+this.GetHashCode().ToString()+" ALDER: "+alder);
                                }
                }

                //En property som babyen har:
                public int Alder{
                                get{return alder;}
                                set{alder=value;}
                }
//Delegate - erklæres som en klasse
//En funktions pointer:

public delegate void BabyHandler(Baby afsender,string baby_msg);

//erklæring af de to mulige events i Baby:
public static event BabyHandler For_stor;
public static event BabyHandler For_lille;

}
```

Det vigtigste i klassen Baby er at den erklærer en delegate ved navn **BabyHandler**.

```
public delegate void BabyHandler(Baby afsender,string baby_msg);
```

En delegate eller på dansk: delegat kan opfattes som en **funktionspointer** der tager en metode som parameter. I en vis forstand er en delegat en metode, som kalder en anden metode. I teknisk forstand er en delegat en C# type (’klasse’) som .NET runtime opretter automatisk. Vi skal se på dette lidt senere.

Denne metode skal have et bestemt **format** men kan i øvrigt være hvad som helst! Derfor er vi også bagefter frit stillet når vi skal skrive vores eventhandler-metoder!!

I dette tilfælde skal de eventhandler-metoder vi **skal** skrive have 2 parametre nemlig en Baby og en streng. Desuden skal vores eventhandler returnere void. Men bortset fra det kan vi benævne vores eventhandler hvad som helst!

Desuden erklærer klassen to **events** nemlig For_stor og For_lille. Events skal **ALTID** erklæres på denne måde. De indeholder altså navnet på den **eventhandler** som skal administrere event’en:

```
public static event BabyHandler For_stor;
```

Desuden har vi skrevet nogle metoder som kan **udløse** events – for at **definere** hvad vi forstår ved at en event er forekommet! (Her er det ikke så indlysende som ved en button klik!).

Vi kan se at en event **afsender** netop 2 argumenter nemlig babyen selv og en tekst fra den baby som oplever event’en. Dette ligner lidt en button klik event (jvf parametrene **object** sender og **EventArgs** arg !).

```
if(alder>4){
                For_stor(this,"FOR STOR: BABY NR: " + this.GetHashCode().ToString() + " ALDER: "+alder);
}
```

Event’en ‘instantieres’ med de to parametre. En event kan minde om en Exception når vi i en metode kan have en ’throw Exception();’ ! **HVIS** vores delegat var defineret **uden** parametre ville koden i stedet være:

```
if(alder>4){
                For_stor();
}
```

Når vi skriver vores egen eventhandler er det meget vigtigt at forstå at vores eventhandler **modtager** disse to argumenter som input!! Akkurat lige som når jeg skriver en muse eventhandler **modtager** jeg musens X og Y koordinater som input eller ind-parameter til min metode!!

Vi mangler nu kun at se de **eventhandlere** som jeg (applikationen) selv skriver. De ser sådan ud i dette tilfælde men kunne selvfølgeligt se meget anderledes ud:

```
//De metoder som delegaten BabyHandler kalder
//Disse metoder skal have disse 2 parametre!!:

public  static void er_blevet_for_stor(Baby b,string event_besked){
                    Console.WriteLine("EVENT: {0}",event_besked);
                    Console.WriteLine("EVENT SENDER: {0}",b.ToString());

}
public  static void er_blevet_for_lille(Baby b,string event_besked){
                    Console.WriteLine("EVENT: {0}",event_besked);
                    Console.WriteLine("EVENT SENDER: {0}",b.ToString());

}
```

Det er disse to metoder som bliver parametre for BabyHandler og som derfor kaldes ved en event – BabyHandler **delegerer** blot arbejdet **videre**!

De to metoder ovenfor gør ikke andet end at **udskrive** de **informationer** de har fået fra event'en i de to input argumenter.

Som sagt er det nok **sjældent** nødvendigt selv at skrive eventhandlere, events osv. Men ovenstående er et forsøg på at forklare det mest basale i C#'s event system!

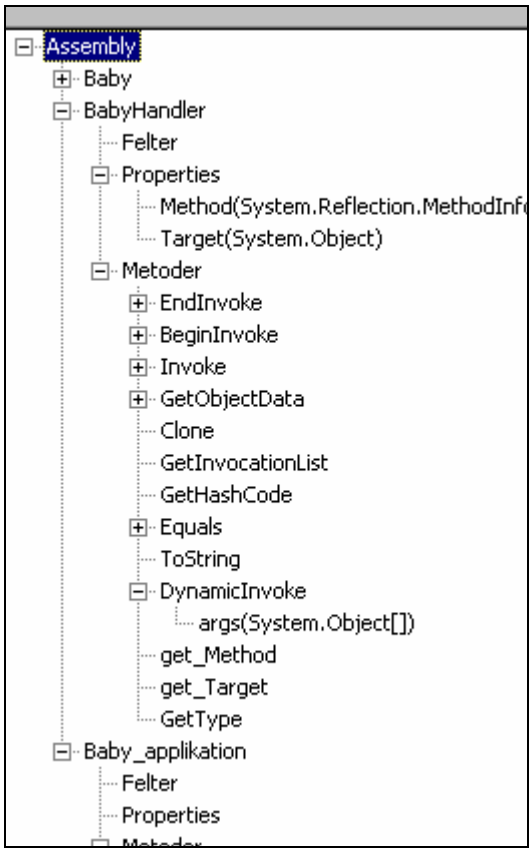
Vi vil her til sidst vende tilbage til hvad der egentligt sker når vi skriver en erklæring som:

```
public delegate void BabyHandler(Baby afsender,string baby_msg);
```

.NET runtime opretter her **automatisk** - på vore vegne - en **ny klasse** BabyHandler, som arver fra klassen **MulticastDelegate**. I teknisk forstand er BabyHandler altså en klasse og **ikke** en funktion eller funktionspointer (helt modsat systemet i C++). Events og delegater i 'type **safe'** og objektorienterede i C# (det er de normalt ikke i C++) !

Derfor kan en erklæring som: public delegate osv stå alene i en CS fil – fordi kravet er at enhver CS fil skal definere en type eller klasse! En fil der kun indeholder denne ene sætning (public delegate osv) kan altså kompileres!!

Dette kan ses når vi undersøger klassen med metoderne fra System.Reflection:



Klasserne Baby og Baby_applikation har vi selv oprettet, men **IKKE** klassen BabyHandler!! Ud fra vores erklæring af delegaten skaber systemet en klasse! Det er også derfor at vi kan anvende formlen: +=**new** Baby.BabyHandler() !

Hvis vi ser på denne klasses metoder er de alle **arvede** fra basis klassen MulticastDelegate. Metoden Invoke() er den centrale metode, som kalder eventhandler-metoden (fx er_blevet_for_stor i vores eksempel) !

Enhver delegat har to metoder BeginInvoke() og EndInvoke() som bevirker et asynkront kald – dvs en applikation med tråde. Eller med andre ord: Metode kaldet er ikke blokerende! I afsnittet senere om C# Remoting bliver der givet et eksempel på anvendelsen af asynkrone kald.

Opgaver:

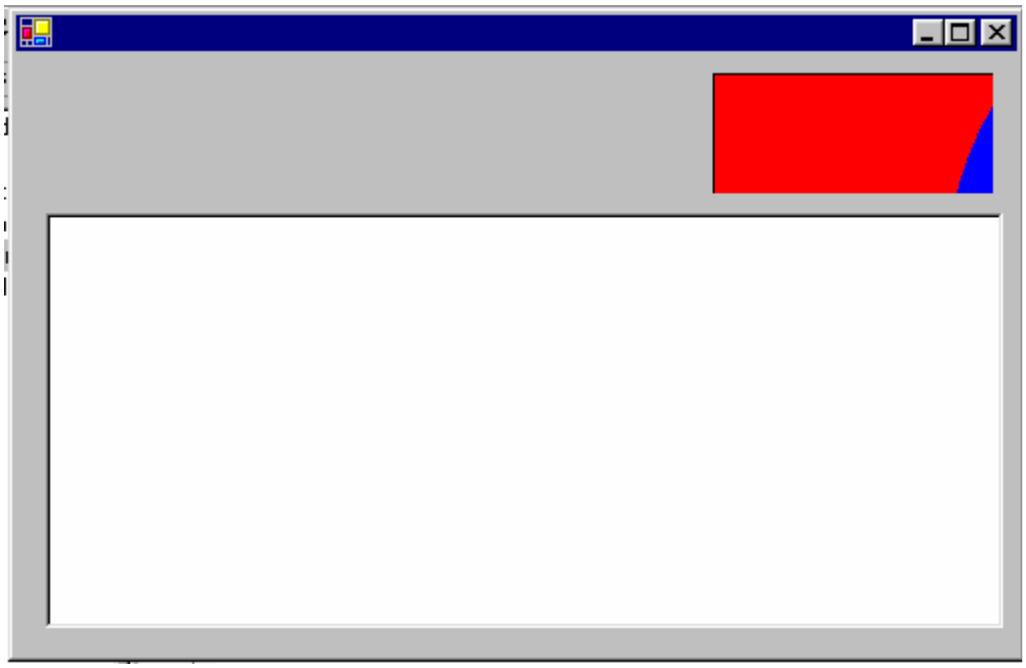
1. Skriv et hospitals program eller en skitse til det. Hospitalet modtager patienter. Der kan tænkes 2 events: PatientenErBlevetRask og Patienten Er Død. Hvordan man skrive delegater til dette program? Hvordan kan disse events realiseres? Hvilke andre events kunne tænkes?

Timer, Threads eller tråde:

En **thread** er i C# en tråd i programmet som kører **selvstændigt** og løser en bestemt delopgave i programmet. Hvis vi siger at hele programmet er en **proces**, kan vi sige at en thread er en **sub**-proces under hovedprocessen. En applikation i C# kan have mange tråde som kører samtidigt.

Mange af de opgaver som kan løses med C# threads vil kunne løses **nemmere** og **sikrere** og mere effektivt (!) med en **timer**. Vi vil derfor først se på et eksempel på en Timer og siden vende tilbage til trådene:

Følgende program viser en form med en tekstboks og en PictureBox med skiftende billeder. (NB Der skal selvfølgelig være 2 bitmaps i mappen med passende navne – ellers fås en fejl!).
Hvert 2. sekund skifter billedet (et slags banner). Når vi bruger programmet **virker** det som om programmet **gør to ting** på en gang: Administrerer tekstbehandlingen og skifter billedet! (På nedenstående billede kan man desværre ikke se at billedet skifter – men det gør det!):



Koden hertil er:

```
// Illustrerer anvendelsen af timers, threads eller traade:


using System;
using System.Windows.Forms;
using System.Drawing;
using System.Threading;

namespace Traade
{
		class MainForm : Form

		{
			private PictureBox box;

			//NB nødvendigt da der findes flere timers i C#:
			private System.Windows.Forms.Timer timer;
```

```
                    //Denne bool bruges til at huske hvilket billedet som nu vises:
                    private bool farver;
                    private RichTextBox tekst;

                    public MainForm()
                    {
                                InitializeComponents();
                    }

                    void InitializeComponents()
                    {
                                ClientSize=new Size(500,300);
                                box=new PictureBox();
                                box.Size=new Size(140,60);
                                box.Location=new Point(Width-160,10);
                                box.Image=new Bitmap("farver.bmp");
                                farver=true;
                                Controls.Add(box);

                                tekst=new RichTextBox();
                                tekst.Size=new Size(Width-30,Height-120);
                                tekst.Location=new Point(15,80);
                                tekst.Font=new Font("Bookman Old Style",20);
                                Controls.Add(tekst);

                                timer=new System.Windows.Forms.Timer();
                                timer.Tick+=new EventHandler(skift_billede);

                                //Dvs. skift for hver 2000 millisekunder dvs 2 sekunder:
                                timer.Interval=2000;
                                timer.Enabled=true;

                    }
                    private void skift_billede(object o,EventArgs a){
                                if(farver){
                                            box.Image=new Bitmap("zoom.bmp");
                                            farver=false;
                                }
                                else{
                                            box.Image=new Bitmap("farver.bmp");
                                            farver=true;
                                }
                    }

                    [STAThread]
                    public static void Main(string[] args)
                    {
                                Application.Run(new MainForm());
                    }
          }
}
```

Som det ses kan en sådan timer startes eller stoppes ved at sætte Enabled til true eller false.

Threads:

Vi vil nu vende tilbage til trådene. Et område hvor tråde anvendes og er svære at undgå er i server/klient programmering. Vi vil derfor lave en kunstig **modtager** situation. Programmet modtager 10 opkald fra **'klienter'** og starter en **ny** tråd for hver klient. (Det plejer en server at gøre).

Alle tråde gør det samme.

Et problem ved denne klient modtage situation er at klienterne typisk bruger fælles ressourcer. Her har vi valgt at alle klienter skriver til den samme fil klienter.txt. Normalt ville dette skabe kaos fordi trådene løber ved siden af hinanden og derfor ville skrive, lukke og åbne på kryds og tværs!!

Dette 'kaos' (**crash** af programmet sandsynligvis) kan undgås med en **lock**() konstruktion i C#. En lock() er en slags **semafor** eller **Monitor** som bevirker at **kun** een tråd kan være inde i proceduren ad gangen.

Hvis vi i første omgang ser på den centrale kode ser den sådan ud:

```
//Start 10 threads:
for(int i=0;i<10;i++){
klient=new Thread(new ThreadStart(modtag_klient));

//En thread kan have et navn (og en priority mv):
klient.Name="KLIENT NAVN: "+i;
//Hver gang en klient modtages startes en ny traad:
klient.Start();
}
```

En ny tråd instantieres **altid** med formlen:

```
new Thread(new ThreadStart(modtag_klient));
```

hvor modtag_klient er den metode som tråden skal udføre og hvor ThreadStart er en slags delegat, EventHandler eller funktions pointer. For at kunne skelne klienterne fra hinanden giver vi dem et kunstigt navn – ligesom vi nedenfor identificerer tråden med metoden GetHashCode() som returnerer trådens (klientens) objektnummer.

Metoden ser sådan ud:

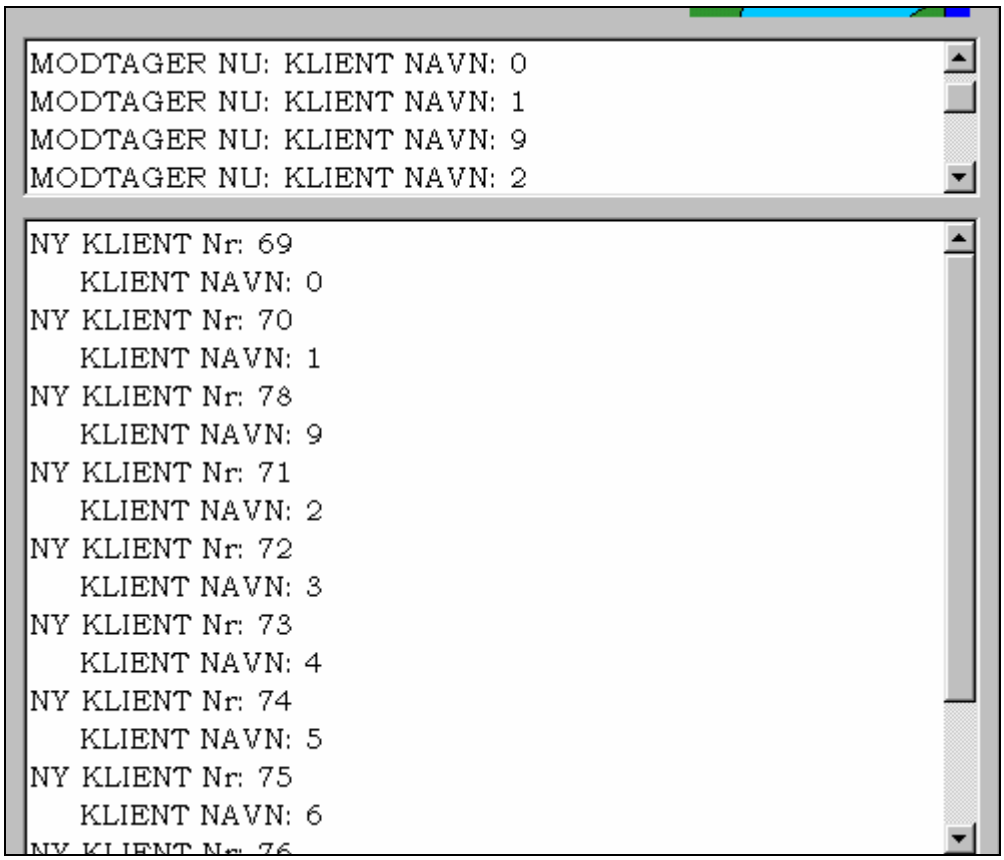
```
private void modtag_klient(){

    //lock() bevirker at klienter ikke 'rodes sammen':
    //dvs kun een thread kan være inde i metoden ad gangen:
    //Vigtigt fordi de alle skriver til filen - fælles ressource:

    lock(this){
    Random rnd=new Random();

    //Bevirker at nogle traade 'overhales'!:
    Thread.Sleep(rnd.Next(100,1000));

    writer=File.AppendText("klienter.txt");
```

```
                                        writer.WriteLine("NY KLIENT Nr: " +
Thread.CurrentThread.GetHashCode());
                                        writer.WriteLine("     "+Thread.CurrentThread.Name);

                                        writer.Close();
                                        sender.Text+="\nMODTAGER NU: "+Thread.CurrentThread.Name;
                                        reader=File.OpenText("klienter.txt");
                                        string s=reader.ReadToEnd();
                                        modtager.Text=s;
                                        reader.Close();
                                        }

                        }
```

Alle tråde skriver klientens navn og nummer ind i en **fælles ressource** klienter.txt. Disse data vises så i to tekstbokse hvoraf den øverste annoncerer hver gang en klient er modtaget og den nederste viser indholdet af filen.

Uden en lock() ville dette ikke gå særligt godt.

Som det ses bliver klienterne modtaget i en lidt tilfældig **rækkefølge**. Dette kan ses i et konkret eksempel:



Som det ses er de **først** modtagne klienter nr 0, 1, 9 og 2 – mens klienterne jo er **startet** i rækkefølgen 0, 1, 2, 3 osv! Heraf kan man se at alle 10 tråde (eller mange af dem) har forsøgt at komme ind i metoden nogenlunde **samtidigt**!

Det viste eksempel er et eksempel på **'konkurrens'** i computer sprog. Konkurrens opstår mange steder – fx når et flyselskab modtager reservationer på pladser i et fly. Det man skal sikre sig i disse situationer er, at ikke to kunder får den samme plads! Flyselskabet har jo mange samtidige tråde ud til mange kunder på en gang!

Vores fælles fil er et eksempel på det som normalt kaldes **shared memory** – en adresse i RAM eller en fil som mange tråde bruger samtidigt. (I vores eksempel kunne man også have gemt alle data i RAM i stedet for i en fil).

I stedet for lock() kan anvendes en anden konstruktion med **Monitor**.**Enter** og **Monitor**.**Exit**() – hvilket giver det samme resultat:

```
 Monitor.Enter(this);
<...metode...>
Monitor.Exit(this);
```

Hele koden til thread programmet ser således ud:

```
// Illustrerer anvendelsen af timers, threads eller traade:


using System;
using System.Windows.Forms;
using System.Drawing;
using System.Threading;
using System.IO;

namespace Traade
{
		class MainForm : Form

		{
				private PictureBox box;

				//NB nødvendigt da der findes flere timers i C#:
				private System.Windows.Forms.Timer timer;
				private bool farver;
				private RichTextBox sender,modtager;

				//Denne er mellemled mellem sender og modtager:
				private Thread klient;

				private StreamWriter writer;
				private StreamReader reader;

				public MainForm()
				{
						InitializeComponents();
				}

				void InitializeComponents()
				{
						ClientSize=new Size(500,500);
						CenterToScreen();
```

```
                box=new PictureBox();
                box.Size=new Size(140,60);
                box.Location=new Point(Width-160,10);
                box.Image=new Bitmap("farver.bmp");
                farver=true;
                Controls.Add(box);

                sender=new RichTextBox();
                sender.Size=new Size(Width-30,80);
                sender.Location=new Point(15,80);
                sender.Font=new Font("Bookman Old Style",11);
                Controls.Add(sender);

                modtager=new RichTextBox();
                modtager.Size=new Size(Width-30,320);
                modtager.Location=new Point(15,170);
                modtager.Font=new Font("Bookman Old Style",11);
                modtager.Anchor=AnchorStyles.Bottom|AnchorStyles.Top;
                Controls.Add(modtager);

                timer=new System.Windows.Forms.Timer();
                timer.Tick+=new EventHandler(skift_billede);
                timer.Interval=2000;
                timer.Enabled=true;


                //Start 10 threads:
                for(int i=0;i<10;i++){
                klient=new Thread(new ThreadStart(modtag_klient));

                //En thread kan have et navn (og en priority mv):
                klient.Name="KLIENT NAVN: "+i;
                //Hver gang en klient modtages startes en ny traad:
                klient.Start();
                }



        }
        private void modtag_klient(){

                //lock() bevirker at klienter ikke 'rodes sammen':
                //dvs kun een thread kan være inde i metoden ad gangen:
                //Vigtigt fordi de alle skriver til filen - fælles ressource:

                lock(this){
                Random rnd=new Random();


                writer=File.AppendText("klienter.txt");
                writer.WriteLine("NY KLIENT Nr:
"+Thread.CurrentThread.GetHashCode());
                writer.WriteLine("     "+Thread.CurrentThread.Name);

                writer.Close();
                sender.Text+="\nMODTAGER NU: "+Thread.CurrentThread.Name;
                reader=File.OpenText("klienter.txt");
                string s=reader.ReadToEnd();
```

```
                        modtager.Text=s;
                        reader.Close();
                        }

                }

                //EventHandler til timeren:
                private void skift_billede(object o,EventArgs a){
                        if(farver){
                                box.Image=new Bitmap("zoom.bmp");
                                farver=false;
                                Text="zoom.bmp";
                        }
                        else{
                                box.Image=new Bitmap("farver.bmp");
                                farver=true;
                                Text="farver.bmp";
                        }
                }

                [STAThread]
                public static void Main(string[] args)
                {
                        Application.Run(new MainForm());

                        //Slet filen naar programmet er slut:
                        File.Delete("klienter.txt");

                }
        }
}
```

Der er her lavet den lille ændring at formens **titellinje** også bliver opdateret med timeren. Programmet kører nu en timer og 10 tråde samtidigt!

En tråd kan afbrydes med metoden **Abort**() og suspenderes midlertidigt med metoden **Suspend**() der så modsvares af **Resume**(), der genoptager tråden. De to sidste metoder kan typisk bruges i Windows programmer til at styre visse processer. Metoden **Join**() bevirker at andre tråde må vente til den ene tråd er færdig.

Mutex:

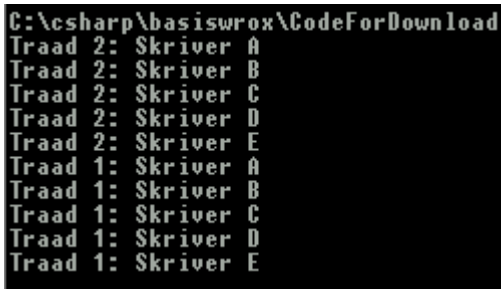
Klassen Mutex i System.Threading er opkaldt efter 'mutual exclusion' og bevirker også at flere trådes arbejde kan holdes adskilt. Følgende korte eksempel illustrerer nogle metoder i Mutex:

```
using System;
using System.Threading;

public class MutexDemo
{
  private Mutex mutex;

  public MutexDemo()
  {
```

```
  mutex = new Mutex();
 }

 public static void Main()
 {
  MutexDemo demo = new MutexDemo();

  // to traade proever begge at få fat i metoden skriv()!:
  Thread t1 = new Thread(new ThreadStart(demo.skriv));
  t1.Name = "Traad 1";
  Thread t2 = new Thread(new ThreadStart(demo.skriv));
  t2.Name = "Traad 2";

  //t2 prioriteres hoejere!:
  t1.Priority = ThreadPriority.BelowNormal;

  t1.Start();
  t2.Start();
 }

 // WaitOne() giver traaden den fulde magt - laasen - over for-løkken:
 public void skriv()
 {
  mutex.WaitOne();

  for(int i = 65; i < 70; ++i)
  {
   Console.WriteLine(Thread.CurrentThread.Name + ": Skriver {0}", (char)i);
   Thread.Sleep(5);
  }
  //aflever laasen!:
  mutex.ReleaseMutex();
 }
}
```

Metoderne WaitOne() og ReleaseMutex() svarer altså til 'lås' og 'lås op'! Her er også vist hvordan tråde kan prioriteres forskelligt.

Udskriften bliver sådan:



På tilsvarende måde kan man anvende en

ReaderWriterLock lock = new **ReaderWriterLock**();

og fx beskytte en metode således:

```
public void skriv()
  {
    lock.AcquireWriterLock(-1);
    Thread.Sleep(1000);
    //skriv et eller andet til en ressource...
    lock.ReleaseWriterLock();
  }
```

ThreadPool – anonyme tråde:

I stedet for at instantiere Thread med new Thread() kan man i C# anvende en anonym tråd fra en såkaldt ThreadPool. Der rådes hele tiden over 2 * 25 af disse tråde. De fungerer altid som baggrundstråde dvs de lukker altid når applikationen (hoved tråden) lukker.

At anvende ThreadPool er på mange måder meget nemmere end den metode vi hidtil har set på:

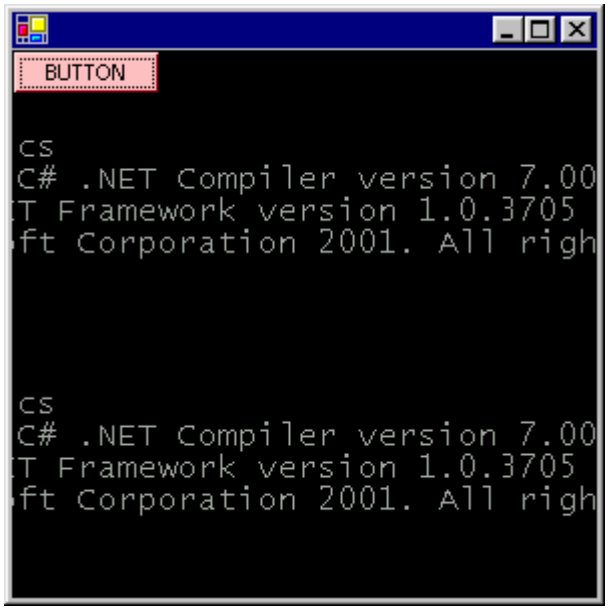
```
        //2 anonyme threads fra threadpool'en:
          ThreadPool.QueueUserWorkItem(new WaitCallback(ur),this);
          ThreadPool.QueueUserWorkItem(new WaitCallback(fil),this);

    }
    private void ur(object state){
          for(;;){
          Text=DateTime.Now.ToString();
          Thread.Sleep(1000);
          }
    }
```

Vi starter her en tråd blot ved at kalde metoden QueueUserWorkItem() i klassen ThreadPool. Denne metode tager så som parameter en delegat eller CallBack funktion der altid hedder WaitCallBack. Det viste eksempel starter så et 'ur' i titellinjen der kører evigt.

En ulempe ved ThreadPool er at en tråd altid er anonym så der er ikke kontrol over hvilken tråd der udfører hvad! En tråd kan heller ikke identificeres med et navn.
På http://csharpkursus.subnet.dk ligger et ThreadPool eksempel som indlæser en filer og ændrer font størrelse samt kører et ur i titellinjen. Kun fantasien sætter grænser!

Samme steds ligger et andet eksempel på anvendelsen af ThreadPool og klassen **ImageAnimator** i forbindelse med visning af animerede gif billeder. Dette eksempel illustrerer også hvordan man kan sætte en forms **ControlStyles** – hvilket er afgørende når der 'paintes' på formen!

Ved at sætte en forms **eller** en kontrols ControlStyles kan opnås mange mere eller mindre interessante effekter (her er formen gjort transparent - gennemsigtig, men med en pink farve, og knappen også gjort transparent!):



Opgaver:

1. Skriv et program hvor formens titellinje viser dato og klokkeslet og hvor den opdateres hvert sekund af en selvstændig tråd! Brug: DateTime.Now.ToString().
2. Skriv et program hvor en PictureBox, der fylder hele formen, viser 'levende billeder' – 10 billeder som vises i en løkke. Brug en timer eller en thread!

Skriv dine egne Windows kontroller:

Det er i C# meget enkelt at skrive sine egne 'kontroller' – sine egne knapper, tekstbokse osv.
Vi har **allerede** gjort dette med klassen Form:

class MinForm : **Form** { }

Denne linje opretter en **subklasse** af klassen Form som jeg derefter kan genbruge. En form er også en kontrol.

Oftest vil man – hvis man skal skrive sine egne kontroller – arve fra en kontrol som findes i forvejen som en C# basisklasse. Hvis man 'vil starte forfra' kan man starte med et **Panel** efter denne formel:

public class MinKontrol : Panel{ }

Et **Panel** er simpelt hen et rektangel eller 'vindue' og jeg kan så definere hvad der skal være på dette rektangel. Siden kan user-kontrollen så indsættes i en form ligesom alle andre kontroller.

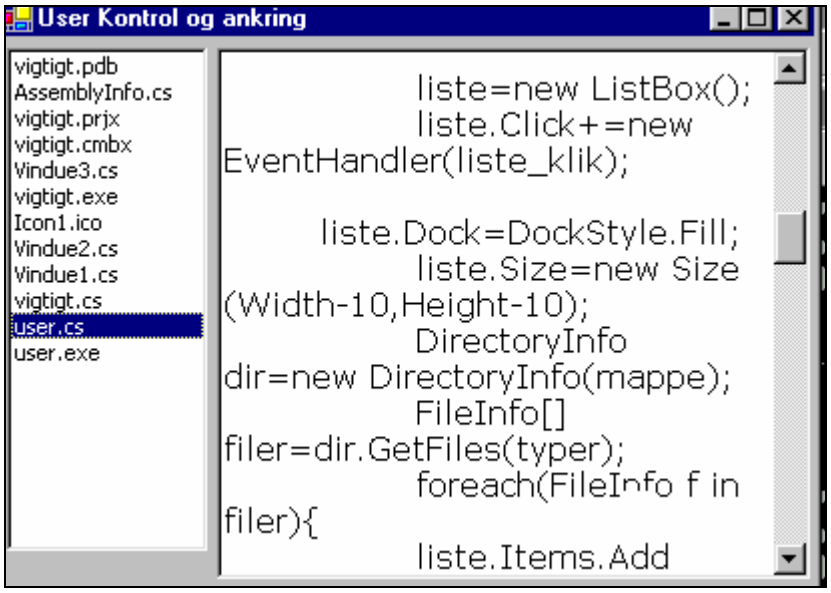
NB i det følgende eksempel er den nye kontrol gemt i samme fil som det program der anvender kontrollen. Hvis den nye kontrol skal kunne genbruges generelt skal den selvfølgeligt gemmes i sin egen DLL fil som fx **filpanel**.**dll**!

I det følgende eksempel er der skabt en ny bruger kontrol ved at arve fra Panel. Denne kontrol FilPanel viser filer med en bestemt type fra en given mappe. Filerne vises i en liste som kan sættes på en form. I det nedenstående oprettes først user kontrollen, så en form som anvender user kontrollen og så en app klasse som kører det hele. Der er anvendt Dock og Anchor fx fylder hele listen jo hele panelet:

```
//user defined kontrol:

using System;
using System.Windows.Forms;
using System.Drawing;
using System.IO;


//user defined kontrol som viser filer i en filliste hvor de kan klikkes:

public class FilPanel : Panel{

		public ListBox liste;
		private string valgt_fil;//intern variabel til den pt valgte fil

		//public property som kan bruges 'udefra' – NB kun get metode:
		public string Fil{
				get{return valgt_fil;}
		}
		public void liste_klik(object o ,EventArgs a){
				valgt_fil=liste.SelectedItem.ToString();

		}


		public FilPanel(string mappe,string typer){

				liste=new ListBox();
				liste.Click+=new EventHandler(liste_klik);
				liste.Dock=DockStyle.Fill;
				liste.Size=new Size(Width-10,Height-10);
				DirectoryInfo dir=new DirectoryInfo(mappe);
				FileInfo[] filer=dir.GetFiles(typer);
				foreach(FileInfo f in filer){
				liste.Items.Add(f.Name);
				}
				valgt_fil=liste.Items[0].ToString();
				Controls.Add(liste);
		}
}
//-------------------------------------------------------------
//En Form som anvender vores egen user kontrol:

class UForm:Form{
		private FilPanel f;
```

```
		RichTextBox boks;

		public UForm(){

		Text="User Kontrol og ankring";
		Size=new Size(400,300);
		f=new FilPanel(".","*.*");
		f.Size=new Size(100,260);
		f.Location=new Point(5,5);
		f.liste.Click+=new EventHandler (find_fil);
		f.Anchor=AnchorStyles.Top|AnchorStyles.Left|AnchorStyles.Bottom;
		Controls.Add(f);

		Panel p=new Panel();
		p.Size=new Size(290,285);
		p.Location=new Point(110,5);
		p.Anchor=AnchorStyles.Top|AnchorStyles.Right|AnchorStyles.Left|AnchorStyles.Bottom;

		boks=new RichTextBox();
		boks.Font=new Font("Verdana",12);
		boks.Size=new Size(p.Width-10,p.Height-20);
		boks.Location=new Point(0,0);
		boks.Anchor=AnchorStyles.Top|AnchorStyles.Right|AnchorStyles.Left|AnchorStyles.Bottom;
		p.Controls.Add(boks);
		Controls.Add(p);

		}

		public void find_fil(object o,EventArgs a){
				StreamReader r=File.OpenText(f.Fil);
				string s=r.ReadToEnd();
				boks.Text=s;
				r.Close();
		}


}
class applikation{
		public static void Main(){
				Application.Run(new UForm());
		}
}
```

Når programmet kører kan det fx se sådan ud – prøv også at resize formen!:



Opgaver:

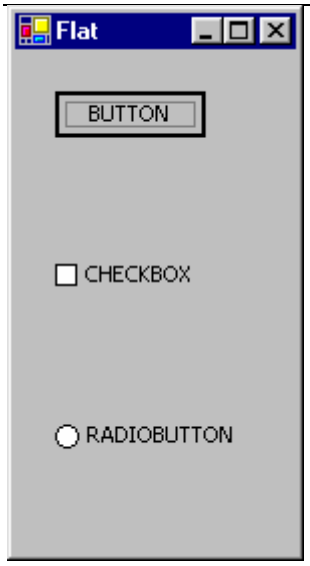
1. Skriv en ny brugerkontrol OKButton (en ny klasse) som lukker en form og som er ankret til nederste højre hjørne
2. Skriv en ny brugerkontrol HelpButton (en ny klasse) som – når den klikkes – viser en form med en hjælpetekst til et program. Kontrollen ankres til nederste venstre hjørne.
3. Skriv en ny brugerkontrol som arver fra RichTextBox (en ny klasse) og hvor du har sat en bestemt Font, ForeColor, BackColor (som du foretrækker) og som ankres til alle formens kanter.
4. Gem disse kontroller i en DLL fil: minekontroller.dll!
5. Skriv en applikation som bruger disse nye kontroller og som **refererer** minekontroller.dll.
6. I stedet for Panel kan man arve fra System.Windows.Forms.UserControl – denne klasse har flere medlemmer (properties) som ikke findes i Panel.

FlatStyle i ButtonBase:

Button, RadioButton og CheckBox arver fra ButtonBase og de har derfor alle en FlatStyle property som kan bruges – som i Java – til at sætte en **'look and feel'** på disse kontroller. Vi kan skrive et enkelt program der kan teste de 4 mulige FlatStyles (se enum FlatStyle) således:

```
// eksempler på FlatStyle i ButtonBase klassen:

using System.Windows.Forms;
using System.Drawing;
using System;

public class x: Form {

        private Button b;
        private CheckBox check;
        private RadioButton radio;
        private static int style=0;
```

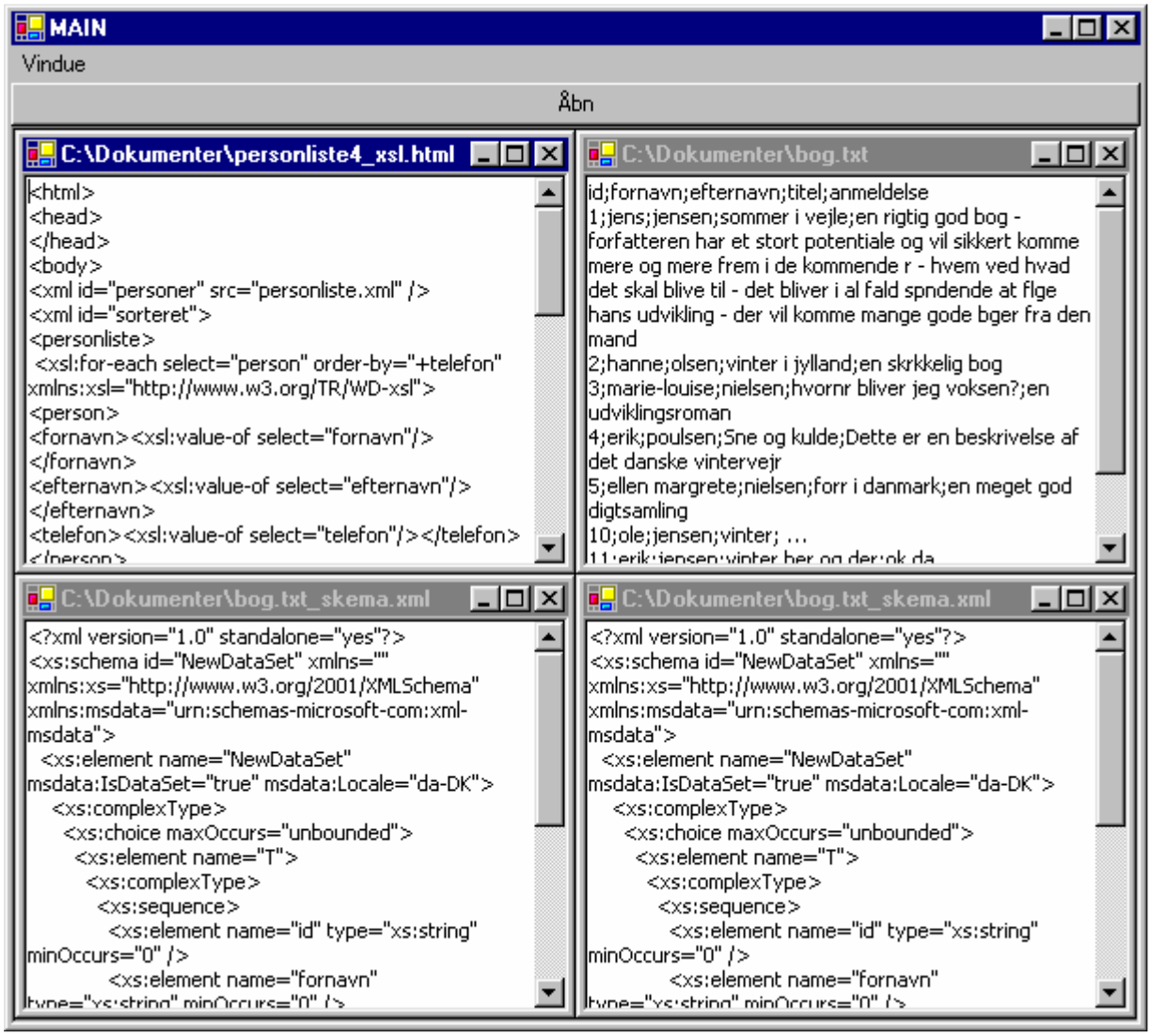
```
		public x(){
			CenterToScreen();
			ClientSize=new Size(140,250);

			b=new Button();
			b.Text="BUTTON";
			b.Location=new Point(20,20);
			Controls.Add(b);

			check=new CheckBox();
			check.Text="CHECKBOX";
			check.Location=new Point(20,100);
			Controls.Add(check);

			radio=new RadioButton();
			radio.Text="RADIOBUTTON";
			radio.Location=new Point(20,180);
			Controls.Add(radio);

			Click+=new EventHandler(flat_style);

		}
		private void flat_style(object o,EventArgs a){
			style=style%4;
			b.FlatStyle=(FlatStyle)style;
			radio.FlatStyle=(FlatStyle)style;
			check.FlatStyle=(FlatStyle)style;
			Text=b.FlatStyle.ToString();
			style++;

		}
		public static void Main(){
			Application.Run(new x());
		}
}
```



MDI Multiple Documents programmer:

På http://csharpkursus.subnet.dk ligger et simpelt eksempel på en MDI applikation, som kan åbne en række filer i forskellige child-vinduer inden for rammerne af en fælles main form. MDI systemet kendes fra mange applikationer.

I C# er det meget enkelt at skrive sådanne applikationer – idet man i princippet kun skal sætte to værdier i en form - nemlig egenskaben **IsMdiContainer** og egenskaben **MdiParent**! Her er også vist, hvordan man kan arrangere vinduer med egenskaben MdiLayout:



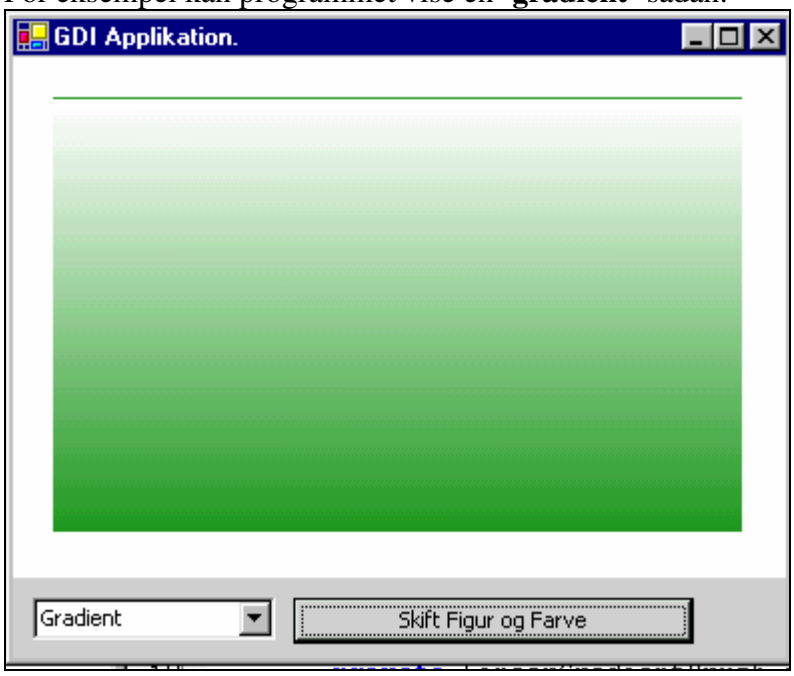
GDI programmer (Grafik programmer) i C#:

Vi vil ikke i dette kursus arbejde meget med GDI (Graphics Device Interface) programmer i C#. Disse programmer fungerer på en helt anden måde end 'almindelige' Windows programmer. I et GDI program tegner og maler man direkte på formen. Disse programmer bruger

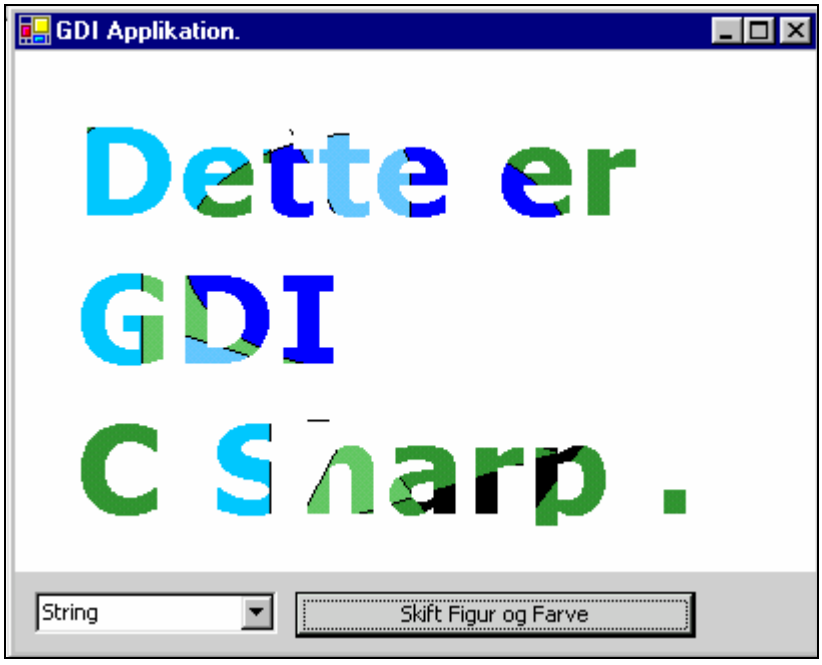
System.Drawing.dll og System.Drawing2D.dll – som indeholder en utrolig masse muligheder! Hvis man fx vil tegne sin egen nye Button fra grunden er det nødvendigt at bruge GDI.

På http://csharpkursus.subnet.dk ligger et C# program 'GDI' som illustrerer metoderne.

For eksempel kan programmet vise en **'gradient'** sådan:



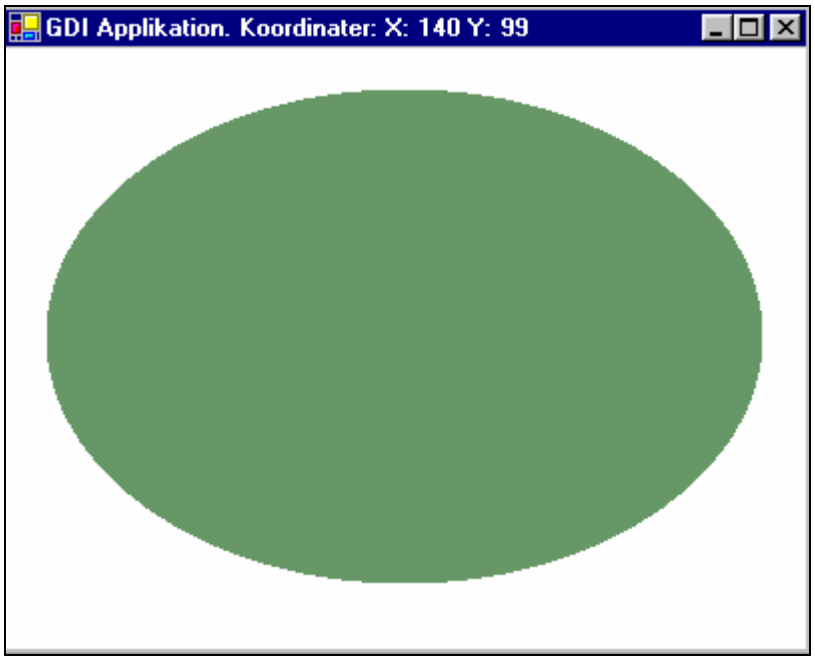
Eller en **tekst** sådan:

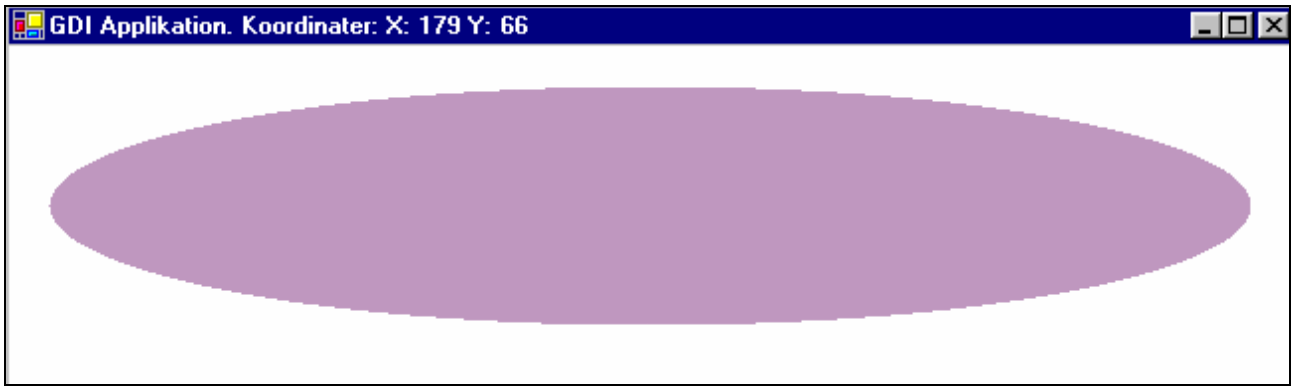


Som et meget enkelt eksempel på et GDI program kan vi se på **følgende** program. Programmet tegner en fyldt cirkel (**ellipse**) midt på formen. Når der **klikkes** på formen, skifter figuren farve. Når

musen er over formen vises dens **koordinater** (X og Y) i titel linjen. Når formen **resizes** bliver figuren også resizet!

De afgørende metoder i GDI programmer er Paint metoden (**enten** en PaintEventHandler **eller** en metode OnPaint()) og **Invalidate**() som opdaterer fladen. Desuden er muse **events** vigtige – fx i et tegne program! Der findes et virvar af forskellige **Pens** og **Brushes**. Men også billeder, **Bitmaps** og **Fonts** findes defineret i disse GDI DLL filer. Derfor er vi **oftest** nødt til at bruge en **using** Drawing; i et form program. At arbejde med klassen **Color** er selvfølgelig også en vigtig del af GDI programmer:





C# koden (som er gjort bevidst **kortfattet**) ser sådan ud:

```
//Eksempel paa GDI applikation
//Eksempel paa events:

using System;
using System.Windows.Forms;
using System.Drawing;
using System.Drawing.Drawing2D;
```

```
namespace MyFormProject
{
		class MainForm : Form
		{
				private string titel="GDI Applikation. ";
				private Color color=Color.FromArgb(100,150,100);

				public MainForm()

				{
						ClientSize=new Size(400,300);
						Text=titel;
						BackColor=Color.White;

						//Event handlers:
						Click+=new EventHandler(klik);
						Paint+=new PaintEventHandler(tegn);
						Resize+=new EventHandler(resize);
						MouseMove+=new MouseEventHandler(flyt_mus);
						MouseLeave+=new EventHandler(ingen_mus);


				}
				//Figuren opdateres ved resize!:
				private void resize(object o,EventArgs a){
						//Opdater Graphics fladen:
						Invalidate();
				}
				//Al tegning og maling foregaar normalt i en Paint metode:
				private void tegn(object o,PaintEventArgs a){

						//Graphics er den flade hvorpaa der kan males:
						Graphics g=a.Graphics;
						g.FillEllipse(new SolidBrush(color),new Rectangle(20, 20, Width-50,
Height-80));
				}

				//Klik formen - skift farve:
				private void klik(object o,EventArgs a){
						Random r1=new Random();
						int blue=r1.Next(0,256);
						Random r2=new Random();
						int red=r2.Next(0,256);
						color=Color.FromArgb(red,150,blue);

						//Kalder Paint metoden for at opdatere:
						Invalidate();
				}

				//Vis musens koordinater i Titellinjen:
				//Musen er over formen:
				private void flyt_mus(object o,MouseEventArgs a){
						Text=titel+"Koordinater: X: "+a.X+" Y: "+a.Y;

				}
				//Vis kun selve formens titel:
				//Musen er ikke over formen:
```

```
                private void ingen_mus(object o,EventArgs a){
                        Text=titel;

                }



                [STAThread]
                public static void Main(string[] args)
                {
                        Application.Run(new MainForm());
                }
        }
}
```

Tabeller og databaser – ADO .NET

ADO eller ActiveX Data Objects er en Microsoft database teknologi som er søgt videre udviklet I .NET. Men grundlæggende ligner den gamle ADO den nye ADO.NET. ADO bygger på det gamle system med **relationelle** databaser. En relationel database består kort fortalt af en eller mange tabeller. Hver **tabel** består af et **skema** nemlig et sæt af kolonner eller **felter**. En **post** i en tabel er en række med værdier. En tabel kan hænge sammen med en anden gennem en **relation**. Hver tabel bør have en **primærnøgle**. Disse begreber vil blive kort forklaret i det følgende.

Vi vil i det følgende arbejde med en simpel database Boghandel som har 3 tabeller:

Tabellen **Bog**:

bogid	forfatter	Titel
1	Jens Hansen	Snart bliver det sommer
2	Lise Jensen	Nu er det vinter

Tabellen **Kunder**:

Kundeid	Kundenavn
1	Margit Jensen
2	Kurt Olsen

Tabellen **Ordrer**:

ordreid	kundeid	bogid
1	2	1

Som det ses er alt her gjort så simpelt som muligt!

Forklaring:
De 3 tabeller hænger sammen på den måde at en ordre **skal** indeholde en bog som findes i lageret (!) og **kun** kan afgives til en kunde som 'findes'.

Hver tabel har en **primær** nøgle sådan at to poster/rækker i en tabel **aldrig** er ens **også** selv om der er 2 kunder der hedder det samme! I dette eksempel er tabellens første felt primær nøgle.
Tabellen Ordrer har 2 **fremmednøgler** som skal svare til poster i de 2 øvrige tabeller.
Relationerne kan udtrykkes sådan: En ordre **har** en kunde og en bog.

Her er et **kodeeksempel** på hvordan dette ser ud i C# - selv om der er meget kode er de meste gentagelser som kan skrives med kopier – sæt ind:

```
// filen: bogklasser.cs

        using System;
        using System.IO;
        using System.Data;
        using System.Windows.Forms;
        using System.Drawing;
        using System.Xml;



        public class Kunde{

                public DataTable tabel;

                public Kunde(){

                //opret en ny tabel med kundeid (nøgle) og kundenavn:
                tabel=new DataTable("Kunder");
                DataColumn kundeid=new DataColumn("kundeid",Type.GetType("System.Int32"));
                kundeid.ReadOnly=true;
                kundeid.AllowDBNull=false;
                kundeid.Unique=true;
                kundeid.AutoIncrement=true;
                tabel.Columns.Add(kundeid);
                DataColumn kundenavn=new DataColumn ("kundenavn", Type.GetType
                ("System.String"));
                tabel.Columns.Add(kundenavn);
                tabel.PrimaryKey=new DataColumn[]{kundeid};
                }
        }
        public class Bog{

                public DataTable tabel;

                public Bog(){

                //opret en ny tabel med bogid (nøgle), forfatter og titel:
                tabel=new DataTable("Bog");
                DataColumn bogid=new DataColumn("bogid",Type.GetType("System.Int32"));
                bogid.ReadOnly=true;
                bogid.AllowDBNull=false;
                bogid.Unique=true;
                bogid.AutoIncrement=true;
                tabel.Columns.Add(bogid);
                DataColumn forfatter=new DataColumn("forfatter",Type.GetType("System.String"));
```

```
                    tabel.Columns.Add(forfatter);
                    DataColumn titel=new DataColumn("titel",Type.GetType("System.String"));
                    tabel.Columns.Add(titel);

                    tabel.PrimaryKey=new DataColumn[]{bogid};
                    }
}
public class Ordrer{

                    public DataTable tabel;

                    public Ordrer(){

                    //opret en ny tabel med ordreid (nøgle), kundeid og bogid:
                    tabel=new DataTable("Ordrer");
                    DataColumn ordreid=new DataColumn("ordreid",Type.GetType("System.Int32"));
                    ordreid.ReadOnly=true;
                    ordreid.AllowDBNull=false;
                    ordreid.Unique=true;
                    ordreid.AutoIncrement=true;
                    tabel.Columns.Add(ordreid);

                    DataColumn kundeid=new DataColumn("kundeid",Type.GetType("System.Int32"));
                    kundeid.AllowDBNull=false;
                    tabel.Columns.Add(kundeid);
                    DataColumn bogid=new DataColumn("bogid",Type.GetType("System.Int32"));
                    bogid.AllowDBNull=false;
                    tabel.Columns.Add(bogid);

                    tabel.PrimaryKey=new DataColumn[]{ordreid};
                    }
}

public class app{
                    public static void Main(){
                    Kunde k=new Kunde();
                    Ordrer o=new Ordrer();
                    Bog b=new Bog();

                    //et DataSet er en slags database af flere tabeller samlet:
                    DataSet dataset=new DataSet("boghandel");
                    dataset.Tables.Add(k.tabel);
                    dataset.Tables.Add(b.tabel);
                    dataset.Tables.Add(o.tabel);

                    //skriv databasens dvs de 3 tabellers skema til filen boghandel.xml
                    dataset.WriteXmlSchema("boghandel.xml");
                    }
}
```

Forklaring:

I Main() instantieres de 3 klasser og der oprettes et DataSet (som kan rumme en eller mange tabeller). Da feltet 'tabel' er gjort public kan det bruges **direkte**:

Bog b=new Bog();

b.tabel;

Main() gemmer hele 'databasens' **skema** i en XML fil.
NB Hvis tabellernes data/poster skal gemmes bruges den tilsvarende metode: dataset.WriteXml("") – men foreløbigt har vore tabeller jo intet indhold!

Hvis vi åbner boghandel.xml i en browser ses dette resultat:

XML filer i fx Internet Explorer kan med + og – knapperne sammenklappes og udvides. I ovenstående eksempel er tabellerne sammenklappet – så deres felter kan ikke ses her. Constraint viser den primærnøgle vi har defineret.
XML er det foretrukne format for at gemme data i .NET. Vi skal straks se hvordan det foregår i praksis.

En tabel i C# oprettes altså sådan:

```
//opret en ny tabel med bogid (nøgle), forfatter og titel:
        tabel=new DataTable("Bog");
        DataColumn bogid=new DataColumn("bogid",Type.GetType("System.Int32"));
        bogid.ReadOnly=true;
        bogid.AllowDBNull=false;
        bogid.Unique=true;
        bogid.AutoIncrement=true;
        tabel.Columns.Add(bogid);
```

En DataTable har både et 'friendly name' ("Bog") og et id (her: tabel).
Et felt eller en kolonne oprettes med et 'friendly name' og en type. Typiske tper er int og string som vist her – men alle typer i C# kan anvendes.
En DataTable har en 'collection' Columns og feltet tilføjes denne collection.

Et felt kan have en række forskellige egenskaber:

Feltets egenskaber eller Property	**Betydning**
AllowDBNull	Skal feltet udfyldes?
AutoIncrement	Opdateres med fx 1 for hver ny post
Caption	Den titel som vises i et DataGrid jvf senere
DefaultValue	Værdi hvis ikke andet indtastes
Expression	Værdi beregnes som fx et felt beregner momsen af et andet felt
ReadOnly	Kan ikke indtastes
Unique	Alle poster skal have en forskellig værdi – bruges om primærnøgler fx

I ovenstående kode eksempel mangler der især en ting: Tabellen Ordrer er **ikke** knyttet til de 2 andre tabeller. Dette er selvfølgeligt ikke strengt nødvendigt – tabellerne fungerer som de er – men det vi ønsker er at en ordre skal referere til en **bestemt** kunde og bog som **'findes'**! Vi ønsker at Kunde og Bog skal være **'parent'** til tabellen Ordrer (**'child'**).

Vi tilføjer derfor 2 **relationer** i Main() metoden sådan:

```
                    DataRelation r1=new
DataRelation("kundeid_i_tabellen_ordrer",dataset.Tables[0].Columns[0],dataset.Tables[2].Columns[1]);
                    dataset.Relations.Add(r1);
                    DataRelation r2=new
DataRelation("bogid_i_tabellen_ordrer",dataset.Tables[1].Columns[0],dataset.Tables[2].Columns[2]);
                    dataset.Relations.Add(r2);
```

Fremmednøgler kan etableres på flere måder men her erklæres en ny DataRelation med 3 parametre:

1. parameter: relation gives et sigende **navn** (helst uden mellemrum!)
2. parameter: det felt i den tabel som er **'parent'**. I det første tilfælde (r1) er tabellen Kunde 'parent' fordi tabellen Ordrer skal referere tilbage til 'parent': I tabellen Ordrer må kun skrives en værdi som findes i tabellen Kunde! Tabellen Kunde er den første tabel vi har tilføjet vores dataset derfor har den **nummer** 0! Feltet kundeid er det første felt i tabellen altså nr 0! Kolonnerne kan omtales dels ved navn dels ved deres nummer!
3. parameter er det felt i **'child'** tabellen som skal stemme overens med 'parent'.

Vi kan også tilføje **beskrivende egenskaber og forklaringer** til vores dataset/database (her er der fuldstændigt frit slag!). Egenskaber tilføjes som name=value, altså først egenskabens navn og derefter dens værdi. Her er 3 eksempler på tilføjelse af egenskaber:

```
                    dataset.ExtendedProperties.Add("Navn","Boghandel");
                    dataset.ExtendedProperties.Add("Formål","Gemme data i Boghandel");
                    dataset.ExtendedProperties.Add("Dato",DateTime.Now.ToString());
```

Hver gang der gemmes noget i filen skrives der en opdateret dato og klokkeslæt i egenskaben Dato. Disse egenskaber kan bruges i C# programmer der arbejder med XML filen.

Disse ændringer gemmes således i XML filen:

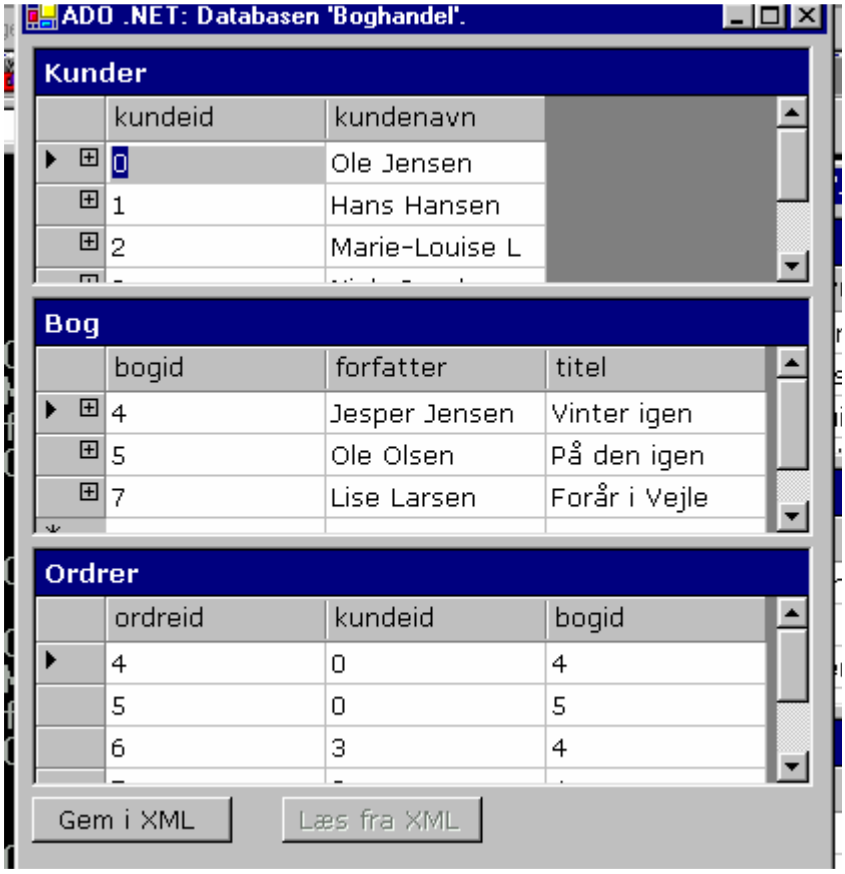
```
<?xml version="1.0" standalone="yes" ?>
<xs:schema id="boghandel" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata" xmlns:msprop="urn:schemas-microsoft-com:
  msprop">
- <xs:element name="boghandel" msdata:IsDataSet="true" msdata:Locale="da-DK"
    msprop:Navn="Boghandel" msprop:Formål="Gemme data i Boghandel" msprop:Dato="03-01-2003
    15:52:31">
  - <xs:complexType>
```

```
  - <xs:keyref name="bogid_i_tabellen_ordrer"
      refer="Bog_Constraint1">
      <xs:selector xpath=".//Ordrer" />
      <xs:field xpath="bogid" />
    </xs:keyref>
  - <xs:keyref name="kundeid_i_tabellen_ordrer" refer="Constraint1">
      <xs:selector xpath=".//Ordrer" />
      <xs:field xpath="kundeid" />
```

Et Windows program som anvender databasen boghandel:

Vi vil nu skabe en form som består af 3 kontroller (af typen **DataGrid**) og 2 knapper til at indlæse og gemme data i de 3 tabeller.
En DataGrid er en meget nyttig kontrol som har en lang række af egenskaber. Bl.a. kan en grid sættes til ReadOnly=true med det resultat at brugeren ikke kan redigere eller slette i tabellerne. Her er vi dog interesseret i kontroller hvor man kan indtaste og gemme data. Enhver DataGrid har enten en DataSource (en tabel) eller en DataBinding (et DataSet) – det sidste bruger vi her fordi vi har samlet alle tabeller i et dataset.

Når formen kører kan den se således ud **hvis** der vel at mærke allerede er gemt data i filen **boghandel.xml**:



Alle data gemmes i boghandel.xml. **Hvis** denne fil ikke findes når programmet åbner fås en lille fejl boks og derefter åbner formen normalt med lutter tomme poster men med de korrekte kolonner osv. Derefter kan indtastes data og klikkes på Gem. De 2 knapper Gem og Læs bliver gråtonede for at sikre at filen boghandel.xml ikke ødelægges.

Posterne **sorteres** i stigende/faldende rækkefølge hvis der klikkes på kolonneoverskriften.

En C# datagrid kan vise relaterede poster i andre tabeller fx hvilke ordrer har en bestemt kunde afgivet:

For at komme frem til denne form kompilerer vi programmets 3 klasser Bog, Ordrer og Kunde til en DLL fil med:

csc /t:library bogklasser.cs

Herved opstår filen bogklasser.dll. Når vi om lidt kompilerer vores form skal det ske sådan:

---

csc /r:bogklasser.dll dataform.cs

Herved kan vi bruge de tabeller/klasser som vi indtil nu har skabt.

Det næste vi gør er at oprette en DataForm som arver fra Form og flytte det meste af koden fra det tidligere Main() i bogklasser.cs over i formens constructor. Du bruger her den samme kode!

Det arbejde vi hidtil gjorde i en app klasse (i Main()) gøres nu når formen konstrueres.

På denne måde kan vores form kun bruges til disse tre tabeller. Man kunne sagtens lave en form som var mere bredt anvendelig men det vil vi ikke gøre her.

C# koden til den nye form ser således ud (for at **opdele** koden er her **indsat** linjer med kort forklaring):

```
using System;
using System.IO;
using System.Data;
using System.Windows.Forms;
using System.Drawing;
using System.Xml;



public class DataForm: Form {

        private DataGrid grid1,grid2,grid3;
        private Button xml_ind,xml_ud;
        private StatusBar status;
        private DataSet dataset;

        public DataForm(){
        Text=@"ADO .NET: Databasen 'Boghandel'.";
```

Den følgende kode er taget fra det hidtidige program med visse overspringelser:

```
        //etabler forbindelse til databasen boghandel:

        Kunde k=new Kunde();
        Ordrer o=new Ordrer();
        Bog b=new Bog();
        dataset=new DataSet("boghandel");
        dataset.Tables.Add(k.tabel);
        dataset.Tables.Add(b.tabel);
        dataset.Tables.Add(o.tabel);
        DataRelation r1=new DataRelation ("kundeid_i_tabellen_ordrer",
dataset.Tables[0].Columns[0], dataset.Tables[2].Columns[1]);
        dataset.Relations.Add(r1);
```

```
                    DataRelation r2=new
DataRelation("bogid_i_tabellen_ordrer",dataset.Tables[1].Columns[0],dataset.Tables[2].Columns[2]);
                    dataset.Relations.Add(r2);
                    try{
                    dataset.ReadXml("boghandel.xml");
                    }catch{
                              MessageBox.Show("Filen kan ikke åbnes ... Findes nok ikke!","Fejl i
fil åbning");
                    }
```

Formen og dens GUI, kontroller oprettes – formen kunne godt gøres lidt bredere i dette tilfælde! Læg mærke til at koden for de 3 datagrids blot er gentagelser:

```
                    //opret formen med dens kontroller:

                    //NB formens font arves af alle kontroller:
                    Font=new Font("Verdana",10);
                    Width=410;
                    Height=470;
                    status=new StatusBar();
                    status.Size=new Size(420,20);
                    status.Text="Alle data gemmes i filen boghandel.xml";
                    Controls.Add(status);

                    grid1=new DataGrid();
                    grid1.Width=390;
                    grid1.Height=120;
                    grid1.Location=new Point(5,5);
                    grid1.PreferredColumnWidth=110;
                    grid1.CaptionText=k.tabel.TableName;
                    Controls.Add(grid1);
                    grid2=new DataGrid();
                    grid2.Width=390;
                    grid2.Height=120;
                    grid2.Location=new Point(5,125+5);
                    grid2.PreferredColumnWidth=110;
                    grid2.CaptionText=b.tabel.TableName;
                    Controls.Add(grid2);
                    grid3=new DataGrid();
                    grid3.Width=390;
                    grid3.Height=120;
                    grid3.Location=new Point(5,125+125+5);
                    grid3.PreferredColumnWidth=110;
                    grid3.CaptionText=o.tabel.TableName;

                    Controls.Add(grid3);

                    //de 3 grids bindes til hver sin tabel og får en passende titellinje:
                    grid1.SetDataBinding(dataset,k.tabel.TableName);
                    grid2.SetDataBinding(dataset,b.tabel.TableName);
                    grid3.SetDataBinding(dataset,o.tabel.TableName);

                    grid1.CaptionText=k.tabel.TableName;
                    grid2.CaptionText=b.tabel.TableName;
                    grid3.CaptionText=o.tabel.TableName;

                    xml_ud=new Button();
```

```
                xml_ud.Text="Gem i XML";
                xml_ud.Width=100;
                xml_ud.Location=new Point(5,Height-90);
                xml_ud.Click+=new EventHandler(xml_ud_klik);
                xml_ud.Enabled=true;
                Controls.Add(xml_ud);

                xml_ind=new Button();
                xml_ind.Text="Læs fra XML";
                xml_ind.Width=100;
                xml_ind.Location=new Point(130,Height-90);
                xml_ind.Click+=new EventHandler(xml_ind_klik);
                xml_ind.Enabled=false;
                Controls.Add(xml_ind);


                CenterToScreen();

        }
```

Der skrives 2 event handlers som bestemmer hvad der skal ske når der klikkes på de 2 knapper. Når der klikkes på Læs knappen sker der en dataset.Clear() dvs. at det dataset som lige nu ligger i RAM og vises i formen bliver slettet og et nyt dataset indlæses fra filen boghandel.xml. Hvis ikke det 'gamle' dataset var blevet slettet ville datasettet fra filen blive sammenblandet med det forhånden værende dataset og der ville opstå poster med de samme værdier!
NB ved sådanne event handlers er det almindeligt at sætte en tekst i statuslinjen som vist her.

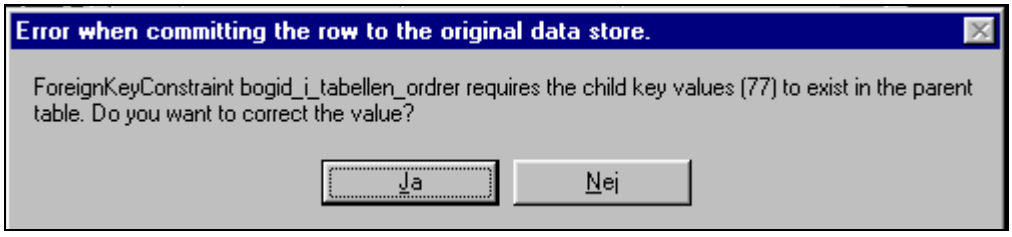
```
        public void xml_ind_klik(object o,EventArgs a){

                dataset.Clear();
                dataset.ReadXml("boghandel.xml");
                xml_ind.Enabled=false;
                status.Text=@"Alle poster er indlæst fra filen 'boghandel.xml'.";
                xml_ud.Enabled=true;

        }

        public void xml_ud_klik(object o,EventArgs a){

                dataset.WriteXml("boghandel.xml");
                status.Text=@"Alle poster er gemt i filen 'boghandel.xml'.";
                xml_ind.Enabled=true;
                xml_ud.Enabled=false;

        }


}
```

Den nye applikations klasse er nu blevet prisværdigt kort!

```
public class app{
        public static void Main(){
        Application.Run(new DataForm());
        }
}
```

Formen kan nu bruges til at vise og oprette poster i tabellerne, vise hvilke relationer som findes fx hvis der klikkes på en bestemt kunde hvilke ordrer har denne kunne så i tabellen ordrer? Der kan ikke indtastes værdier der hvor vi har gjort feltet ReadOnly (i primærnøglerne).
I teorien kunne nye poster direkte indskrives manuelt i xml filen. Prøv det! De nye poster vil så vises i formen når den åbnes.

Det er meget væsentligt at formen automatisk **kontrollerer** (validerer) om de værdier vi indtaster er gyldige – fx fås følgende boks hvis der indtastes en bog som ikke eksisterer:



Det som sker når vi klikker på Gem er en såkaldt **'Commit'** tilbage til den oprindelige database – her gemt i XML filen. Når et dataset i RAM skal committes sker der en validering af de nye poster og værdier! De poster vi sidder og indtaster i formen findes jo indtil videre kun i RAM lageret – ikke gemt andre steder!
Hvis jeg har indtastet en post med 'ugyldige' værdier bliver de **aldrig** gemt i databasen – **uanset** om jeg svarer Ja eller Nej i ovenstående boks!

Opgaver:

1. opret en ny tabel i en ny klasse Kursus med disse kolonner/felter: id (autoincrement, nøgle), kursusbeskrivelse (tekst), underviser (tekst), startdato (tekst), slutdato (tekst).
2. Opret en ny tabel: Underviser med disse felter: id (nøgle), fornavn, efternavn, telefon, mobil, email.
3. Opret en relation mellem underviser i tabellen Kursus og id i tabellen Underviser.
4. Kreer en form med en datagrid som viser tabellen kursus.
5. Opret en form med en datagrid som viser tabellen underviser.
6. Opret en form med een datagrid men hvor man med to knapper kan vælge hvilken tabel der skal vises.

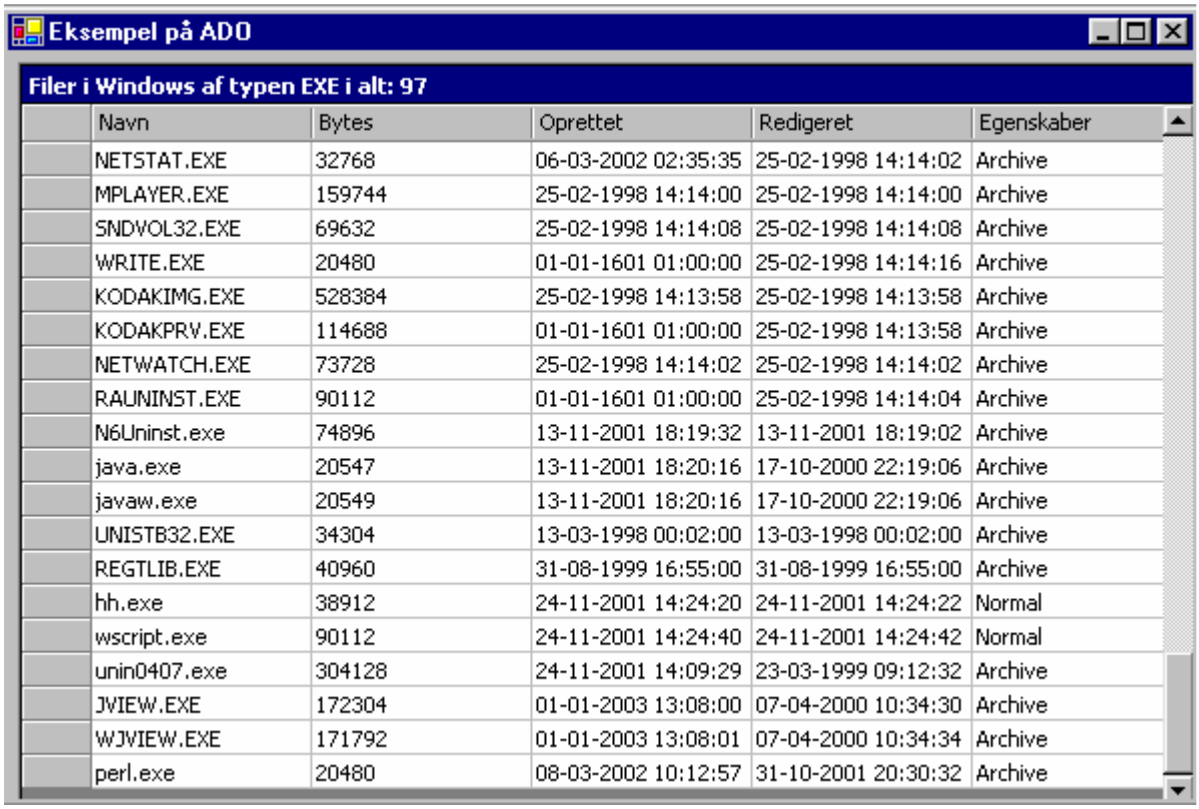
Tabeller kan bruges til meget:

Det er en god ide at vise data i tabel formatet i Windows programmer hvor det er muligt. Tabeller og datagrids kan vise data **overskueligt** og let forståeligt.

Følgende eksempel illustrerer hvordan en tabel kan oprettes ved at sætte posterne direkte – på en anderledes måde end i eksemplet med databasen boghandel.

Eksemplet tager udgangspunkt i og gentager koden fra et **tidligere** program (se afsnittet om **foreach** strukturen tidligere i kurset): Ved hjælp af klassen **DirectoryInfo** kan oprettes en tabel af filer.

Når **formen** kører vises dette:



Navn	Bytes	Oprettet	Redigeret	Egenskaber
NETSTAT.EXE	32768	06-03-2002 02:35:35	25-02-1998 14:14:02	Archive
MPLAYER.EXE	159744	25-02-1998 14:14:00	25-02-1998 14:14:00	Archive
SNDVOL32.EXE	69632	25-02-1998 14:14:08	25-02-1998 14:14:08	Archive
WRITE.EXE	20480	01-01-1601 01:00:00	25-02-1998 14:14:16	Archive
KODAKIMG.EXE	528384	25-02-1998 14:13:58	25-02-1998 14:13:58	Archive
KODAKPRV.EXE	114688	01-01-1601 01:00:00	25-02-1998 14:13:58	Archive
NETWATCH.EXE	73728	25-02-1998 14:14:02	25-02-1998 14:14:02	Archive
RAUNINST.EXE	90112	01-01-1601 01:00:00	25-02-1998 14:14:04	Archive
N6Uninst.exe	74896	13-11-2001 18:19:32	13-11-2001 18:19:02	Archive
java.exe	20547	13-11-2001 18:20:16	17-10-2000 22:19:06	Archive
javaw.exe	20549	13-11-2001 18:20:16	17-10-2000 22:19:06	Archive
UNISTB32.EXE	34304	13-03-1998 00:02:00	13-03-1998 00:02:00	Archive
REGTLIB.EXE	40960	31-08-1999 16:55:00	31-08-1999 16:55:00	Archive
hh.exe	38912	24-11-2001 14:24:20	24-11-2001 14:24:22	Normal
wscript.exe	90112	24-11-2001 14:24:40	24-11-2001 14:24:42	Normal
unin0407.exe	304128	24-11-2001 14:09:29	23-03-1999 09:12:32	Archive
JVIEW.EXE	172304	01-01-2003 13:08:00	07-04-2000 10:34:30	Archive
WJVIEW.EXE	171792	01-01-2003 13:08:01	07-04-2000 10:34:34	Archive
perl.exe	20480	08-03-2002 10:12:57	31-10-2001 20:30:32	Archive

C# koden ser således ud:

```
//eksempel at oprette en 'automatisk' tabel af alle exe filer i Windows:
//eksempel på visning af data i DataGrid:

using System;
using System.Data;
using System.Drawing;
using System.Windows.Forms;
using System.IO;//i dette namespace fides alle fil klasser

public class FilForm : Form {

public FilForm(){
CenterToScreen();
Size=new Size(600,400);
Text="Eksempel på ADO";
DataTable tabel=new DataTable();
string[] kol={"Navn","Bytes","Oprettet","Redigeret","Egenskaber"};
for(int i=0;i<5;i++){
tabel.Columns.Add(new DataColumn(kol[i],typeof(string)));
}
DirectoryInfo dir=new DirectoryInfo("C:\\windows");
if(dir.Exists){
FileInfo[] filer=dir.GetFiles("*.exe");
foreach(FileInfo f in filer){
DataRow r=tabel.NewRow();
r[0]=f.Name;
r[1]=f.Length;
r[2]=f.CreationTime;
```

```
r[3]=f.LastWriteTime;
r[4]=f.Attributes.ToString();
tabel.Rows.Add(r);
}
}

DataGrid grid1=new DataGrid();
grid1.Width=590;
grid1.Height=390;
grid1.Location=new Point(5,5);
grid1.PreferredColumnWidth=110;
grid1.ReadOnly=true;
grid1.CaptionText="Filer i Windows af typen EXE i alt: "+tabel.Rows.Count;
grid1.DataSource=tabel;
Controls.Add(grid1);
}
}
class app{
public static void Main(){
Application.Run(new FilForm());
}
}
```

Her oprettes en række poster I tabellen med en **foreach** løkke. En ny post oprettes med formlen:

```
DataRow r=tabel.NewRow();
```

og postens værdier indsættes med [] notationen sådan at [0] er første kolonne/felt – til sidst tilføjes rækken til tabellens collection af DataRows:

```
FileInfo[] filer=dir.GetFiles("*.exe");
foreach(FileInfo f in filer){
DataRow r=tabel.NewRow();
r[0]=f.Name;
r[1]=f.Length;
r[2]=f.CreationTime;
r[3]=f.LastWriteTime;
r[4]=f.Attributes.ToString();
tabel.Rows.Add(r);
}
```

Læg også mærke til at tabellen kender antallet af rækker i tabellen.

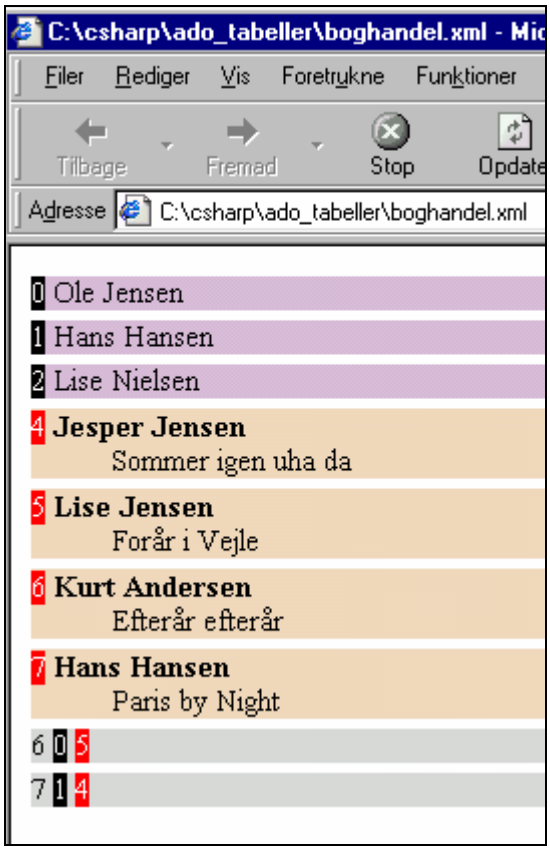
Opgaver:

1. Opret på lignende måde en form som viser den store tabel
2. Modificer formen med filer så brugeren kan **vælge** hvilke filtyper der skal vises og fra hvilken mappe.

Visning/konvertering af XML filer:

En database gemmes typisk i XML **formatet** i .NET. XML formatet er et rent tekstformat og egner sig derfor til at blive transporteret over netværk – sammen med at formatet har mange andre fordele. Disse XML filer er dog ikke så 'brugervenlige' umiddelbart!

Ved hjælp af 'style sheets' eller 'typografi ark til hyper text' (**Cascading Style Sheets**, CSS) kan XML filer dog vises på en mere 'venlig' måde:



Her er brugt databasen boghandel som tidligere blev omtalt. Starten af filen boghandel.xml er nu tilføjet en **tag** <> med en henvisning til et style sheet sådan:

```
<?xml version="1.0" standalone="yes"?>

<?xml-stylesheet type="text/css" href="boghandel.css"?>

<boghandel>
  <Kunder>
    <kundeid>0</kundeid>
    <kundenavn>Ole Jensen</kundenavn>
  </Kunder>
  <Kunder>
    <kundeid>1</kundeid>
    <kundenavn>Hans Hansen</kundenavn>
  </Kunder>
```

Alle XML filer er **hierarkisk** opbyggede – her er det overordnede **element** boghandel som under sig har 3 under elementer Kunder, Bog og Ordrer ... som så igen har under elementer under sig. (Et **'element'** er i XML en start og slut tag: <>...</>).

Typografiarket **boghandel.css** starter derfor med at definere hvordan siden **overordnet** skal vises, **derefter** hvordan de 3 tabeller skal vises osv. I virkeligheden skal der ikke så meget til før at XML filen bliver 'brugervenlig':

```
boghandel{
font-size:11pt;
}
Kunder{
background-color:rgb(220,190,220);
display:block;
margin-bottom:5px;
}

Ordrer{
background-color:rgb(220,220,220);
display:block;
margin-bottom:5px;
}
kundeid{
font-style:bold;
background-color:black;
color:white;
}
bogid{
font-style:bold;
background-color:red;
color:white;

}
Bog{
background-color:rgb(240,220,190);

display:block;
margin-bottom:5px;
}
titel{
text-indent:40px;
display:block;
}
forfatter{
font-weight:bold;
}
```

Som det ses består CSS filen udelukkende af koder (elementer og formater).
De fleste af koderne er ret indlysende. Formatet for et element defineres altså ved at skrive dets **navn** efterfulgt af en **blok** { } med definitioner (blok defineret som i C#).

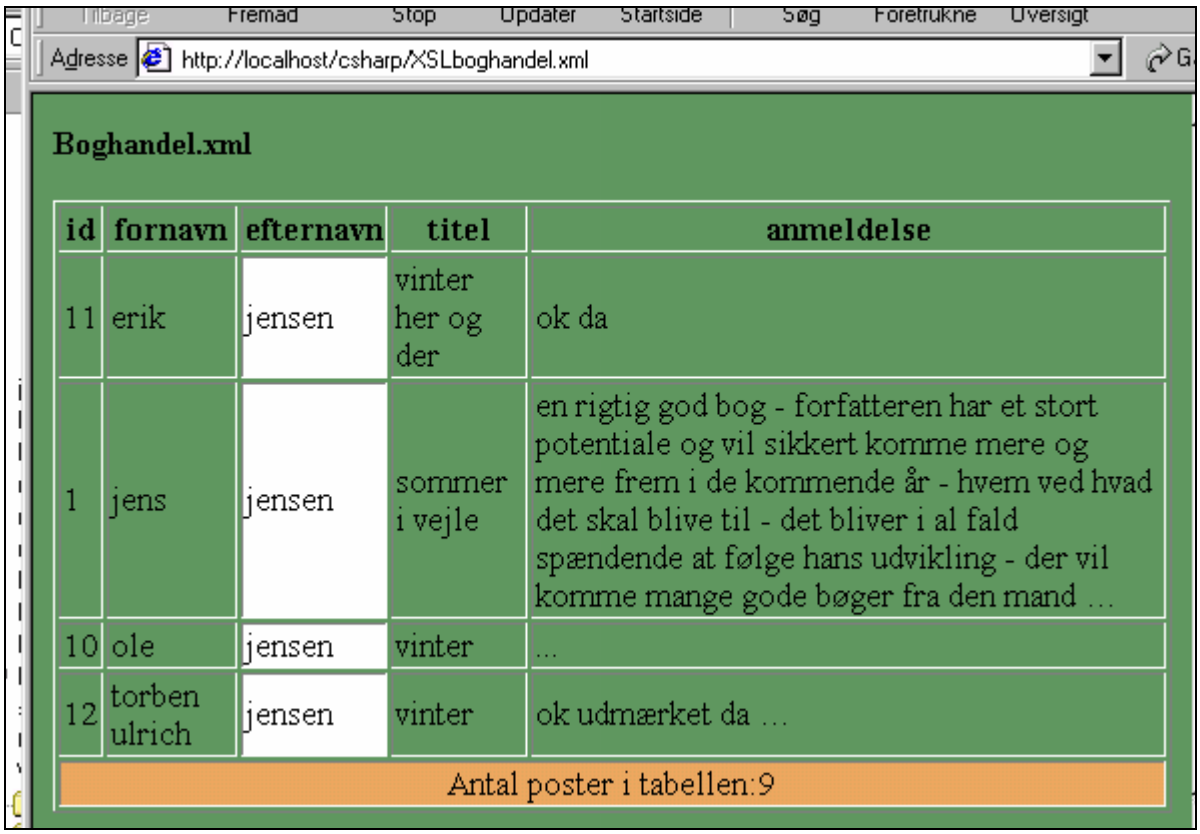
display:block betyder at der skal skiftes linje.
display: inline at elementet skal stå på samme linje som det forrige.
der kan sættes en kant om feltet/elementet med border: 5px double solid (f.eks).

Vi vil ikke her komme nærmere ind på CSS stilen men ved at forsøge sig frem og hente lidt på nettet er det ikke så svært.

Det fordelagtige ved XML er at det er systematisk, struktureret og anvendeligt i mange sammenhænge. Et XML dokument har lige som en tabel et skema som kan bruges til at validere dokumentet og bevise at det er 'vel formet' og korrekt. XML filer kan bruges til søgning og sortering mv.
Den nu producerede side med cascading style sheets kan direkte anvendes som web side. Med rigtige XML teknologier kan man komme meget længere med fx databaser i XML format.

En anden teknologi som anvendes sammen med XML er XSL som er XML stylesheets. Fordelen ved disse er at de er langt mere kraftfulde end CSS: De kan sortere, søge, tælle, beregne osv. Ved hjælp af XSL style sheets kan man få noget der ligner en dynamisk website.

På adressen http://csharpkursus.subnet.dk ligger 2 eksempler på XSL nemlig XSLklienter og XSLboghandel som viser nogle af XSL's muligheder. Hvis XSL kombineres med et rigtigt programmeringssprog som C# kan man virkeligt bruge XML og XSL til at kommunikere med en database gemt i XML formatet.



Her er **søgt** på alle forfattere som hedder 'jensen' og posterne er **sorteret** først efter efternavn, så efter fornavn! Læg også mærke til at man med XSL kan finde det samlede **antal** poster i tabellen.

På adressen http://csharpkursus.subnet.dk ligger også 2 eksempler på hvordan på en simpel måde kan skrive til og læse fra XML filer.

Opgaver:

1. Opret en lang række poster i databasen boghandel, gem dem i boghandel.xml og skriv nogle style sheets der kan præsentere data på en rimelig måde.
2. Gå på nettet og skaf dig flere informationer om XML (som de fleste mener har en stor fremtid for sig): fx på www.w3.org eller www.xml.com eller www.webdeveloper.com/xml.
3. da du installerede .NET fik du flere config filer (i XML) med i købet fx **security**.config og **machine**.config som er globale konfigurations filer. Find dem og formater dem med et style sheet. (Efter princippet: Start fra top niveau og formatter **kun** så meget at filen ser rimelig ud!).

At koble op til en Access database:

NB For at få de følgende (og muligvis andre) eksempler til at virke er det muligt at du skal downloade MDAC (seneste version) fra http://msdn.microsoft.com!

DataReader Eksempel:

Det følgende **forudsætter** at du har oprettet en Access database boghandel.mdb i c:\dokumenter. Databasen har kun en tabel: **bog** som har 4 **felter**: id (nøgle, autonummer), fornavn (tekst), efternavn (tekst) og anmeldelse (notat, en tekst 'blob' i database-sprog).

Tabellen bog:

Når C# programmet **datareader**.exe kører bliver resultatet dette:

Det mest afgørende er at der oprettes en forbindelse (en OleDbConnection) til Access databasen. Der kan gå rigtigt mange ting galt når man prøver at få forbindelse til databaser, men det efterfølgende eksempel er testet og skulle virke korrekt. (Du skal være meget omhyggelig med de værdier som sættes til **Provider – driveren -** og **Data Source - datakilden**!):

```
// datareader.cs

//velegnet til blot at vise indhold i database som ikke kan opdateres
//ikke saa fleksibel
//viser data fra tabel i tekst bokse:

//NB forbindelsen til databasen lukkes ikke her eksplicit - af nemheds grunde!


		using System;
		using System.IO;
		using System.Data;
		using System.Data.OleDb;//fx Access
		using System.Windows.Forms;
		using System.Drawing;
		using System.Xml;

		public class DataForm : Form

		{
				private OleDbConnection cn;
				private TextBox id,fornavn,efternavn,titel,anmeldelse;
				private Button frem;
				private		OleDbDataReader reader;
```

```
public DataForm(){

        Text="DataReader: Boghandel.mdb";
        Size=new Size(300,370);
        Font=new Font("Verdana",10);

        //tilføj kontroller:
        id=new TextBox();
        id.Width=200;
        id.Location=new Point(40,20);
        Controls.Add(id);
        fornavn=new TextBox();
        fornavn.Width=200;
        fornavn.Location=new Point(40,60);
        Controls.Add(fornavn);
        efternavn=new TextBox();
        efternavn.Width=200;
        efternavn.Location=new Point(40,100);
        Controls.Add(efternavn);
        titel=new TextBox();
        titel.Width=200;
        titel.Location=new Point(40,140);
        Controls.Add(titel);
        anmeldelse=new TextBox();
        anmeldelse.Multiline=true;
        anmeldelse.Size=new Size(200,100);
        anmeldelse.Location=new Point(40,180);
        anmeldelse.ScrollBars=ScrollBars.Both;
        Controls.Add(anmeldelse);
        frem=new Button();
        frem.Text="Næste";
        frem.Click+=new EventHandler(videre);
        frem.Location=new Point(40,300);
        Controls.Add(frem);

        //DB connection: Kald op til Access databasen:
        cn=new OleDbConnection();
        cn.ConnectionString="provider=Microsoft.JET.OLEDB.4.0;data
source=c:\\dokumenter\\boghandel.mdb";
        cn.Open();

        //SQL kommando: vælg alle kolonner i tabellen bog

        OleDbCommand kommando=new OleDbCommand("select * from
bog",cn);
        reader=kommando.ExecuteReader();
}

//event handler som viser næste post:
private void videre(object sender,EventArgs a){

        //er der flere poster: Read() returnerer true eller false:
        if(reader.Read()){
                id.Text=reader["id"].ToString();
                fornavn.Text=reader["fornavn"].ToString();
                efternavn.Text=reader["efternavn"].ToString();
                titel.Text=reader["titel"].ToString();
```

```
                                        anmeldelse.Text=reader["anmeldelse"].ToString();
                                }
                                else Text="Ikke flere poster i tabellen bog i boghandel.mdb!";


                        }


                        public static void Main(string[] args)
                        {
                                Application.Run(new DataForm());
                        }
                }
```

Den helt afgørende kode er altså:

```
        //DB connection: Kald op til Access databasen:
        cn=new OleDbConnection();
        cn.ConnectionString="provider=Microsoft.JET.OLEDB.4.0;data source =
c:\\dokumenter\\boghandel.mdb";
        cn.Open();

        //SQL kommando: vælg alle kolonner i tabellen bog

        OleDbCommand kommando=new OleDbCommand("select * from bog",cn);

        reader=kommando.ExecuteReader();
```

OleDbConnection er defineret i System.Data.OleDb og skal indeholde en streng som definerer **mindst** en driver og en database. Hvis databasen ligger i samme mappe som programmet kan man blot anføre 'boghandel.mdb'.

En **OleDbCommand** indeholder en SQL sætning udført på en forbindelse.

Nogle **eksempler** på SQL sætninger:

SQL	Betydning
select * from bog	find alle rækker alle kolonner/felter
select titel from bog	vis kun feltet titel
select fornavn, efternavn from bog	viser de 2 kolonner
select id from bog where id < 3	viser kun de første rækker
select * from bog where fornavn='jens'	find de rækker hvor der står 'jens' i feltet

---

Opgaver:

1. Omskriv ovenstående program så det anvender de forskellige slags SQL sætninger
2. Indfør en tekstboks på formen hvor man kan skrive en SQL sætning og en knap Vis så at formen viser resultatet af den SQL brugeren har skrevet
3. Skriv en ny Access database med en tabel efter eget valg og skriv en ny form som kan vise tabellen.
4. Opret en knap 'Forfra' der bevirker at posterne vises forfra igen! (NB en datareader kan ikke 'gå tilbage' i posterne!).

Vis database i DataGrid med en DataAdapter og et DataSet:

Den anden metode vi skal se på viser databasen/tabellen i en C# **DataGrid** – en meget mere **fleksibel** løsning, som gør det muligt at skrive nye poster, redigere i data og opdatere tilbage til Access databasen!

Når **programmet** kører bliver resultatet dette:

Der kan nu redigeres og opdateres og skrives nye poster som gemmes i den originale Access database.

Koden er meget lignende det tidligere eksempel men boghandel.mdb indlæses nu i et **dataset** ved hjælp af en data **adapter** som har metoden Fill(). Den viste data grid viser så automatisk de rigtige kolonne overskrifter og poster uden at vi skal skrive kontroller som i det tidligere eksempel med data readeren. En DataGrid har en metode: **SetDataBinding**() som binder den til et dataset og en property **DataSource** som binder den til en tabel.

NB en **ulempe** er – som det ses – at lange tekstfelter ikke får megen plads i en datagrid – derfor vil det ofte være nødvendigt at oprette en tekst boks til disse!
Programmet gemmer også data i en **XML** fil – hvad der selvfølgeligt ikke er strengt nødvendigt i dette tilfælde.
De metoder som findes i data adapeteren gør det **overraskende** nemt at committe tilbage til (opdatere) den oprindelige database! Et godt eksempel på OOP 'indkapsling'!

Koden ser sådan ud:

// database.cs

```
using System;
using System.IO;
using System.Data;
using System.Data.OleDb;//fx Access
using System.Windows.Forms;
using System.Drawing;
using System.Xml;

public class DataForm:Form

{
		private OleDbConnection cn;
		private OleDbCommandBuilder builder;
		private OleDbDataAdapter adapter;
		private DataSet dataset;
		private DataGrid grid=new DataGrid();
		private Button opdater;


		private void opdater_database(object o,EventArgs a){
			try{
				grid.Refresh();

				//opdater tilbage til originalen ('commit'):
				adapter.Update(dataset,"bog");

			}
			catch{
				MessageBox.Show("Fejl","Fejl i opdateringen af
databasen!");
			}

		}

		public DataForm(){

			Text="Data fra boghandel.mdb";
			Size=new Size(500,300);
			Font=new Font("Verdana",10);
			grid.Size=new Size(490,230);
			grid.Location=new Point(5,5);

grid.Anchor=AnchorStyles.Left|AnchorStyles.Top|AnchorStyles.Bottom|AnchorStyles.Right;
			grid.BackColor=Color.Silver;
			Controls.Add(grid);
			opdater=new Button();
			opdater.Text="Opdater";
			opdater.Size=new Size(150,24);
			opdater.Location=new Point(240,245);
			opdater.Anchor=AnchorStyles.Bottom|AnchorStyles.Right;
			opdater.Click+=new EventHandler(opdater_database);
			Controls.Add(opdater);

			//DB connection:
			cn=new OleDbConnection();
			cn.ConnectionString="provider=Microsoft.JET.OLEDB.4.0;data
source=c:\\dokumenter\\boghandel.mdb";
			cn.Open();
```

```
                adapter=new OleDbDataAdapter("select * from bog",cn);
                dataset=new DataSet();
                adapter.Fill(dataset,"bog");
                //tabellen oprettes først her

                grid.SetDataBinding(dataset,"bog");
                grid.CaptionText="Boghandel.mdb";

                //NB hver adapter sin command builder!
                builder=new OleDbCommandBuilder(adapter);
                dataset.WriteXml("boghandel.xml");

                cn.Close();

        }


        public static void Main(string[] args)
        {
                Application.Run(new DataForm());
        }
}
```

En helt afgørende forskel på en datareader og det nuværende eksempel er at en datareader bliver ved med at holde forbindelsen til databasen åbnet mens et dataset fyldes på en gang hvorefter forbindelsen lukkes!
Et dataset er et 'in memory' dataset. Dvs. jeg kan redigere frit i dette dataset (i datagrid'en) lige indtil jeg 'committer' (opdaterer) tilbage – i det øjeblik **kontrolleres** det at tabellens nøgler, fremmednøgler osv bliver overholdt!!

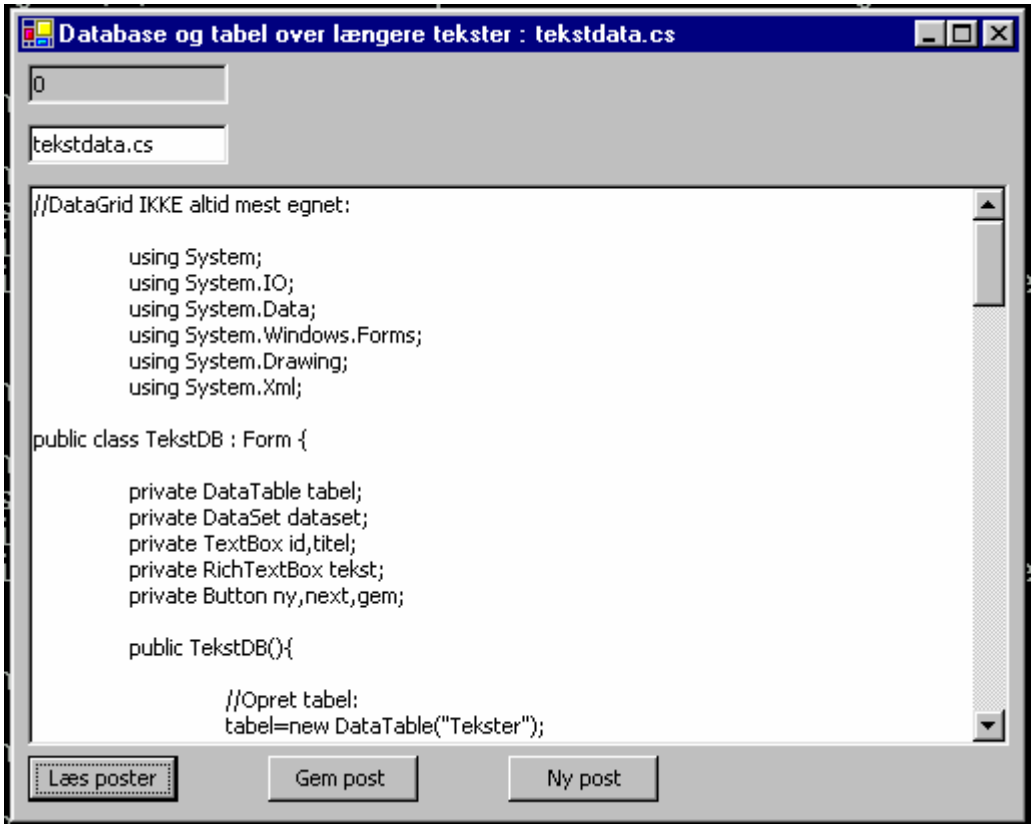
Prøv at indtaste flere nye poster og redigere i de nuværende og kontroller om de bliver gem i boghandel.mdb!

Prøv at indtaste **ugyldige** data – fx to bøger med samme id - og du vil få at vide at databasen ikke kan opdateres pga. de og de poster som ikke overholder kravene!

(I dette tilfældige har vi dog lavet vores egen primitive **exception** handling med try..catch – men prøv at fjerne den!).

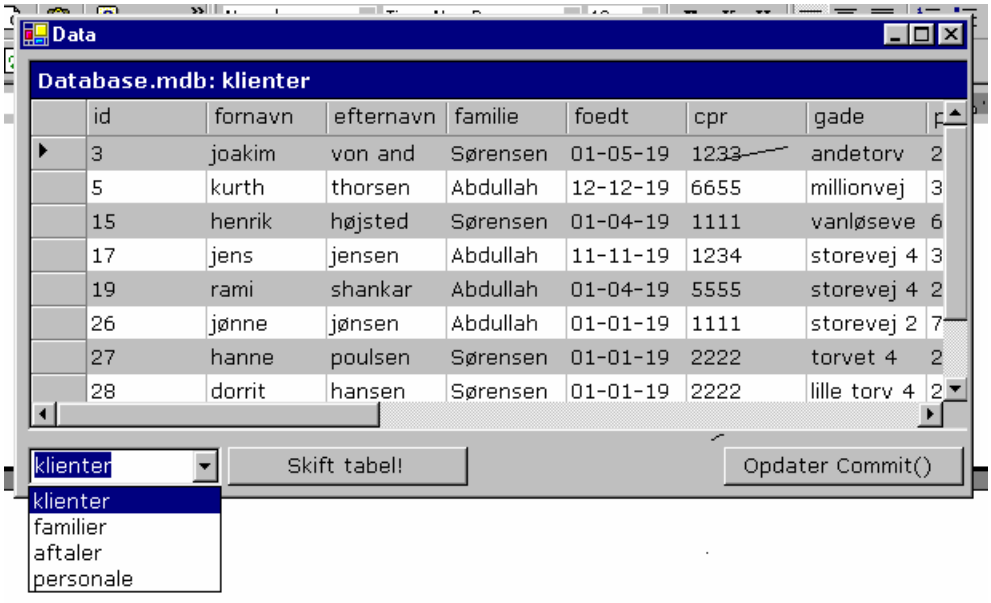
En DataGrid er nem, men ikke altid velegnet:

En C# DataGrid er **ikke** altid lige velegnet til at vise data – fx ikke hvis længere tekster skal gemmes og vises – eller lyd eller billeder! På http://csharpkursus.subnet.dk ligger et eksempel 'Tekst database uden DataGrid' som viser hvordan man kan vise data i almindelige tekst bokse. Denne fil viser også hvordan man er nødt til **manuelt** at kontrollere om brugeren overholder tabellens **constraints** dvs regler om primær nøgler, fremmednøgler og nul værdier. Det er interessant at en række af de funktioner som en DataGrid udfører automatisk – er man ellers nødt til at udføre selv:



Opgaver:

1. Skriv nogle flere tabeller i Access og få dem vist i programmet
2. en datagrid kan sættes til: datagrid.ReadOnly=true/false. Opret 2 knapper som skifter mellem disse 2 tilstande for datagrid'en!
3. Opret en ComboBox der kan skifte mellem forskellige tabeller i boghandel.mdb. en combobox har hele tiden værdi der hedder SelectedItem og de kan så sætte navnene på tabellerne ind i comboboksen efter nedenstående model:



At anvende en tekst database:

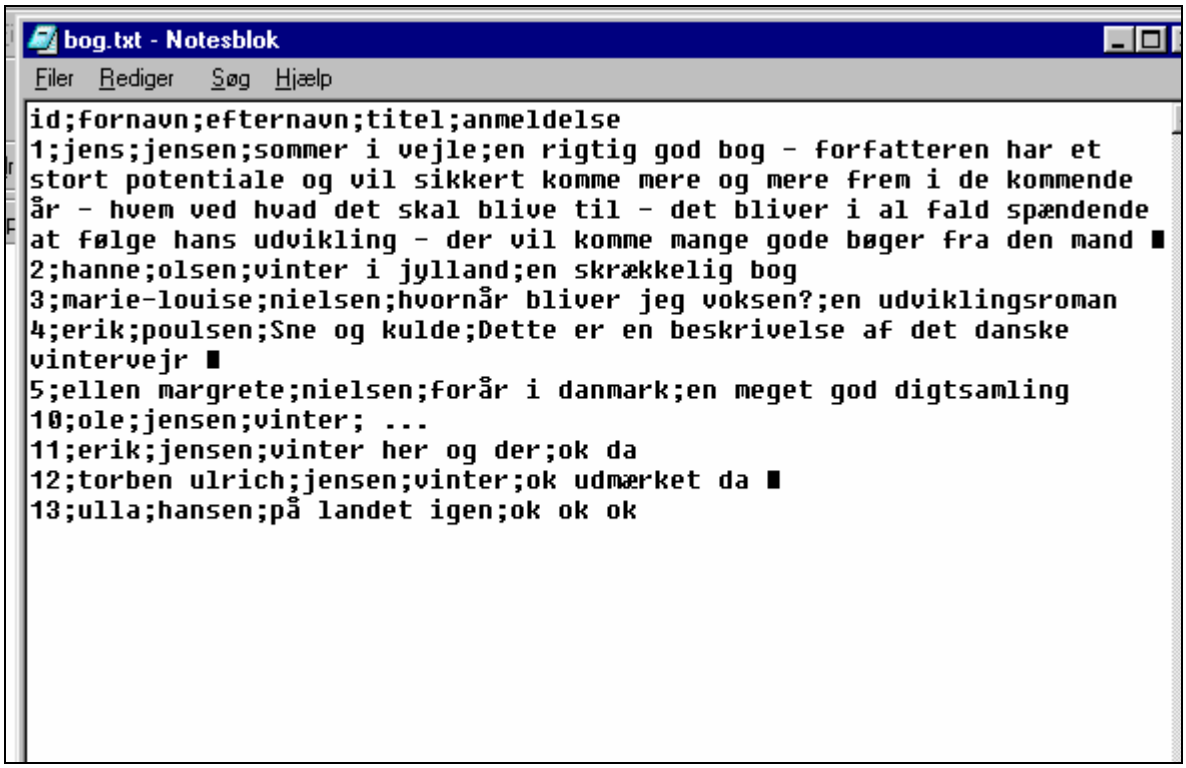
Problemet med at anvende databaser i C# og .NET programmer kan være at der er så mange **forskellige** slags database systemer (DBMS – Data Base Management Systems) at der i praksis kan opstå problemer med at koble sig på en bestemt slags database.

Nedenstående eksempel løser dette på en simpel måde: **Konverter** databasen til en **tekstdatabase** (som egentligt bare er en 'flad' tekstfil) og brug den! Alle databaser kan konverteres til ren tekst. Desuden findes gamle databaser som simpelt hen **er** tekstdatabaser (Mange af dem har fil typen *.csv – komma separerede filer).

I dette eksempel har vi taget udgangspunkt i **Microsoft Access**-databaser og konkret i tabellen Bog i den database Boghandel som vi tidligere har brugt – men du kan sagtens bruge et andet eksempel! NB kun een tabel kan konverteres ad gangen!

I **Access** vælges Filer -> Eksporter og der vælges filtypen tekst. Filen bliver her gemt i mappen dokumenter men det er selvfølgeligt endnu nemmere at gemme den i den mappe hvor EXE filen ligger – så behøves ingen sti til teksten.
Derefter guides man gennem konverteringen. Her er valgt **semikolen** som grænse mellem kolonnerne og **ingen** tekst afgrænser. Desuden er valgt at **feltnavne** skal med i første linje Husk det!

Tabellen kommer så til at se sådan ud som Bog.txt:



Tabellens felter og rækker er oversat til ren tekst. Som det ses **afgrænses** to felter med semikolon. Dette kan vi så udnytte i C# koden ved simpelt hen at læse (**parse**) filen og lægge data ind i en **DataTable** og et **DataSet** sådan:

```
//Eksempel på anvende en tekst database:

using System;
using System.Windows.Forms;
using System.IO;
using System.Drawing;
using System.Data;

public class DB: Form{

        private string fil=null;
        private StreamReader r=null;
        private DataGrid grid;
        private DataRow rk;

        public DB(string tekst){

                CenterToScreen();
                ClientSize=new Size(500,300);
                grid=new DataGrid();
                grid.Location=new Point(5,5);
                grid.Size=new Size(490,290);
```

```
                grid.CaptionText=@"Data fra Tekst Databasen: '"+tekst+@"'";

grid.Anchor=AnchorStyles.Top|AnchorStyles.Bottom|AnchorStyles.Left|AnchorStyles.Right;
                Controls.Add(grid);

                DataTable tabel=new DataTable("T");
                try{
                r=File.OpenText(tekst);
                fil=r.ReadToEnd();
                }catch{}
                r.Close();

                //Parsing:

                //Find hver linje i database filen:
                string[] linjer=fil.Split('\n');

                //Find første linje og sæt som kolonne overskrifter:

                string linje=linjer[0];
                string[] kol=linje.Split(';');
                int k;

                //kol.Length er antallet af kolonner:

                for(k=0;k<kol.Length;k++){
                                tabel.Columns.Add(new DataColumn(kol[k],typeof(string)));
                }
                int i=1;//nu er vi i række 1

                while(linjer[i]!=""){

                linje=linjer[i];

                //find kolonnernes værdier:

                kol=linje.Split(';');
                rk=tabel.NewRow();
                for(k=0;k<kol.Length;k++){
                                rk[k]=kol[k];
                }
                tabel.Rows.Add(rk);
                i++;
                }

                //bind grid til datasettet og vis grid:

                DataSet dataset=new DataSet();
                dataset.Tables.Add(tabel);
                grid.SetDataBinding(dataset,"T");

                //gem data i XML fil

                dataset.WriteXmlSchema(tekst+"_skema.xml");
                dataset.WriteXml(tekst+".xml");
```

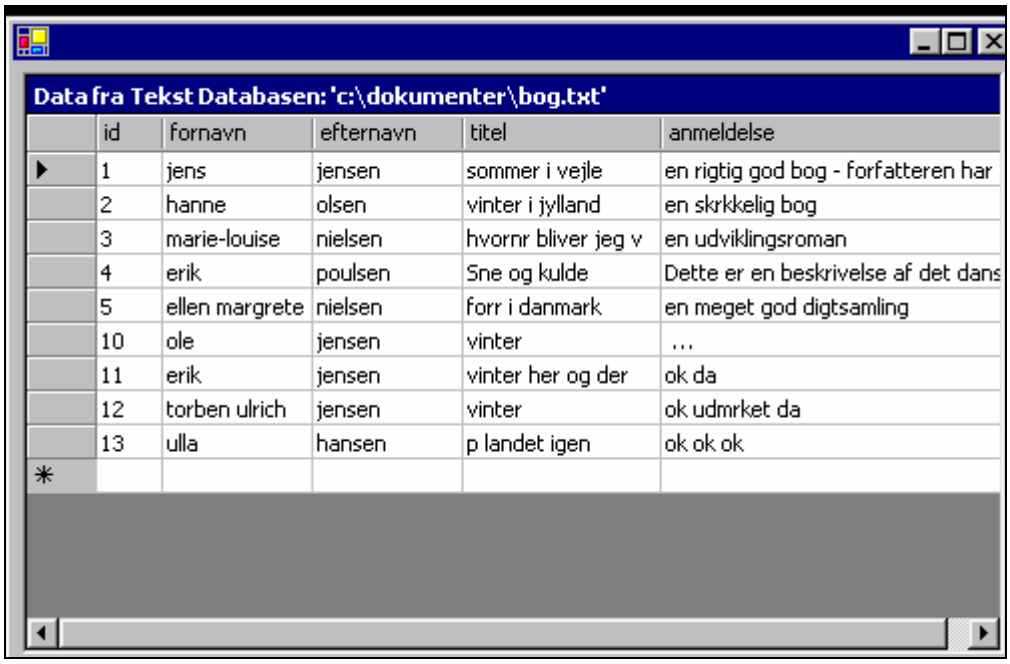
```
                }

}

class MainClass
{
                public static void Main(string[] args)
                {

                                //kunder.txt er en meget stor tabel fra MS NorthWind
                                //eksporteret fra Access til tekst database:

                                Application.Run(new DB("c:\\dokumenter\\bog.txt"));
                }
}
```

Koden er forklaret løbende ovenfor. Klassen DB instantieres altså altid med en bestemt tekst. Her indlæses kun en tabel i datasettet **men** man kunne **let** indlæse 10 forskellige tabeller i det samme dataset!

Den vigtigste metode er metoden **Split**() i System.Strng som er en 'string **tokenizer'** der deler en lang streng i dele (skaber en streng tabel) efter et tegn (en **separator** fx ';').

Fordelen ved ovenstående metode er at vi kan indlæse en hvilket som helst tabel fra en hvilket som helst database - også uden at kende antallet af kolonner (felter) i tabellen. C#'s komponenter som DataGrid og DataSet viser her deres store styrke og fleksibilitet!

Resultatet af at køre programmet er:



	id	fornavn	efternavn	titel	anmeldelse
▶	1	jens	jensen	sommer i vejle	en rigtig god bog - forfatteren har
	2	hanne	olsen	vinter i jylland	en skrkkelig bog
	3	marie-louise	nielsen	hvornr bliver jeg v	en udviklingsroman
	4	erik	poulsen	Sne og kulde	Dette er en beskrivelse af det dans
	5	ellen margrete	nielsen	forr i danmark	en meget god digtsamling
	10	ole	jensen	vinter	...
	11	erik	jensen	vinter her og der	ok da
	12	torben ulrich	jensen	vinter	ok udmrket da
	13	ulla	hansen	p landet igen	ok ok ok
*					

At anvende en database på Internettet eller et netværk:

Fordelen ved **XML** er at det bl.a. er nemt at transportere gennem netværk.

En databaseform som vi har set eksempler på kan opstartes simpelthen ved at hente et database XML **'skema'** fra et netværk eller fra Internettet.

Nedenstående er et eksempel på hvordan en sådan form starter:

Eksemplet er identisk med det tidligere uden al koden vedrørende opkobling til Access databasen. Det eneste der sker er at et skema for tabellen/tabellerne hentes et eller andet sted fra.

Som illustration er 'boghandel_skema.xml' lagt på adressen:

http://csharpkursus.subnet.dk/boghandel_skema.xml

Prøv at **downloade** filen derfra og etabler et dataset med dataset.**ReadXmlSchema**()!

Du kan også lægge XML filer i http://localhost og loade dem derfra. Det er stort set det samme.

Man kan her se at det er rimeligt nemt at hente og gemme databaser på netværk via XML (Hvis man ellers har rettigheder til at gemme det pågældende sted!).

ODBC drivere:

ODBC eller **Open Data Base Connectivity** er den traditionelle måde at kommunikere med databaser på. ODBC er ikke egentligt en del af .NET. Men ODBC kan downloades fra http://msdn.microsoft.com.

Du skal downloade filen **odbc_net**.msi, som så kan **installere** disse ODBC drivere på dit system. Du kan se om de bliver installeret i Windows Kontrolpanelet under ODBC.

Når det er gjort kan du koble op til en række forskellige slags databaser – afhængigt af det system du p.t. har! Men du burde – fra et C# program - kunne koble op til Access databaser, Paradox databaser, Text databaser, dBase databaser og Oracle databaser.

Du skal registrere en database i Windows Kontrolpanel med et 'globalt' navn (en DSN eller Data Source Name).

Følgende er et lille forkortet eksempel på hvad man så kan gøre:

```
using System;
using Microsoft.Data.Odbc;//skal downloades fra msdn
using System.Data;
using System.Windows.Forms;

class MainClass
{
		public static void Main(string[] args)
		{
			OdbcConnection cn=new OdbcConnection();

			//Data fra Paradox database ( *.db ): country.db registreret som DSN borland
			cn.ConnectionString="Provider=MSDASQL;DSN=borland;";

			try{
			cn.Open();
			}catch{
				MessageBox.Show("Der er ingen forbindelse.","Forbindelsen til
databasen:");

			}

			OdbcDataAdapter adapter=new OdbcDataAdapter("select * from country",cn);
			DataSet dataset=new DataSet();
			adapter.Fill(dataset,"country");

			//Opret en form med en datagrid:

			Form f=new Form();
			DataGrid grid=new DataGrid();
			grid.Dock=DockStyle.Fill;
			f.Controls.Add(grid);
			grid.SetDataBinding(dataset,"country");

			f.ShowDialog();


			cn.Close();
		}
}
```

Det afgørende nye er:

```
using Microsoft.Data.Odbc;
```

som gør at vi kan arbejde med helt andre klasser. **Ulempen** ved at bruge disse ODBC drivere er at de er helt specielle for Microsoft – og derfor ikke helt passer ind i .NET konceptet! (Fordelen er at de fungerer!).

Som det ses anvendes grundlæggende de **samme** metoder som vi har set tidligere. Det ses også her hvor kort koden kan være!

Ressourcer:

Mange har sikkert oplevet Windowsprogrammer som ikke kan starte, som crasher eller ikke virker korrekt, fordi programmet pludseligt **mangler** en ressource eller fil. Det kan være et ikon, en bitmap, en fil eller en DLL. Det hjælper ikke meget at jeg har designet et flot ikon til mit nye program hvis jeg har 'glemt' at levere det sammen med programmet! Det kan også være at programmet skal vise en bestemt fil som så skal ligge i samme mappe som EXE filen. Men hvad nu hvis brugeren sletter, redigerer i, omdøber eller flytter filen eller EXE programmet!!

NB I dette afsnit bruges **Bitmap,** men denne type kan i C# rumme en lang række af billedformater også jpg, gif, tiff, png osv. og **ikke kun** Windows bitmaps (*.bmp filer).

Den eneste måde man kan sikre sig mod dette på er at **'embedde'** (integrere) sine ressourcer i selve programmet således at et ikon ikke skal ligge ved siden af programmet, men simpelthen er en **del** af EXE filen (eller DLL filen).
Problemet med at 'embedde' sine ressorcer kan løses på **flere** måder i C#:

Eksempel: ResourceWriter:

De tekster og billeder jeg vil have ind i programmet kan gemmes i XML som er det foretrukne format i .NET og C#. Dette sker rimeligt enkelt på denne måde (NB nogle af linjerne er bevidst kommenteret ud):

```
//Illustrerer hvordan ressourcer kan bindes til exe filen:

//Pointen: at sikre at tekster, billeder og ikoner er tilstede!
//uanset hvor hen programmet flyttes!

using System;
using System.Windows.Forms;
using System.Resources;
using System.Drawing;
using System.Reflection;
using System.IO;
using System.Collections;
using System.Collections.Specialized;


namespace MyFormProject
{
                class ResForm : Form
```

```
	{
		private void bind_ress(){

		ResourceWriter writer=new ResourceWriter("eksempel.resources");
		writer.AddResource("farver",new Bitmap("farver.bmp"));
		writer.AddResource("titelfarver","Dette er farvet.bmp");
		writer.AddResource("ikon",new Icon("ikon.ico"));

		//OBS en fil kan bindes som ressource!:
		//StreamReader reader=File.OpenText("mainform.cs");
		//string fil=reader.ReadToEnd();
		//writer.AddResource("denne fil",fil);

		//Generate() opretter filen eksempel.resouces:
		writer.Generate();


		}
		private ResourceManager manager;


		public ResForm()
		{
			//NB her er anvendt en primitiv exception handling:
			try{bind_ress();}catch{}

			manager=new ResourceManager ("eksempel",
Assembly.GetExecutingAssembly());

			InitializeComponents();

		}

		void InitializeComponents()
		{
			this.SuspendLayout();
			this.Size = new System.Drawing.Size(500, 300);

			//hent ressourcer:
			//BackgroundImage=(Bitmap)manager.GetObject("farver");
			//Text=manager.GetString("titelfarver");
			//Icon=(Icon)manager.GetObject("ikon");
			this.ResumeLayout(false);
		}

		[STAThread]
		public static void Main(string[] args)
		{
			Application.Run(new ResForm());
		}
	}
}
```

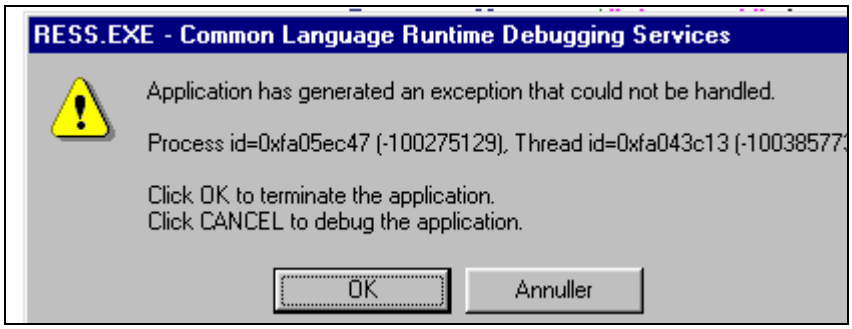
NB! Du skal åbne et nyt projekt i **SharpDevelop** og der bruge denne fil i stedet for Mainform.cs! (Det er i al fald det nemmeste!).

I projektmappen skal der ligge et ikon, en bitmap og en cs fil – og du må sørge for at navnene kommer til at passe sammen – ellers vil intet virke!!

Det nye her er metoden bind_ress(). Denne metode kalder en ressource writer som skriver en *.resources fil med de 3 ressourcer: et ikon, et billede og en tekst. Yderligere er vist hvordan en hel fil kan lægges ind som ressource.

Hvis vi nu kører programmet går alt godt (en tom form), fordi vi endnu **ikke** har brugt vore ressourcer. Men læg mærke til at der er oprettet en **eksempel.resources** i mappen!
Prøv derefter at fjerne kommentar linjerne ud for manager.GetObject() osv, kompiler og kør programmet.

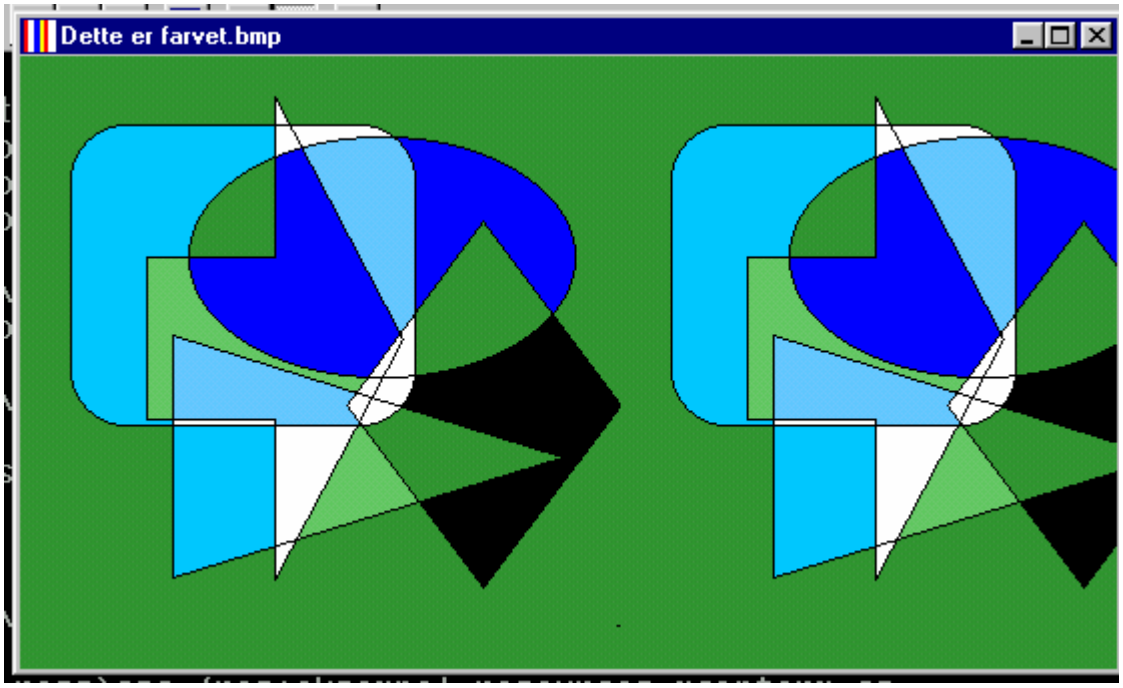
Vi får nu en fejl:



Når man prøver at embedde sine ressourcer kan man få rigtigt mange af den slags fejl!

Problemet er at vi ikke har **tilføjet** eksempel.resources til vores mainform.cs.
Dette kan ske ved at du højre klikker Ressourcer i Projektet (Vælg View Projects) og vælger Add Resource.

Herefter skulle resultatet bliver nogenlunde sådan her – men du har sikkert ikke mine ikoner og billeder (brug dine egne):



Du skal nu **fjerne** kaldet til bind_ress() (kommenter det ud med //) – ellers bliver programmet ved med at skrive en ny *.resources fil (og det er **IKKE** meningen).

Egentligt har vi jo slået 2 programmer sammen til eet – fordi selve skabelsen af resources filen jo skulle foregå **FØR** selve programmet!

Prøv nu at **flytte** dit program til en anden mappe og **kontroller** at det virker – **også** selv om ikoner og bitmaps ikke ligger i samme mappe!!

Der ligger andre eksempler på brug af ressource filer på http://csharpkursus.subnet.dk.

Skriv dine ressourcefiler i hånden:

Ressourcefiler kan i mange tilfælde meget nemmere skrives manuelt endda såre simpelt. Sådanne filer har typen *.**res**.

Et eksempel herpå er følgende som er skrevet i Notesblok og gemt som 'hjemmelavet.**res'**:

```
firma="Computer APS"
adresse="Storevej 44"
telefon="34345667"
email=mail@computeraps.dk
```

I den mappe hvor SharpDevelop er installeret findes i /bin en fil resasm.exe som kan 'assemble' res filen således:

```
resasm hjemmelavet.res
```

Herudaf kommer 'hjemmelavet.resources' som kan tilføjes projektet.

Man kan også kompilere projektet på kommandolinjen med **/res**: således:

csc **/res**:hjemmelavet.resources main.cs

På den måde bliver ressourcerne i hjemmelavet.resources embedded i programmet. Husk at ændre **ressourcemanageren** til navnet 'hjemmelavet'!

Hvis man kompilerer **uden** /res flaget fås:

```
Unhandled Exception: System.Resources.MissingManifestResourceExceptio
t find any resources appropriate for the specified culture (or the ne
re) in the given assembly.  Make sure "hjemmelavet.resources" was cor
dded or linked into assembly "MainForm".
baseName: hjemmelavet  locationInfo: <null>  resource file name: hjem
ources  assembly: MainForm, Version=0.0.0.0, Culture=neutral, PublicK
l
   at System.Resources.ResourceManager.InternalGetResourceSet(Culture
```

*.resources filer er 'ulæselige' **binære** filer, men de kan **dis-assembles** med den samme SharpDevelop utility (/d: står for disassemble):

resasm /d:hjemmelavet.**resources**

producerer en hjemmelavet.**res** fil som kan læses i en tekst editor.

Gem ressourcer som XML:

En anden metode er at gemme sine ressourcer i XML formatet som er det foretrukne format i C# og .NET:

Dette resulterer i en XML fil eksempel.resx, som kan kompileres med en utility tool **resgen**.exe til en *.resources fil. (resgen findes desværre hverken i SharpDevelop eller i .NET redistributable!).

I XML filen **'eksempel**.**resx'** bliver data gemt som name=value par.

Fx er her vist hvordan **bitmappen** gemmes i eksempel.resx:

```
  </resheader>
- <data name="farver" type="System.Drawing.Bitmap, System.Drawing, Version=1.0.3300.0,
    Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" mimetype="application/x-
    microsoft.net.object.bytearray.base64">

      <value>Qk1WvwIAAAAAADYAAAAoAAAALAEAACwBAAABABAAAAAAAAAAAADEDgAAxA4AA
      RRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWR
      RRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWR
      RRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWR
      RRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWR
      RRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWR
      RRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWR
      RRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWRRZFFkUWR
```


Hver ressource starter med **<data** og indeholder en **type mimetype** og en **value**. Prøv selv at gemme på denne måde og se hvor lang bitmap ressourcen er!

Andet eksempel fra samme fil:


```
- <data name="ikon" type="System.Drawing.Icon, Sys
    PublicKeyToken=b03f5f7f11d50a3a" mimetype="a

      <value>AAABAAEAICAQAAAAAAADoAgAAFgAAAC
      AAAAAAAAAACAAACAAAAAgIAAgAAAAIAAgACAg
      AAD///8AmZn////MzMzLu7mZn////5mZ////z
      zMzLu7mZn////5mZ////zMzMy7u5mZ////+Zm
      mZmf////mZn////MzMy7u5mZn////5mZ////z
      zMy7u5mZn////5mZ////zMzMu7uZmZ////+Zm
      uZmf////mZn////MzMzLu7mZn////5mZ////zM
      AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
      AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
  </data>
- <data name="titelfarver">
    <value>Dette er farver.bmp</value>
```


Her ses data name ”ikon” og vores gemte streng ”titelfarver”.

Prøv at skriv følgende metoder ind i ovenstående program så du kan få programmet til at gemme i XML (og undersøg XML filen i InternetExplorer):

```
private void skriv_XML_ress(){

        //en ResXResourceWriter er en ResourceWriter
        //der i stedet for en *.resources fil skriver en (lang!) XML fil:
        ResXResourceWriter xmlwriter=new ResXResourceWriter
("eksempel.resx");

        Image f=new Bitmap("farver.bmp");
        xmlwriter.AddResource("farver",f);
        Icon ikon=new Icon("ikon.ico");
        xmlwriter.AddResource("ikon",ikon);


        //en streng tilføjes direkte:
        xmlwriter.AddResource("titelfarver","Dette er farver.bmp");
```

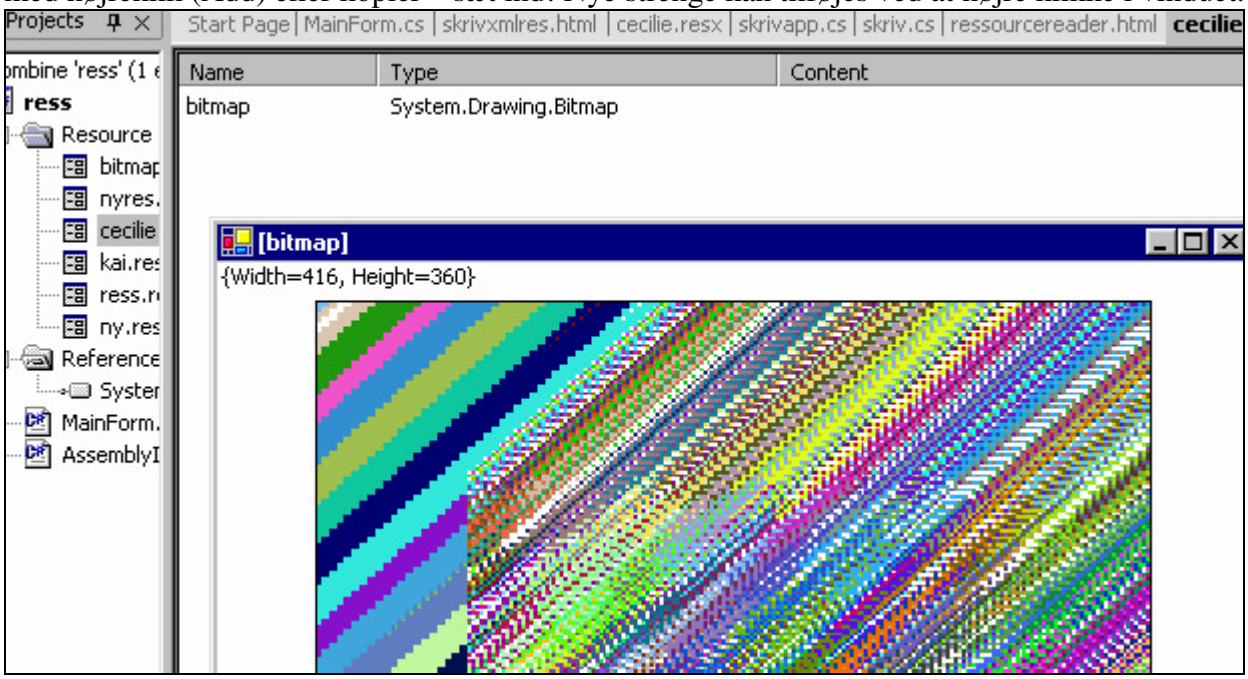
```
                                        //OBS en fil kan bindes som ressource!:
                                        //StreamReader reader=File.OpenText("mainform.cs");
                                        //string fil=reader.ReadToEnd();
                                        //xmlwriter.AddResource("denne fil",fil);

                                        xmlwriter.Generate();
                                        xmlwriter.Close();

                                }

                                //læs vores egen eksempel.rex XML fil:
                                private void get_XML_ress(){
                                        ResXResourceReader reader=new ResXResourceReader
("eksempel.resx");
                                        IDictionaryEnumerator e=reader.GetEnumerator();
                                        string navn=null,vaerdi=null;

                                        while(e.MoveNext()){
                                                navn=e.Key.ToString();//Key er selve objektet!
                                                vaerdi=e.Value.ToString();
                                                MessageBox.Show(vaerdi,navn);
                                        }
                                        reader.Close();
                                }
```

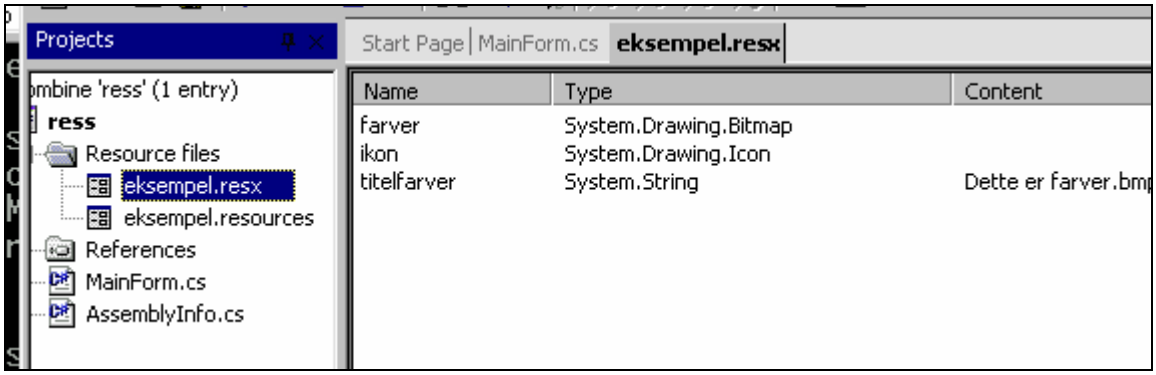
Som det ses kan meget lange ressourcer både gemmes og hentes frem gennem XML formatet. I vores sammenhæng er *.resources filer dog nemmere at bruge!

I programmet **SharpDevelop** er en række ressourcer gemt som XML også fx dets About Dialog boks (på dansk: Om Programmet). Her er et udsnit af den xml fil som definerer dialog boksen og dens OK knap:

```
                                        <ICSharpCode.SharpDevelop.Gui.
                                </Controls>
                        </System.Windows.Forms.TabPage>
                </TabPages>
        </System.Windows.Forms.TabControl>

        <!-- OK Button -->
        <System.Windows.Forms.Button>
                <Name    value = "okButton"/>
                <Width   value = "80"/>
                <Height  value = "24"/>
                <Location value = "{X=328, Y=472}"/>
                <TabIndex value = "1"/>
                <Text     value = "${res:Global.OKButtonText}"/>
                <DialogResult value = "OK"/>
                <FlatStyle value = "${FlatStyle}"/>
        </System.Windows.Forms.Button>
  </Controls>
em.Windows.Forms.Form>
```

Både *.resources filer og *.resx filer kan **åbnes** og **redigeres** (!!) i fx SharpDevelop, og man kan se hvad ressourcen indeholder (Det kan ellers være svært at huske!). Man kan tilføje fx nye bitmaps med højreklik (Add) eller kopier – sæt ind! Nye strenge kan tilføjes ved at højre klikke i vinduet.





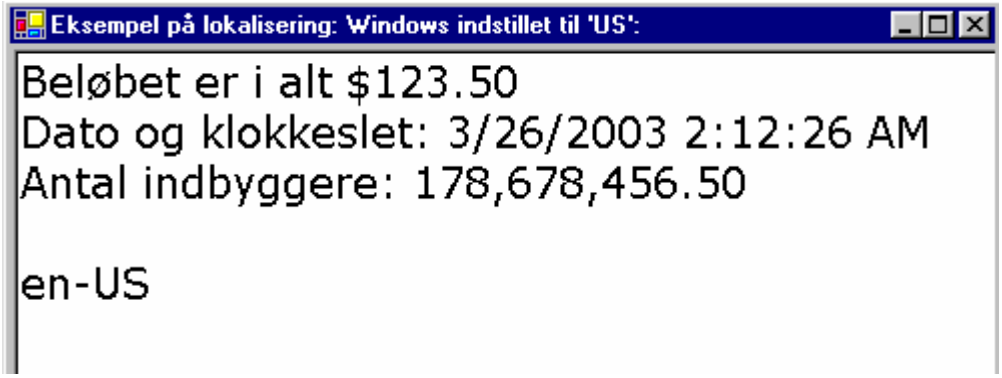
Opgaver:

1. Opret en form med en RichTextBox der viser en informationsfil som er en **embedded** resource. Flyt EXE filen og kontroller at den virker!
2. Opret en form med et ikon (embedded) som du selv har valgt – i stedet for C# standard ikonet. Flyt EXE filen og kontroller at den virker!
3. Skriv et program med en form og 2 knapper og to labels der bruger en **embedded** ressource således: Brugeren kan vælge om tekster på formen og dens knapper skal være på engelsk eller dansk. Flyt EXE filen og kontroller at den virker!
4. Skriv exception handling til disse programmer når du prøver at binde ressourcer!
5. Der findes en **ResourceReader** der er fuldstændigt mage til ResXResourceReader – skriv et program hvor du læser en *.resources fil med en ResourceReader!

Lokalisering og ressourcer:

Når Windows installeres sættes der altid en 'locale' værdi d.v.s. Windows installeres med et bestemt sprog og geografisk område.

Hvis Windows installeres med engelsk og USA som område fås fx dette resultat:



Koden hertil ser i fragment sådan ud:

```
                                        string valuta=String.Format("Beløbet er i alt {0:C}",123.50);
                                        box.Text=valuta;
                                        string dato=String.Format("\nDato og klokkeslet:
{0}",DateTime.Now.ToString());
                                        box.Text+=dato;
                                        string tal=String.Format("\nAntal indbyggere: {0:n}",178678456.5);
                                        box.Text+=tal;
```

Som det ses sker der helt automatisk en lokalisering af den måde teksterne formatteres på. Hvis en værdi skal formatteres som valuta ({0:C}) vises det som $ (dollar) og datoen vises efter amerikansk format: måned/dato/år hvor vi på dansk ville skrive: dato/måned/år! Se også brugen af AM! Ved formatteringen af tallet bruges komma hvor vi på dansk bruger punktum og omvendt! Alt dette sker automatisk hvis sprog-geografi værdien sættes i Windows. Man kan eksperimentere med dette ved at ændre indstillingen i Windows Kontrolpanel (International)!

Den nederste linje 'en-US' fås ved i koden at skrive:

box.Text+="\n\nCurrentUICulture: "+Thread.CurrentThread.CurrentUICulture.ToString();

Enhver tråd/proces/program kører – som det ses – altid i en bestemt 'kultur' (fx 'en-GB', 'en-US', 'da-DK', 'sv', 'no' osv – man behøver ikke angive geografisk område: 'da' er tilstrækkeligt til at angive at kulturen er dansk sprog!).

Meningen med at 'lokalisere' et program er altså at visse tekster, billeder osv **automatisk** skal vises på det sprog osv som brugeren har installeret i Windows. Hvis jeg producerer et 'lokaliseret' program skal det altså køre på dansk hvis det bruges i Danmark, på engelsk hvis det bruges i England, på tysk hvis det kører på en 'tysk' maskine osv! Det er ikke meningen at brugeren skal stille på programmet for at få det til at køre 'på tysk' f.eks. Det skulle gerne ske helt automatisk!

I .NET er der skabt muligheder for at lave en sådan automatisk lokalisering og vi vil nu se et **enkelt eksempel** på hvordan lokalisering kan gennemføres i .NET. (I virkeligheden kan man lokalisere et program på mange forskellige måder – mere eller mindre elegant!).

Vi starter med at skrive en enkel ressource fil: strings.res med 2 værdier:

---

titel="Denne applikation er lokaliseret til: "

tekst="Dette er en dansk tekst ... Dette er en dansk tekst ... Dette er en dansk tekst"

---

strings.res kompileres til en binær ressource fil strings.resources sådan:

resasm strings.res

Vi skriver herefter en form DLL fil som anvender ressourcer sådan:

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.Threading;
using System.Globalization;
using System.Reflection;
using System.Resources;



    public class MainForm : Form
    {
        private RichTextBox box;
        private ResourceManager manager;

        public MainForm()
        {
            InitializeComponents();
        }

        void InitializeComponents()
        {
            this.SuspendLayout();
            this.Name = "Globalisering";
            this.Size = new System.Drawing.Size(500, 300);
            box=new RichTextBox();
            box.Location=new Point(2,2);
            box.Size=new Size(496,296);
            box.Font=new Font("Verdana",16);
            Controls.Add(box);
            manager=new ResourceManager ("strings", Assembly.GetExecutingAssembly());
            string titellinje=manager.GetString("titel");
            Text=titellinje;
            string entekst=manager.GetString("tekst");
```

```
            box.Text=entekst;

            Text+=" - CurrentUICulture: "+Thread.CurrentThread.CurrentUICulture.ToString();


            this.ResumeLayout(false);
        }


    }
```

Denne fil kompileres således:

csc /t:library /res:strings.resources lokal.cs

Herved får vi **lokal**.**dll**. Vi kan nu instantiere denne form i en applikation fx sådan i en fil kaldet lokapp.cs:

```
// csc /r:lokal.dll lokapp.cs


using System;
using System.Windows.Forms;
using System.Drawing;
using System.Threading;
using System.Globalization;
using System.Reflection;
using System.Resources;




class app{

            public static void Main(string[] args){

                                    try{
                                    Thread.CurrentThread.CurrentUICulture=new CultureInfo(args[0]);
                                    }catch{MessageBox.Show("Fejl i angivelsen af kultur!","Fejl");}


                        MainForm f=new MainForm();
                        f.ShowDialog();
            }
}
```

Filen kompileres med en reference til DLL filen:

csc /r:lokal.dll lokapp.cs

lokapp.exe (som fremkommer) skal så startes med en parameter der er en gyldig 'kultur' som fx 'da-DK'. Vi bruger denne løsning for at kunne skifte kultur på maskinen på en nem måde! I en rigtig applikation ville vi selvfølgelig ikke gøre dette – selve Windows indstillingen ville automatisk vælge det rigtige sprog!!

Hvis programmet opstartes med en ikke eksisterende UICulture fx med:

lokapp xy

fås en fejlmelding (og derefter programmet 'på dansk' som her er default eller standard):

.NET er altså i stand til med det samme at checke om strengen er en gyldig UI kultur.

Hvis programmet startes med parameteren 'da-DK' fås følgende:

Vi er nu klar til at lokalisere programmet. .NET **kræver** her at der oprettes en undermappe (under den mappe hvor exe filen og lokal.dll ligger) som opkaldes efter sprog/lande koden.

Vi opretter til en start en undermappe: 'en-US'. I denne mappe placerer vi en amerikansk udgave af strings.res som kompileres med resasm til strings.en-US.resources:

---

titel="This application is localized to: "

tekst="This is a very short and very stupid American text ... "

Vi opretter derefter en stort set **tom** cs fil lokal.resources.cs med en attribut som sætter kulturen til 'en-US' sådan:

```
using System.Reflection;
```

```
using System.Resources;
using System.Globalization;

[assembly: AssemblyCulture("en-US")]
```

Denne fil kompileres nu sammen med US ressourcefilen sådan:

csc /res:strings.en-US.resources /t:library lokal.resources.cs

og vi får en ny DLL: **lokal**.**resources**.**dll**.

Det er meget vigtigt at filerne kommer til at hedde det rigtige – ellers vil lokaliseringen **ikke** fungere. Hvis den oprindelige dll fil hedder **lokal**.dll så **skal** de forskellige sprogs filer alle hedde **lokal**.**resources**.dll og de **skal** ligge i hver sin mappe med navn efter sprogkoden.

Filer som lokal.ressources.dll kaldes i .NET for en 'satellit assembly'.

Vi kan nu oprette en mappestruktur fx sådan (men evt med mange flere sprog!):

I hver mappe skal der så ligge en strings fil og en dll fil. Her er nogle eksempler:

I mappen 'fr' placerer vi denne cs fil:

```
using System.Reflection;
using System.Resources;
using System.Globalization;


[assembly: AssemblyCulture("fr")]
```

Denne lokale lokal.resources.dll kompileres så sammen med en fransk sproget ressource fil fx:

```
titel="Ici - La France: "

tekst="Paris est une tres belle ville - le capital de la France!"
```

Hvis vi nu starter vores applikation (i hovedmappen) med lokapp fr får vi en lokaliseret udgave på fransk:

Vi kan skrive en tysk ressource fil:

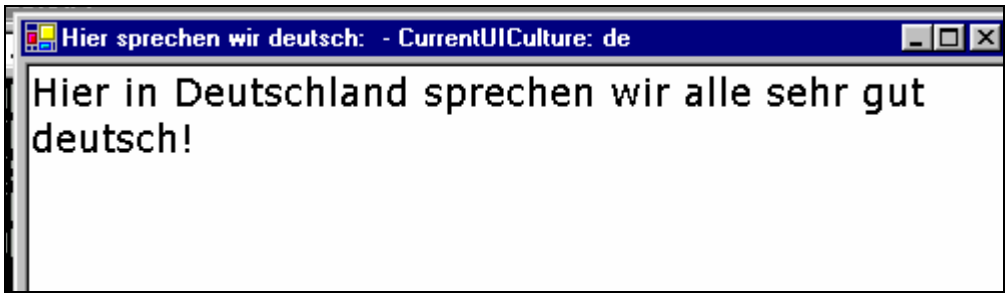
---

titel="Hier sprechen wir deutsch: "

tekst="Hier in Deutschland sprechen wir alle sehr gut deutsch!"

---

og få en tysk udgave osv:



I dette eksempel har vi gjort dansk til default sprog. Dvs. hvis intet andet angives vil programmet køre på dansk. Dette er bare et valg – man kunne vælge hvad som helst som default – men det er nødvendigt at der i hovedmappen ligger en default DLL – her: lokal.dll – som definerer hvad der skal gælde hvis intet andet angives!

De specielle lokaliserede udgaver som fx en-US behøver **ikke** at definere alle variable. Hvis fx en-US ikke havde defineret variablen 'tekst' ville værdien blive hentet fra default DLL'en - altså i dette tilfælde lokal.dll i hovedmappen!!

En alternativ metode til lokalisering:

Lokalisering kan gennemføres på en lidt enklere måde ved at lade programmet dynamisk (under run time) vælge en bestemt ressourcefil. I dette tilfælde behøver filerne ikke at ligge i en bestemt mappestruktur. Koden kunne se sådan ud:

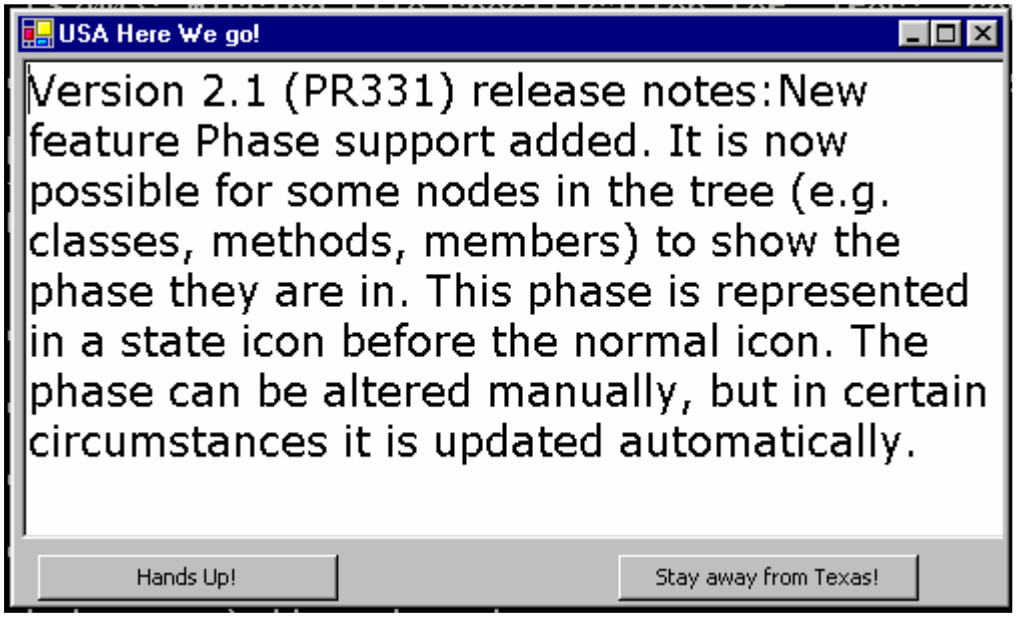
```
// lokalisering alternativ

using System;
using System.Windows.Forms;
using System.Drawing;
```

```
using System.Threading;
using System.Globalization;
using System.Reflection;
using System.Resources;


namespace MyFormProject
{
	class MainForm : Form
	{
		private RichTextBox box;
		private Button b1,b2;
		ResourceManager manager;

		public MainForm()
		{
			InitializeComponents();
		}

		void InitializeComponents()
		{
			this.SuspendLayout();
			this.Size = new System.Drawing.Size(500, 300);
			box=new RichTextBox();
			box.Location=new Point(2,2);
			box.Size=new Size(496,240);
			box.Font=new Font("Verdana",16);
			Controls.Add(box);

			b1=new Button();
			b1.Width=150;
			b1.Location=new Point(10,250);
			Controls.Add(b1);

			b2=new Button();
			b2.Width=150;
			b2.Location=new Point(300,250);
			Controls.Add(b2);


			  //der vælges en bestemt ressource ud fra CurrentUICulture:

			switch(Thread.CurrentThread.CurrentUICulture.ToString()){

			case "da": manager=new ResourceManager("da",Assembly.GetExecutingAssembly());break;
			case "en": manager=new ResourceManager("en",Assembly.GetExecutingAssembly());break;
			case "en-US": manager=new ResourceManager("en-US",Assembly.GetExecutingAssembly());break;
			case "it": manager=new ResourceManager("it",Assembly.GetExecutingAssembly());break;
			case "de": manager=new ResourceManager("de",Assembly.GetExecutingAssembly());break;
			default: manager=new ResourceManager("da",Assembly.GetExecutingAssembly());break;
			}

			  //NB alle ressource filerne behøver ikke at definere alle værdierne:
			Text=manager.GetString("titel");
			b1.Text=manager.GetString("b1");
			b2.Text=manager.GetString("b2");
			box.Text=manager.GetString("fil");
```

```
            this.ResumeLayout(false);
        }

        public static void Main(string[] args)
        {

            try{
            Thread.CurrentThread.CurrentUICulture=new CultureInfo(args[0]);
            }catch{MessageBox.Show("Fejl: Ingen gyldig værdi for CultureInfo!","Fejl");}

            Application.Run(new MainForm());
        }

    }
}
```

Metoden går simpelthen ud på at programmet vælger en af en række .resources filer som ligger i samme mappe som EXE filen.

Nogle eksempler:



Opgaver:

1. Skriv et demoprogram med to menuer fx Filer og Rediger. Skriv en lokaliseret udgave af programmet så det kan køres på 3 forskellige sprog sådan at menupunkterne optræder på et af de 3 sprog.
2. Skriv et lokaliseret program hvor du anvender andre typer ressourcer som ikoner, bitmaps eller lignende.

Hjælp!

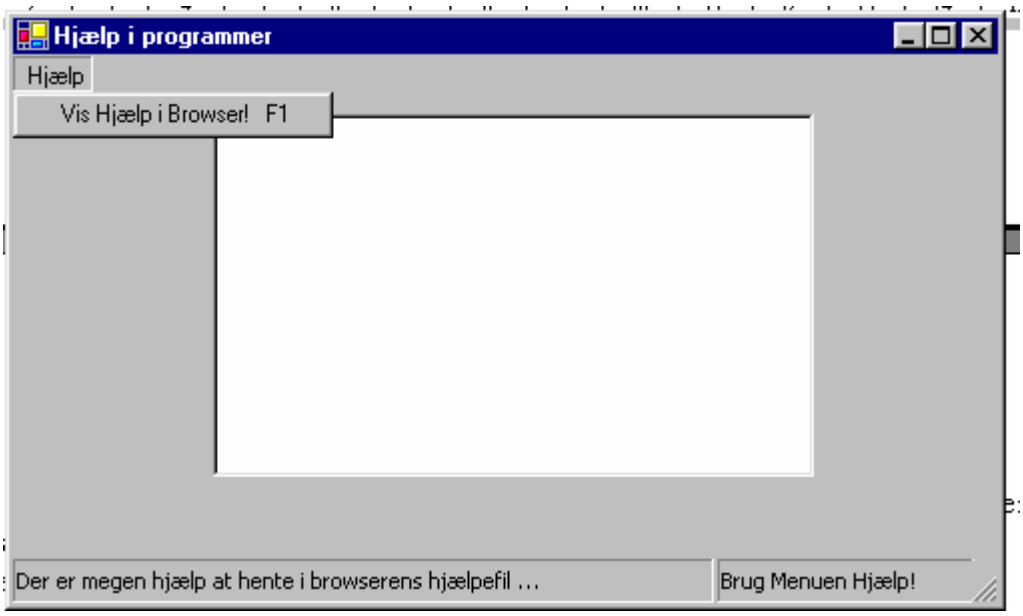
De fleste computerprogrammer er faktisk meget svære at finde rundt i for næsten alle andre end dem som har udviklet programmet.

Vanskelighederne her er enorme. Og der er skrevet tykke bøger om design og brugervenlighed.
Vi skal her se på et aspekt heraf nemlig 'Hjælp'.
De fleste programmer har en eller anden form for hjælp. I det følgende eksempel vil vi se på nogle måder at give hjælpen på:

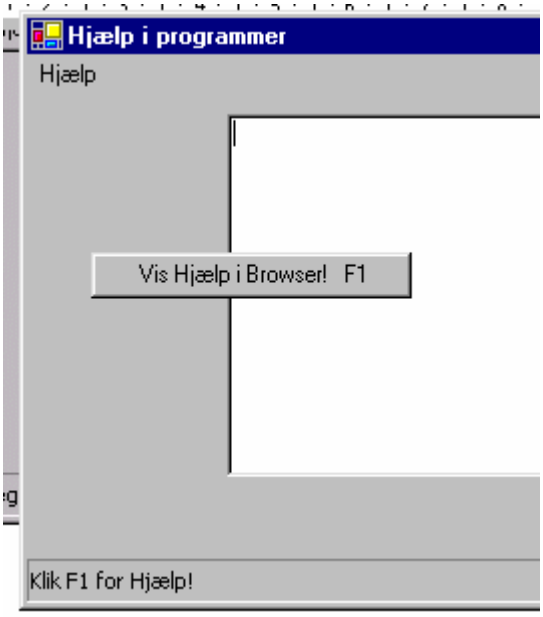
- i form af en menu Hjælp
- med 'tooltips' dvs små gule vinduer der kommer frem når musen er over et bestemt punkt
- statuslinjer
- genvejsmenuer
- Funktions tast 1 (F1)
- tabulator rækkefølge

Her er et simpelt vindue med **eksempler** på dette:

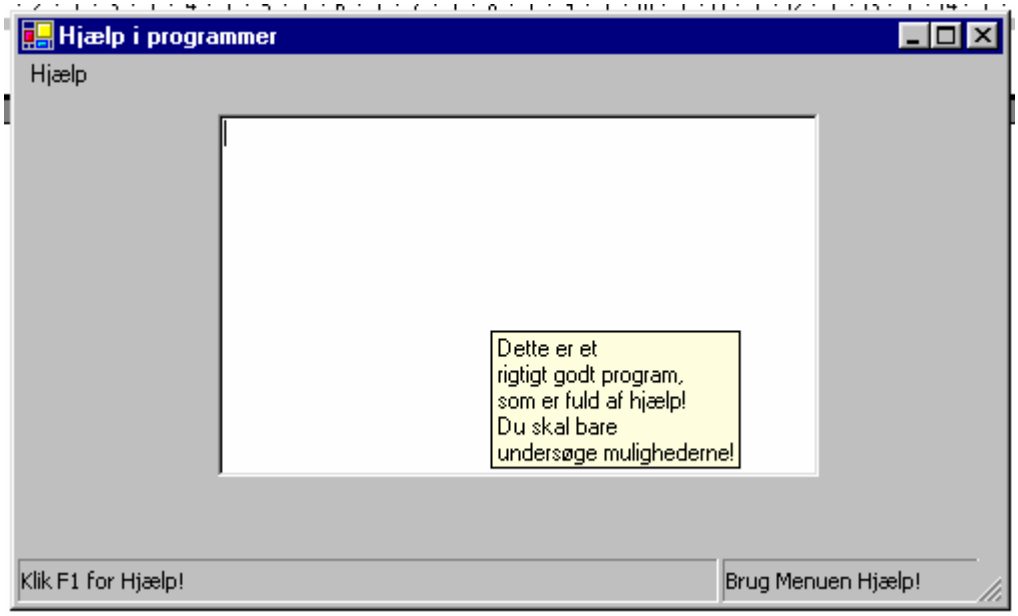
Menu og to **paneler** i statuslinjen:

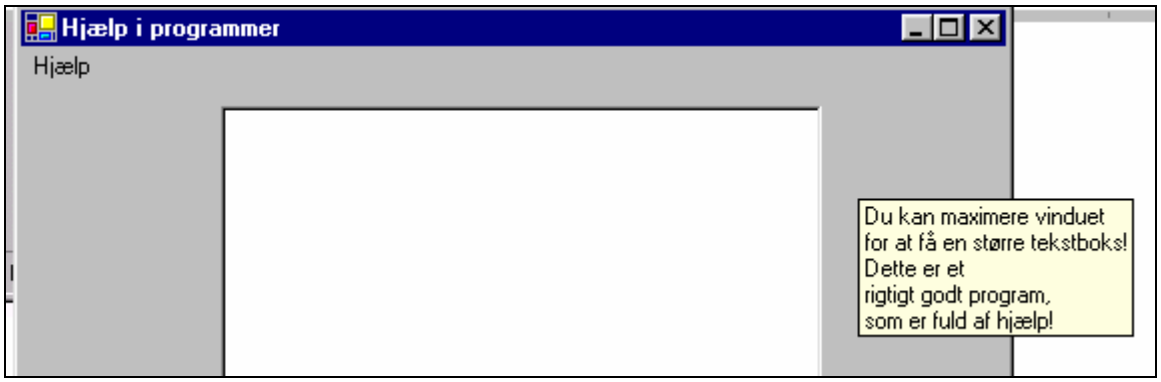


Højrekliks eller **genvejs** menu:

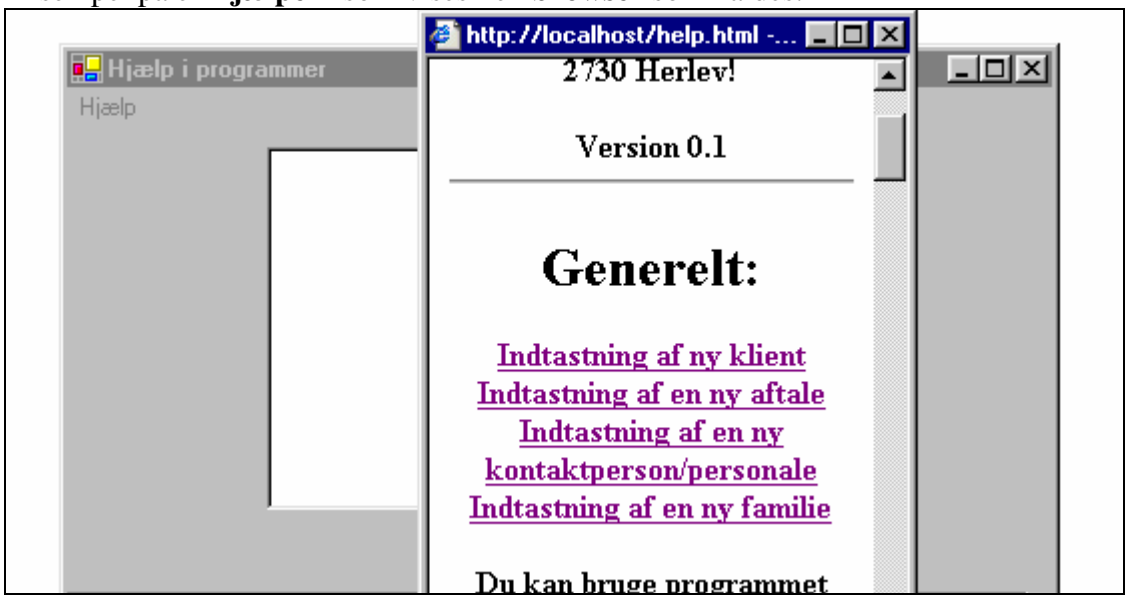


Eksempler på **tool tips:**





Eksempel på en **hjælpefil** som vises i en **browser** som kaldes:



Selve koden til programmet følger her. Der er indført **kommentarer** løbende i kildekoden.

Vigtigt er at der skal refereres til en gammel (dvs før .NET) DLL fil nemlig **SHDocVw.dll** (en **COM** – Common Object Model - fil) ellers kan vi **ikke** instantiere webbrowseren!! Denne DLL er IKKE en .NET fil!

SHDocVw.dll er en fil på 1,1 megabyte som ligger i c:\windows og som rummer de centrale MS Windows internet kontroller.

Hvis filen skal kompileres manuelt sker sådan:

csc /r:System.dll;System.Drawing.dll;System.Windows.Forms.dll;Interop.SHDocVw.dll help.cs (!!)

men det kræver at Interop filen (som er den gamle DLL fil i 'forklædning') **først** er produceret!

Så det AFGJORT nemmeste er at lave et projekt i SharpDevelop og tilføje en **reference** – og her vælge **COM** dll filer – **Microsoft Internet Controls**.

Denne fil bliver så **kopieret** over i den mappe hvor du gemmer projektet under navnet **Interop**.SHDocVw.dll.

Dette er et eksempel på hvordan man bruger **'unmanaged'** kode i .NET altså 'gammel' Windows kode. **Unmanaged** kode er typisk alle de gamle DLL filer som er udviklet til Windows gennem tiden. Al denne kode kan sagtens bruges i C# programmer men det kræver at den bliver 'forklædt' i en **'wrapper'** klasse der gør at C# opfatter den som **'managed'** kode (.NET kode).

```
//Demo af forskellig slags Hjælp i program:

//Menuen Hjælp og Hjælp i browser
//Anvendelse af F1 som genvej til Hjælp
//genvejs menu
//tool tips
//statuslinje med 2 paneler til information


using System;
using System.Windows.Forms;
using SHDocVw;
using System.Drawing;


namespace Help
{
		public class MainForm : Form
		{
				private MainMenu menu;
				private ContextMenu kontekst;
				private StatusBar status;
				private StatusBarPanel panel1;
				private RichTextBox rich;

				public MainForm()
				{
						//NB titellinjen kan evt. også bruges til 'hjælp'
```

```
                                        Text="Hjælp i programmer";
                                        Size=new Size(500,300);

                                        rich=new RichTextBox();
                                        rich.Size=new Size(300,200);
                                        rich.Location=new Point(100,10);


rich.Anchor=AnchorStyles.Top|AnchorStyles.Right|AnchorStyles.Left|AnchorStyles.Bottom;
                                        Controls.Add(rich);

                                        //menulinje:
                                        menu=new MainMenu();
                                        MenuItem help=menu.MenuItems.Add("Hjælp");
                                        help.Select+=new EventHandler(help_selected);
                                        MenuComplete+=new EventHandler(menu_valgt);
                                        help.MenuItems.Add(new MenuItem("Vis Hjælp i Browser!",new
                                        EventHandler(vis_help),Shortcut.F1));
                                        Menu=menu;

                                        //højre kliks menu:
                                        kontekst=new ContextMenu();
                                        kontekst.MenuItems.Add(new MenuItem("Vis Hjælp i Browser!",new
                                        EventHandler(vis_help),Shortcut.F1));
                                        ContextMenu=kontekst;

                                        //statuslinje:
                                        status=new StatusBar();
                                        status.Size=new Size(500,24);
                                        panel1=new StatusBarPanel();
                                        panel1.Width=350;
                                        StatusBarPanel panel2=new StatusBarPanel();
                                        panel2.Width=130;
                                        panel2.Alignment=HorizontalAlignment.Left;
                                        status.Panels.Add(panel1);
                                        status.Panels.Add(panel2);
                                        status.ShowPanels=true;
                                        panel2.Text="Brug Menuen Hjælp!";
                                        Controls.Add(status);

                                        //tooltips skal helst sættes specifikt på en kontrol:
                                        ToolTip tip1=new ToolTip();
                                        tip1.Active=true;
                                        tip1.SetToolTip(rich,"Dette er et \nrigtigt godt program, \nsom er fuld
af hjælp!\nDu skal bare \nundersøge mulighederne!");
                                        tip1.SetToolTip(this,"Du kan maximere vinduet \nfor at få en større
tekstboks!\nDette er et \nrigtigt godt program, \nsom er fuld af hjælp!");



                                }

                                //eventhandler som viser browser med hjælp:
                                private void vis_help(object obj,EventArgs a){
                                        InternetExplorerClass ie=new InternetExplorerClass();
                                        ie.Width=250;
                                        ie.Height=400;
                                        ie.Visible=true;
```

```
                                        //fjern sædvanlige browser egenskaber:
                                        ie.StatusBar=false;
                                        ie.AddressBar=false;
                                        ie.MenuBar=false;
                                        ie.ToolBar=0;
                                        object o=null;
                                        ie.Navigate("http://localhost/help.html",ref o,ref o,ref o, ref o);

                                }

                                //eks. på at tekst vises i statuslinjen når menu markeres
                                private void help_selected(object o,EventArgs a){
                                        panel1.Text="Der er megen hjælp at hente i browserens hjælpefil ...";
                                }

                                //betyder: gør dette når brugeren HAR klikket i en menu:
                                private void menu_valgt(object o,EventArgs a){
                                        panel1.Text="Klik F1 for Hjælp!";
                                }


                                public static void Main(string[] args)
                                {
                                        Application.Run(new MainForm());
                                }
                        }
}
```

Endelig er det skik og brug at brugeren kan skifte imellem en række kontroller (fx tekst bokse) på en form ved at bruge tabulator.
I C# kan en kontrol sættes på denne måde fx med tekstboks:

```
tekstbox1.TabIndex=0;
tekstbox1.TabStop=true;
```

På denne måde kan sættes en tabulator rækkefølge fra 0 og opad. Hvis en kontrol skal 'overspringes' sættes TabStop til false.

NB Tidligere er forklaret hvordan et C# program kan kalde fx en browser eller en tekstbehandling med metoden System.Diagnostics.**Process**.Start(). Denne metode kan også anvendes til Hjælp – fx ved en eventhandler på en knap.

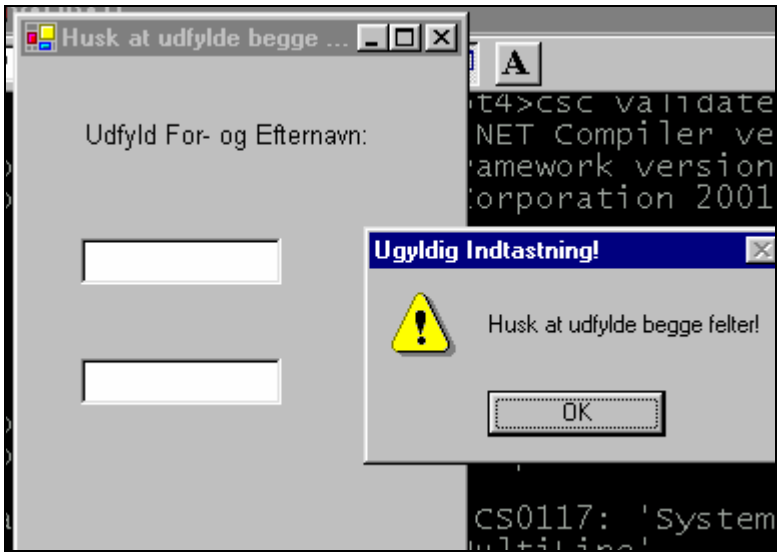
C# rummer også en klasse **ErrorProvidor** som kan bruges til kontrollere og vejlede en bruger som indtaster noget i et tekstfelt: Fx er der et '@' i email-adressen?

En endnu mere enkel metode er at bruge den **CancelEventHandler**, som er defineret i System.ComponentModel. Enhver tekst boks har en event **Validating** som kan udløse en valideringsprocedure efter denne model:

```
// eksempel paa validering af inputs:
```

```
using System;
using System.Drawing;
using System.ComponentModel;
using System.Windows.Forms;

class validate : Form {

		private TextBox fornavn, efternavn;
		private Label label;

		public validate() {

				ClientSize=new Size(220,250);
				Font=new Font("",10);

				label=new Label();
				label.AutoSize=true;
				label.Text="Udfyld For- og Efternavn:";
				label.Location=new Point(30,30);
				Controls.Add(label);

				fornavn=new TextBox();
				fornavn.Location=new Point(30,90);
				fornavn.Validating += new CancelEventHandler(valid);

				Controls.Add(fornavn);

				efternavn=new TextBox();
				efternavn.Location=new Point(30,150);
				efternavn.Validating += new CancelEventHandler(valid);

				Controls.Add(efternavn);


		}
		private void valid(object sender, CancelEventArgs e)
		{
				if (((TextBox)sender).Text == "")
				{
						string advarsel="Husk at udfylde begge felter!";
						MessageBox.Show(advarsel, "Ugyldig Indtastning!",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
						Text=advarsel;
						e.Cancel = true;
				}
   }

		public static void Main() {

				Application.Run(new validate());
		}
}
```

Når dette program kører, er det (nærmest) umuligt at få lukket det, førend der er indtastet værdier i de to tekstbokse. Programmet bliver stædigt ved med at 'cancel' en lukning!



Hvis vi forestiller os en formular med en tekst boks til indtastning af en e-mail adresse:

```
this.txtEmail.Text = "";
this.txtEmail.KeyPress += new System.Windows.Forms.KeyPressEventHandler
(this.txtEmail_KeyPress);
```

- kan vi validere den indtastede email adresse fx på denne måde med en event handler:

```
private void txtEmail_KeyPress(object sender,
System.Windows.Forms.KeyPressEventArgs e)
{
// \S+ betyder mindst et ikke white space tegn!:
System.Text.RegularExpressions.Regex regex;
regex = new System.Text.RegularExpressions.Regex(@"\S+@\S+\.\S+");

Control ctrl = (Control)sender;
if (regex.IsMatch(ctrl.Text))
{
errProvider.SetError(ctrl, "");
}
else
{
errProvider.SetError(ctrl, "Dette er IKKE nogen gyldig
e-mail adresse!");
}

}
```

Ved hjælp af såkaldt regulære udtryk (defineret i System.Text) kontrolleres det her, at e-mail adressen indeholder et '@', et '.' og et passende antal tegn! Dette eksempel anvender klassen ErrorProvider.

Opgaver:

1. Tag udgangspunkt i de metoder som du nu har set, konstruer et simpelt Windows program eller brug et du allerede har og opret en menu Hjælp med 3 forskellige slags hjælp.
2. Sæt skiftende vejledende tekster i statuslinjen (opret en statuslinje hvis der ikke er en)
3. Udform nogle passende og korte og præcise tooltips
4. Opret en genvejsmenu til hele formen eller til en bestemt kontrol
5. Sæt nogle tekstbokse på formen og kontroller hvordan du kan bruge TabIndex og TabStop.
6. Overvej det generelle: Hvordan kan man bedst sikre sig at brugeren effektivt kan bruge et bestemt program??

Interop eksempler: MS Word:

Hvis vi først har importeret COM DLL-er ind i .NET i form af interop-wrapper klasser kan vi bruge disse klasser frit i C#.

Forudsat at vi har installeret fx MS Office pakken, kan vi starte og operere med et Word program. Følgende er et eksempel herpå:

```
// illustrerer brug af interop.word.dll - MS Word:

//csc med /r:interop.word.dll;interop.office.dll <cs navn.cs>


using System;
using System.Runtime.InteropServices;

public class W_Word {

//programmet viser en fil i Word - med et kommando linje argument (en sti):

public static void Main(string[]args){

                Word.ApplicationClass word_applikation=new Word.ApplicationClass();
                word_applikation.Visible = true;
                word_applikation.Caption="Startet fra C# ...";

                //eksempel paa tilpasning af MS Word programmet:

                word_applikation.CommandBars["Standard"].Visible = false;
                word_applikation.CommandBars["Formatting"].Visible = false;
                word_applikation.CommandBars["Drawing"].Visible = false;
                word_applikation.CommandBars["Web"].Visible = false;

                                object missing = System.Reflection.Missing.Value;
                                object docfil=args[0];
                                object readOnly = false;
                                object isVisible = true;

                Word.Document dokument = word_applikation.Documents.Open(ref docfil, ref missing,ref readOnly,
ref missing, ref missing, ref missing, ref missing, ref missing, ref missing, ref missing, ref missing, ref isVisible);
                dokument.Activate();
                }
}
```

Programmet formatterer MS Word på forskellig måde og åbner en fil i Word.

Programmering i Word er – reelt – Visual Basic (VBA) kode nedenunder – og er ikke altid lige let tilgængelig! Som hjælp kan man optage makros i Word og bruge VBA koden i et C# program!

Et andet eksempel på at udnytte MS Word i C# kunne være følgende:

```
// illustrerer brug af interop.word.dll - MS Word:

//csc med /r:interop.word.dll;interop.office.dll <cs navn.cs>

using System;
using System.IO;
using System.Runtime.InteropServices;

public class W_Word {

//programmet udskriver fx en CS fil i Word - to kommando linje argumenter:

public static void Main(string[]args){

                                Word.ApplicationClass word_applikation=new Word.ApplicationClass();
                                //word_applikation.Visible = true;

                                object missing = System.Reflection.Missing.Value;
                                object docfil=args[0];
                                object readOnly = false;
                                object f=false;
                                object t=true;
                                object n=null;
                                object isVisible = true;
                                object intet="";

                                StreamReader reader=File.OpenText(args[1]);
                                string filtekst=reader.ReadToEnd();
                                reader.Close();

                                Word.Document dokument = word_applikation.Documents.Open(ref docfil, ref
missing,ref readOnly, ref missing, ref missing, ref missing, ref missing, ref missing, ref missing, ref
missing, ref isVisible);
                                dokument.Activate();

                                word_applikation.Selection.Font.Bold=1;
                                word_applikation.CommandBars["Standard"].Visible = false;
                                word_applikation.CommandBars["Formatting"].Visible = false;
                                word_applikation.CommandBars["Drawing"].Visible = false;
                                word_applikation.CommandBars["Web"].Visible = false;


                                word_applikation.Selection.TypeText(filtekst);

                                //eksempel paa gemsom sti:
                                object gemsom="C:\\csharp\\excel\\eksempel.doc";
```

```
                        dokument.SaveAs(ref gemsom,ref n,ref f,ref intet,ref t,ref intet,ref f,ref f,ref f,ref f,ref
f);
                        //dokument.Close(ref n,ref n,ref n);
                        word_applikation.Quit(ref n,ref n,ref n);


            }
}
```

Programmet kan fx indlæse en cs fil og gemme den som en Word fil. Programmering med MS Word kan være en tricky affære – metoderne tager mange ref object argumenter! - og man skal være ret omhyggelig!

På samme måde kunne man indlæse data fra **Excel** eller **Access** og udnytte dem!

Mange .NET folk mener, at det generelt **IKKE** er nogen god ide at importere COM klasser – som i dette eksempel: interop.office.dll og interop.word.dll – 'i tide og utide'! Dette bør så kun ske hvis det er den abolut **eneste** måde et problem kan løses på! Interop systemet hænger jo ikke godt sammen med ideen med .NET!

Internet og netværks programmer:

Microsoft ASP:

NB Dette afsnit forudsætter at du har installeret Microsofts Personal Web Server eller IIS på din maskine! **Alle** filer i dette afsnit skal gemmes i den mappe hvor dine local host filer ligger – dvs normalt i C:\Inetpub\wwwroot eller i en undermappe (en virtuel mappe herunder). Filerne skal kaldes i fx Internet Explorer med http://localhost/… (I nedenstående eksempler er filerne lagt i en virtuel undermappe: dotnet).

Alle Internet eller net programmer foregår som en kommunikation mellem en **klient** og en **server**. Samtalen indledes normalt ved at klienten sender en **request** (anmodning) til serveren om en service eller information.

ASP eller 'Active Server Pages' er en Microsoft teknologi til skabelse af **'dynamiske'** sider på internettet eller på et andet netværk. En almindelig HTML side er **'statisk'** dvs den kan ikke reagere på brugerens (klientens) input eller andre betingelser.

HTML ('Hyper Text MarkUp Language') er et op-mærknings sprog med **tags** eller mærker som formatterer teksten på web sider. HTML vil ikke som sådant blive gennemgået her. Du kan nemt finde forklaringer på HTML på Internettet eller andetsteds.

Du kan læse om ADO, ASP mv på http://msdn.microsoft.com.

Statisk HTML:

Et simpelt eksempel på en HTML fil gemt som 'eksempel.html' ville være:

```
<html>
<head>
<title>Eksempel.html
</title>
</head>

<body>
<h3>Indtast brugernavn og adgangskode:</h3>
<form>
<input type="text" id="bruger" name="bruger" />
<br><input type="password" id="kode" name="kode" />
<br><input type="submit" id="ok" name="ok"  value="OK - Kontroller!" />
</form>
</body>
</html>
```

Når den rigtige adresse anføres fås følgende resultat:

Dette er en **statisk** side idet siden på ingen måde reagerer fornuftigt på mine indtastninger og siden er altid den samme for alle brugere!

Som det ses bruges <> som start og slut **tags** (mærker) i HTML.

Alligevel **sker** der faktisk noget hvis jeg indtaster data i de 2 tekstbokse og klikker 'submit' (OK) knappen. I browserens **adresse** linje vises nemlig fx:

http://localhost/dotnet/eksempel.html?bruger=jensjensen&kode=123456&ok=OK+-+Kontroller%21

I dette tilfælde er indtastet 'jensjensen' og '123456' og klikket på knappen.

Forklaring:

De 3 HTML-kontroller (2 tekst bokse og en knap) er omgivet af en **form** eller en form-tag (**<form>** … </form>).
Når submit knappen klikkes gør en form altid det, at den sender værdierne af sine kontroller videre. Her sker det ved metoden **'get'** – derfor kan vi se værdierne i adresselinjen efter '?'.

I dette tilfælde har formen desværre ikke nogen steder at sende værdierne hen – så den sender dem til sig selv!

Den tekst som kommer efter '?' kaldes en **querystring** og indeholder en forespørgsel/meddelelse til siden (til **serveren,** som leverer siden).

Du kan læse alt om HTML fx på http://www.w3.org (den officielle standard organisation).

Dynamiske sider:

Vi vil nu omdøbe filen eksempel.html til eksempel.**asp** og foretage 3 små rettelser sådan:

```
<% @language= "VBScript" %>

<html>
<head>
<title>Eksempel.asp - Klokken er: <%=Time() %>
</title>
</head>

<body>
<h3>Indtast brugernavn og adgangskode:</h3>
<form id="form" name="form" method="post" action="kontroller.asp">
<input type="text" id="bruger" name="bruger" />
<br><input type="password" id="kode" name="kode" />
<br><input type="submit" id="ok" name="ok"  value="OK - Kontroller!" />

</form>

</body>
</html>
```

Første linje:

```
<% @language= "VBScript" %>
```

markerer at det som nu står i filen mellem mærkerne '<%' og '%>' **ikke** er HTML men Visual Basic **script** kode. Serveren skal altså kunne forstå det sprog som følger efter **language**!!

Det er nu muligt at gøre browserens titel linje 'dynamisk' – hvis du klikker 'Opdater' i browseren bliver klokken faktisk opdateret.

Formen har fået en attribut **'action=**"kontroller.asp"' – dvs at de indtastede data i denne form sendes til en anden ASP fil 'kontroller.asp'. Formen har fået et sted at sende sine data hen til.

Formen sender nu sine data videre med **'post'** – derved undgås at data vises i adresselinjen – hvilket ofte er meget uønskeligt (fx mht et password!).

Ofte lægges en fil som kontroller.asp i en undermappe cgi-bin. I så fald skal attributten selvfølgelig være:

<form action="/cgi-bin/kontroller.asp">

Hvis du viser siden i browseren som http://localhost/eksempel.asp og vælger Vis Kilde opdager du noget andet meget væsentligt: Vis Kilde viser **ikke** den tekst, vi har skrevet (vores koder bliver faktisk **skjult** for klienten), men den tekst, som **dynamisk** er blevet produceret af serveren (!) – i dette tilfælde ses ikke koden Time(), men det faktiske klokkeslæt:

```
<html>
<head>
<title>Eksempel.asp - Klokken er: 16:25:31
</title>
</head>
……..
```

Forskellen på HTML og ASP er altså, at ASP filer dynamisk kan producere **ny** HTML tekst, som jeg ikke har skrevet!

Hvis vi skriver en ny fil **kontroller**.asp kan vi få serveren til at 'godkende' eller 'afvise' en bruger:

```
<% @language= "VBScript" %>

<html>
<head>
<title>Svar tilbage:
</title>
</head>

<body>
<h3>Svar vedr: indtastet brugernavn og adgangskode:</h3>
<h4>
<% if request("kode") = "1234" then %>
Brugernavn og adgangskode er korrekt!
<% else %>
Brugernavn og adgangskode er IKKE korrekt!
<% end if %>
</h4>
</body>
</html>
```

Hvis du chekker Vis Kilde, vil du opdage, at klienten **ikke** kan **se** vores 'godkendelses procedure'! Klienten kan **ikke** se koden, kun resultatet af koden.

Som det ses er Visual Basic koden (markeret med fed her) ikke så forskellig fra C# kode – selv om krav til syntaks, semikolon og parenteser er mere 'løsagtige'!!

Eksempel ASP og ODBC Database (Access):

Dynamiske web sider vil ofte åbne for databaser og vise data for brugeren. Det følgende eksempel er skrevet så enkelt som muligt for at vise, hvordan ASP kan koble sig op på en Microsoft Access Database. Denne database 'personer.mdb' har en tabel 'personer' hvis indhold vises. For at dette eksempel skal kunne køre skal du altså oprette en lille database med 4 felter: nr, fornavn, efternavn og telefon.

Denne skal så registreres som 'personer' som en ODBC database (System database) i **Kontrolpanelet** i Windows. Som det ses kobler ASP sig op på 'globale' databaser – IKKE på en sti på maskinen!:

```
<% @language = VBScript %>
<html>
<head>
<title>Tabellen personer</title>
</head>
<body>
<center>

<h3>Tabellen personer i Microsoft Access databasen personer.mdb</h3><hr>
<table border=1>
<tr>
<td><b>nr</td><td><b>fornavn</td><td><b>efternavn</td><td><b>telefon</td>
</tr>

<%
dim connection,query,recordset
set connection=server.createobject("ADODB.Connection")
call connection.open("personer","","")
set session("databasesession")=connection
query="select * from personer"
set recordset=server.createobject("ADODB.Recordset")
call recordset.open(query,connection)
on error resume next
call recordset.movefirst()
```

```
do while not recordset.eof
%>

<tr>
<td><b><% =recordset("nr") %></td>
<td><% =recordset("fornavn") %></td>
<td><% =recordset("efternavn") %></b></td>
<td><% =recordset("telefon") %></td>
</tr>

<% call recordset.movenext()
loop
connection.close()
%>
</table>
</center>
</body>
</html>
```

Den vigtigste ASP kode er her markeret med fed og kan forklares sådan:

```
<%
'Erklæring af 3 objekter:

dim connection,query,recordset

' der oprettes en connection til databasen personer uden brugernavn eller kode:

set connection=server.createobject("ADODB.Connection")
call connection.open("personer","","")
set session("databasesession")=connection

'Der defineres en SQL streng som beder om alle poster i tabellen personer:
query="select * from personer"

'Der oprettes et recordset dvs et sæt af poster:

set recordset=server.createobject("ADODB.Recordset")
call recordset.open(query,connection)
'Hvis der er fejl I en post: spring den over!

on error resume next
'Gå til første post I tabellen og fortsæt til der ikke er flere:

call recordset.movefirst()
do while not recordset.eof
%>
```

Eksempel på visning af **personer**.**asp**:

Som det ses af koden bruges her også en anden Microsoft teknologi **ADO** eller **ActiveX Data Objects,** som bruges til at formidle en **connection** eller forbindelse til en database.

SQL (Structured Query Language) vil ikke blive gennemgået her – men du kan let finde informationer om SQL mange steder. Den SQL sætning som er vist i eksemplet kunne også lyde:

query="select * from personer where fornavn='Erik'"

og tabellen ville så vise alle de poster, hvor personen hedder 'Erik' til fornavn (i dette tilfælde kun én post!).

Klient side scripts:

I websider (HTML eller ASP sider) indlægges meget ofte små koder eller scripts, som ikke udføres på serveren, men hos klienten selv.

Normalt anvendes Javascript til disse koder, idet Javascript forstås af alle browsere (det gør VBScript ikke!).

Med klient scripts undgås en mængde traffik hen over nettet – så hvor det kan lade sig gøre bør man ofte anvende klient scripts og ikke server scripts!

Det efterfølgende er et eksempl på et javascript, som kontrollerer om klienten har udfyldt begge felter i en formular:

```
<!-- Eksempel paa javascript funktion som kaldes ved klik
Eksempel paa klient side script som IKKE afvikles paa serveren
Der chekkes om klienten har udfyldt begge felter
-->

<html>
<head>
<title>Eksempel Javascript.asp
```

```
</title>

<script language="javascript" id="clienteventhandlersjs">
<!--
function udfyld_kontrol(){
if(form.kode.value==""||form.bruger.value==""){
alert("OBS Begge felter (bruger og kode) skal udfyldes!");
}
}
-->
</script>
</head>

<body>
<h3>Indtast brugernavn og adgangskode:</h3>
<form id="form" name="form">
<input type="text" id="bruger" name="bruger" />
<br><input type="password" id="kode" name="kode" />
<br><input type="button" language="javascript" onclick="return udfyld_kontrol()" id="ok" name="ok"  value="OK - Kontroller!" />

</form>

</body>
</html>
```

Filen er gemt som 'java_eksempel.asp'.

Der oprettes en **<script>** blok og heri skrives en funktion som kaldes når knappen klikkes.
Som det ses er der store ligheder mellem syntaksen i C# og i Javascript (modsat Visual Basic eller VBScript).
<!-- markerer start af en kommentar og --> slut på en kommentar i HTML.

alert("") er en javascript funktion som svarer til C# MessageBox.Show("").

Microsoft ASPX eller ASP .NET:

DotNet eller .NET har bygget videre på ASP formatet og skabt en ny filtype: ASPX. Helt grundlæggende er det ASP (som vi indtil nu har beskrevet) en stor del af ASPX eller 'ASP.NET'.

ASP.NET er en revision af ASP.

For at bevise det kan vi omdøbe den sidste fil til: javaeksempel.**aspx**. Filen vil fungere fuldstændigt som før – blot vil filens ikon være forandret – idet aspx filer hos Microsoft kaldes **'webforms'**.

En simpel server i C#:

C# bygger som et Objekt Orienteret sprog på **indkapsling,** som igen betyder, at vi ikke behøver at bekymre os om de indre mekanismer i en klasse - men bare kan bruge dens 'public interface' dvs. dens **metoder** og **properties**. På den måde bliver komplicerede programmeringsopgaver gjort simple.

Vi skal her kort se på hvordan man rimeligt enkelt kan skrive serverprogrammer til netværk og Internet.

En **server** er en maskine som har en adresse på et net (fx 'www.enserver.dk' eller '192.168.0.1') og som kan kaldes op på samme måde som man gennem telefonen kan ringe til en person. Den maskine som kalder op og **'requester'** en **'service'** kaldes en **klient**.

En server har en **adresse** og en **port**. En port er en program-adresse på en bestemt maskine. Portnumrene fra 0 til og med 1023 er 'reserverede' - dvs de bør **ikke** bruges, men portnumre fra 1024 til 65.535 (!) kan frit benyttes (med mindre man er så uheldig at der i forvejen er installeret en server på maskinen med dette portnummer – men sandsynligheden er ringe!).

De følgende simple eksempler er baseret på at serveren blot installeres på 'denne maskine' – derfor kaldes den med http://localhost eller http://127.0.0.1 eller http://<maskinens navn>.

En server kører i en **evig løkke** og er – i princippet (!) - altid klar til at yde sin 'service'!

I C# kan en server implementeres fx med klassen **TcpListener** som implementerer Internet protokollen TCP. Internettet styres af **protokoller** som definerer hvad der er lovligt eller bør foregå!

Koden kan skrives ganske kort sådan:

```
// HTTPserver.cs
//DEMO af simpel TCP HTTP server:

using System;
using System.IO;
using System.Net.Sockets;
using System.Net;
```

```
class HTTPServer
{
		public static void Main(string[] args)
		{
			//En Listener som fortsætter med at lytte evigt:
			TcpListener tcp=new TcpListener(2222);
			tcp.Start();

			//NB denne server kan kun betjene een klient ad gangen:
			while(true){

				//en Socket er en forbindelse til en anden maskine:
				Socket klient=tcp.AcceptSocket();

				//er lige som en FileStream: samme metoder:
				NetworkStream net_stream=new NetworkStream(klient);

				//StreamWriter bruges også om en FileStream fx:
				StreamWriter writer=new StreamWriter(net_stream);
				//send en lille 'hjemmeside':
				string str="<html><head><title>SERVER: 2222</title></head><body
bgcolor=#669966><hr><h1> Velkommen til  <hr>Denne Server  <hr>Port 2222</h1><i><b>DETTE ER KUN EN
TEKST SERVER ...";
				writer.WriteLine(str);
				writer.Close();
				net_stream.Close();
				klient.Close();
			}


		}
}
```

Polymorfisme i OOP kan ses her ved at metoder som WriteLine() anvendes mange steder jvf også Console.WriteLine()!

Ved slutningen af hver løkke **lukkes** det hele inklusive forbindelsen til klienten – som det også fremgår af FTP og Telnet billederne nedenfor (’Ingen tilslutning til værten!’).

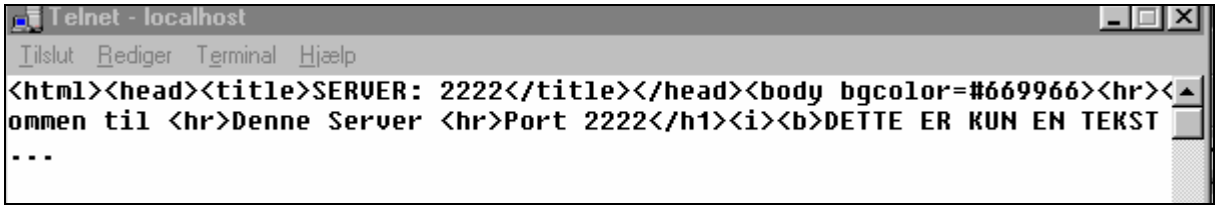
På Internettet kan en forbindelse defineres som ’Keep Alive’ – dette her er altså IKKE ’Keep Alive’!

En **Socket** er et ’stik’ eller en ’ledning’ der etableres til en fjern-computer. Begrebet har lange aner i UNIX hvor det blev opfundet i 1950-erne.

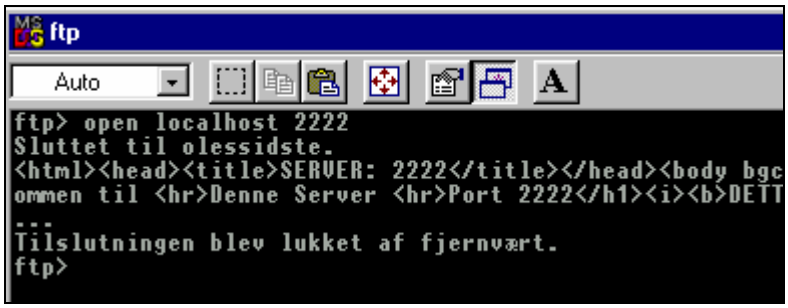
NB en **’normal’** HTTP server lytter efter en **request** fra klienten om en bestemt side eller ydelse – denne simple server sender blot uden videre en side tilbage! En normal request har som regel denne form i HTTP: **’GET /**index.html HTTP/1.1’. HTTP protokollen består af regler om hvad der kan og må siges mellem server og klient (en ’etikette’).

Når programmet er startet bliver det stående og **venter**. Hvis serveren kaldes i en **browser** med http://localhost:2222 bliver resultatet dette:

Serveren kan også kaldes via **telnet** og giver dette resultat hvis der åbnes en forbindelse til 127.0.0.1 (eller localhost eller computerens navn) port 2222:



Den kan også kaldes via **ftp** (ftp.exe) med open localhost 2222 sådan:



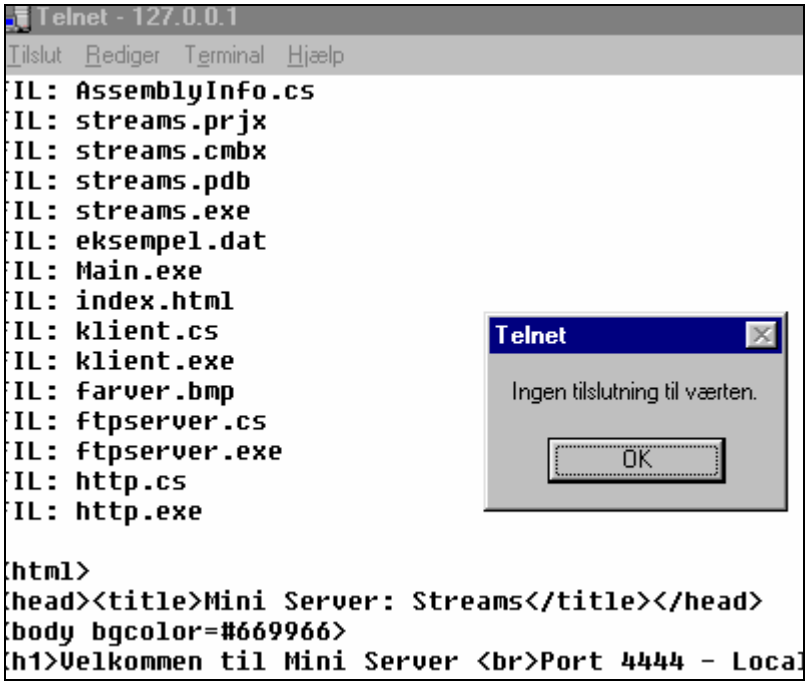
Et andet eksempel på hvad en server – en slags **FTP** server (port er her lagt på 2121) kunne gøre er følgende hvor serveren udskriver en liste over de **filer** som ligger i brugerens mappe og til sidste udskriver filen index.html:

Kun en del af koden er med her – resten er identisk med ovenstående:

```
                                        writer.WriteLine("Velkommen til FTP paa Localhost
2121");
```

```
writer.WriteLine();
DirectoryInfo dir=new DirectoryInfo(".");
FileInfo[] filer=dir.GetFiles();
foreach(FileInfo fil in filer){
                writer.WriteLine("FIL: "+fil.Name);
}
writer.WriteLine();
string req="index.html";
StreamReader fra_fil=File.OpenText(req);
string f=fra_fil.ReadToEnd();
writer.WriteLine(f);
fra_fil.Close();
```

I **telnet** bliver resultatet:



Opgaver:

1. Skriv en server som leverer en informationstekst til klienten ved opkobling. Teksten læses fra harddisken.
2. Skriv en server som leverer dato og klokkeslet ved opkobling (brug DateTime.Now.ToString()).

En simpel netværks klient i C#:

Endeligt vil vi se på et lille eksempel på en netværks klient, som kan kalde op til localhost eller en hvilken som helst anden server på Internettet.

I dette eksempel er valgt at klienten startes med to kommando linje parametre: en host (server) og den side som ønskes for eksempel således:

klient csharpkursus.subnet.dk kursus1.html

I stedet kan der også kaldes op således med en IP adresse (4 tal med punktum imellem):

klient 127.0.0.1 index.html

Her kaldes op til localhost (din egen server).

Programmet kan let udbygges.

En klient kan i C# implementeres som en **TcpClient**:

```
//Eksempel på HTTP klient:

//EKS: klient csharpkursus.subnet.dk ny_boghandel.xml eller kursus1.html

using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.IO;

public class Klient
{

        public static void Main(string[] args)
        {
        string internet_adresse=null;
        TcpClient tcp;
        byte[] svar_fra_server;
        byte[] request_til_server;

        tcp=new TcpClient();
        internet_adresse=args[0];

        //Connect:
        tcp.Connect(internet_adresse,80);
        NetworkStream net=tcp.GetStream();
        request_til_server=new byte[1024];

        string side=args[1];
        //korrekte opkald ifl HTTP protokollen:
        string req="GET /"+side+" HTTP/1.1\nHost: "+args[0]+"\n\n";
        request_til_server=Encoding.ASCII.GetBytes(req);

        //send request
        net.Write(request_til_server,0,request_til_server.Length);

        //modtag respons:
        svar_fra_server=new byte[4096];
        net.Read(svar_fra_server,0,svar_fra_server.Length);
        string svar=Encoding.ASCII.GetString(svar_fra_server);
```

```
                Console.WriteLine(svar);
                }
}
```

NB Der er **ikke** skrevet **exception** handling til denne kode – for at gøre den kort – derfor kan programmet resultere i system-fejl!

Klienten connecter til en server og en port. Porten er her kodet fast ind (port 80 er HTTP Internet porten til HTML sider), men dette kan ændres.

Hvis en klient kalder op til en HTTP server **skal** opkaldet have et bestemt format som det fremgår! Protokollen HTTP handler simpelt hen om disse 'adfærdsregler'. Klienten kan sende en række **headers** til serveren. Du kan se eksempler på disse når du bruger det efterfølgende program – en server i en ny udgave. Se nedenstående billede!

Du kan læse om alle disse regler mange steder på nettet ved at søge på HTTP protokollen - bl.a. på http://w3c.org.

Programmet opretter en NetworkStream der er en Stream som alle andre Streams. **Alle** Streams (FileStreams, MemoryStreams, NetworkStreams osv) har en metode Write som tager en byte tabel som parameter.

Alle **Streams** har også en anden metode: Read();

Jvf også **Console**.Read() og Write()! (Dette er et eksempel på OO **Polymorfisme**!).

Her er valgt simpelt hen at bruge denne metode (selv man også kunne bruge en StreamWriter).

For at oversætte fra string til byte tabel bruges metoder i: **System.Text Encoding.ASCII**: nemlig metoden GetBytes() og GetString().

Da det er nemmere at arbejde med en streng, laver programmet disse konverteringer.

At oversætte en tekst til en byte tabel **kan** strengt taget godt ske således:

```
string str = "Her kunne der have vaeret en lang tekst...";
   byte[] buf = new byte[str.Length];
   for(i=0; i<str.Length; ++i)
   {
     buf[i] = (byte)str[i];
   }
```

Men metoderne i System.Text giver flere muligheder og er at foretrække!

Konverteringen kan også foregå på denne alternative måde:

ASCIIEncoding encoder = new ASCIIEncoding();

string str = "q w e r t y u i o p";

```
byte[] buf = new byte[encoder.GetByteCount(str)];
int j = encoder.GetBytes(str, 0, str.Length, buf, 0);

int length = encoder.GetCharCount(buf, 0, buf.Length);
char[] charArray = new char[length];

encoder.GetChars(buf, 0, buf.Length, charArray, 0);
Console.WriteLine(charArray);
```

Klassen **ASCIIEncoding** har to metoder GetByteCount() og GetCharCount() som bruges til at finde antal elementer i de to arrays. I stedet for GetString() er her anvendt GetChars().

Unicode:

I stedet for ASCII kan – ud fra et europæisk synspunkt – måske mere passende anvendes en Unicode kodning. Unicode anvender 2 bytes (16 bits) til hvert tegn:

```
using System;
using System.Text;

public class UnicodeDemo
{
  public static void Main()
  {

    UnicodeEncoding encoder = new UnicodeEncoding();
    String str = "De løvfældende træer når helt ud i søen!";   //40 tildels danske tegn!

    byte[] buf = new byte[encoder.GetByteCount(str)];  //80 bytes!
    int j = encoder.GetBytes(str, 0, str.Length, buf, 0);

    for(int i=0; i<buf.Length; ++i)
    {
      Console.WriteLine("\tByte nr {0} er {1,2:X}",i,buf[i]);
    }
    Console.WriteLine();
    String str2 = encoder.GetString(buf);
    Console.WriteLine(str2);
  }
}
```

Metoderne her ved at et tegn fylder 2 bytes! Derfor læses altid 2 bytes ad gangen. Alle de danske tegn ligger inden for en byte altså i området 0 til 255 eller hexadecimalt fra 0 til FF:

På adressen http://csharpkursus.subnet.dk ligger en C# fil **DNS** som kan kalde op til en server på Internettet og udskrive en række data om serveren ved hjælp af Domain Name Service (**DNS**) som er den tjeneste der 'oversætter' en adresse som fx 'msdn.microsoft.com' til et binært IP adresse tal.

På samme adresse ligger en ret fuldstændig kode til en **FTP** klient – nem at bruge og ret nem at tilpasse til konkrete behov! (Denne kode er downloadet fra http://c-sharpcorner.com).

Opgaver:

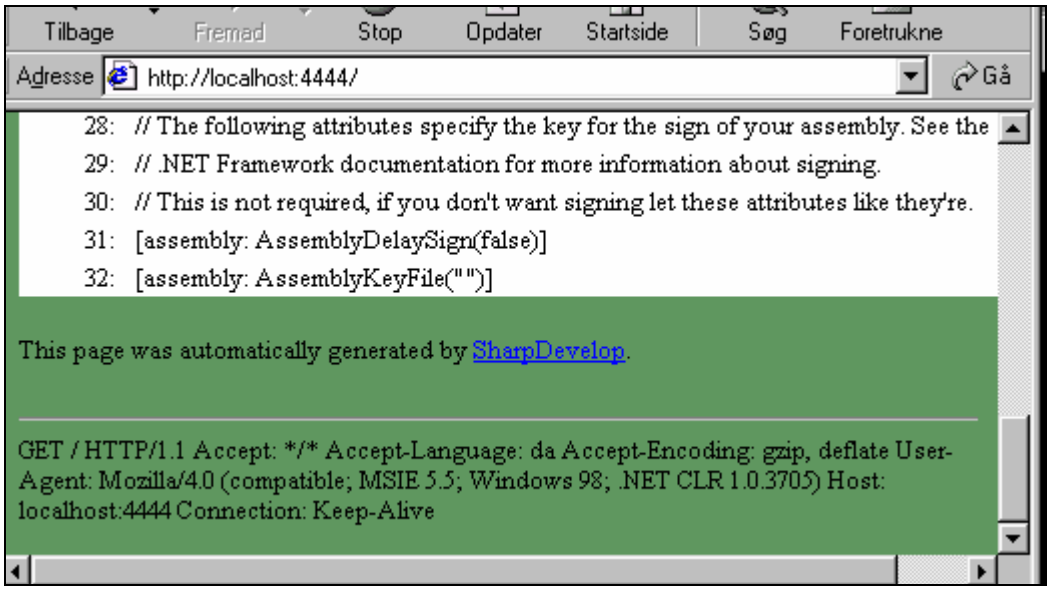
1. Omskriv programmet så det i stedet for en 'GET' sender en 'HEAD' – hvad er resultatet?
2. Omskriv programmet så det kan kalde op til forskellige porte.
3. Lav et Windows program som indeholder en tekstboks der viser svaret fra serveren
4. Lav et Windows program hvor bruger kan indtaste en server og side og port og få svar!

En anden server:

Vi kan nu **omskrive** vores server med de metoder der er anvendt i klassen Klient. I mange tilfælde fungerer serveren mere **effektivt** sådan. Følgende er et eksempel her på – programmet er forklaret i koden:

```
//HTTP Server på port 4444 sender index.html plus klientens egen request!
//Serveren sender uden at undersoege hvad klienten requester!

using System;
using System.IO;
using System.Net.Sockets;
using System.Text;
using System.Net;

public class HTTPServer
{
                public static void Main(string[] args)
```

```
		{
			//indledning og titel til hjemmesiden:
			string hjemmeside="<head><title>HTTPServer 4444</title></head><body
bgcolor=#669966>";
			StreamReader reader=null;

			//find index.html:
			try{
				reader=File.OpenText("index.html");
				hjemmeside+=reader.ReadToEnd();
			}
			//NB finally uden catch er OK:
			finally{
				reader.Close();
			}
			//vandret linje i HTML filen:
			hjemmeside+="<hr>";

			TcpListener tcp=new TcpListener(4444);
			tcp.Start();
			Socket s;
			byte[] svar;
			byte[] request;

			//evig loop:
			while(true){

			s=tcp.AcceptSocket();
			svar=new byte[2048];
			request=new byte[1024];

			//request modtages - men sendes blot retur - som et eksempel!
			s.Receive(request);

			//send index.html + formattering + request:
			svar=Encoding.ASCII.GetBytes(hjemmeside);
			s.Send(svar);
			s.Send(request);

			//luk forbindelsen til klienten:
			s.Close();

			}

		}
}
```



Hvis programmet skulle være mere korrekt skulle serveren sende visse **headers** inden den sender selve svaret tilbage til klienten. Hvis vi forudsætter at alt er OK skal serveren som det allerførste sende minimum disse linjer til klienten:

HTTP/1.1 200 OK
Content-Type: text/html

og derefter 2 tomme linjer. (Dette er reglerne i HTTP protokollen).

Opgaver:

1. Omskriv denne server så den indledende sender disse headers!
2. Omskriv serveren så den checker om de 3 første tegn i requesten er 'GET'!

IEEXEC:

Når .Net installeres, installeres også en fil **ieexec**.exe i .NET mappen (sammen med de øvrige .NET filer som csc.exe). Dette har den konsekvens at du kan lægge exe filer på din server (på localhost), sætte et **link** til exe filen og starte programmet direkte i browseren!!

På denne måde kan du lade besøgende på din hjemmeside starte et **hvilket** som helst C# program – også **Windows** programmer!

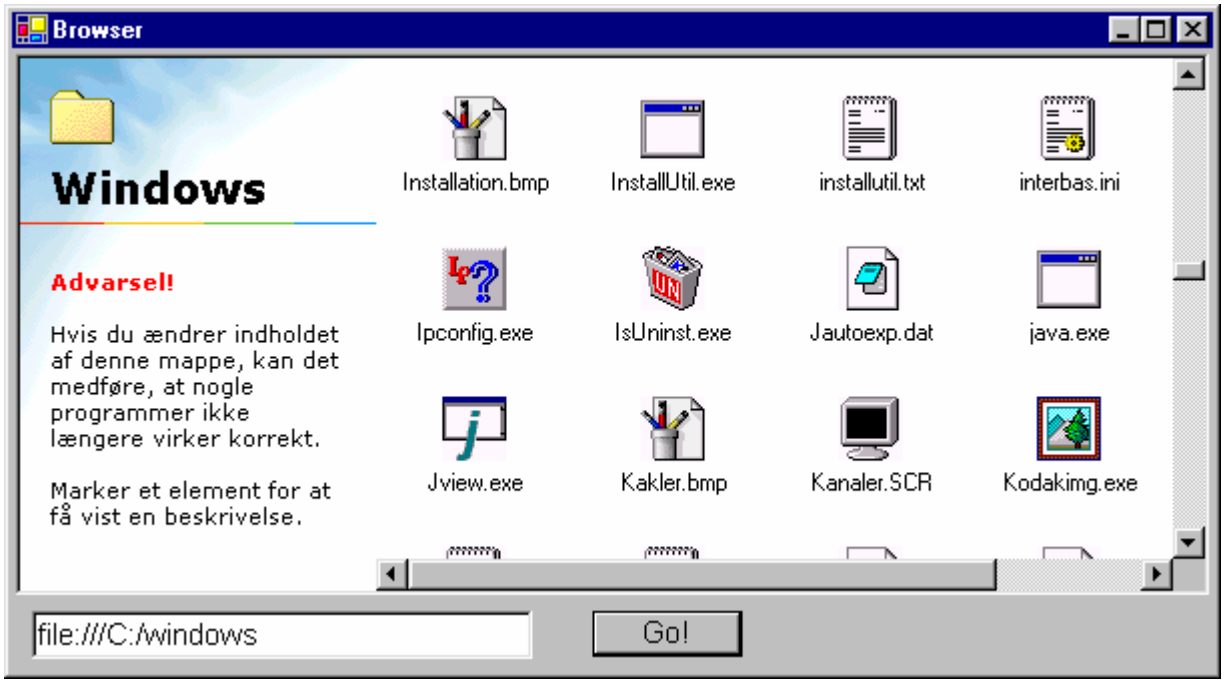
Dette kan selvfølgeligt **kun** lade sig gøre fordi du har installeret DotNet. Hvis du har din website liggende på en fremmed server kan dette trick ikke lade sig gøre!

Du kan på denne måde skrive såkaldte **CGI** programmer - vel at mærke i C# og **ikke** i fx ASP eller Perl, som også er meget brugt - som kan køre i og udføres på serveren. De C# sætninger du skriver som **Console**.**WriteLine**() bliver **IKKE** sendt til skærmen – men til den **klient** som kalder op til

serveren!! Dette er grundlæggende hvad **ieexec**.exe sørger for. Hvis man har installeret en Microsoft server, er dette måske den **nemmeste** form for serverprogrammering.

Brug af ActiveX WebBrowser i C# applikation:

Det er også muligt – ved at importere ActiveX komponenten WebBrowser – at skrive C# applikationer med en browser komponent fx således:



Man skal her referere til **AxInterop**.ShDocVw.dll i et projekt. Denne web browser er en COM komponent med mange muligheder. På http://csharpkursus.subnet.dk ligger et lille eksempel herpå. Her ligger også et eksemplar af AxInterop.ShDocVw.dll.

Sammenfatning af databaser og netværksprogrammer:

Vi har nu set på både anvendelsen af databaser og af netværksrogrammer i C#. Disse to emner kan **sammenfattes** idet vi nu kan skrive vores egen enkle server i C# som så kan hente og vise data fra en database.
Denne database kan være af enhver slags og **fordi** programmet kodes i C# (og ikke fx i VbScript eller ASP) kan vi frit bruge alle .NET og C# klasser og metoder i serveren.
På denne måde er det muligt at levere internet og netværksprogrammer 'hvor kun fantasien sætter grænser'.

Følgende eksempel viser hvordan man kan skrive en klasse som producerer en HTML side ud fra en database, en tabel og en SQL sætning. Vi genbruger koden fra det tidligere eksempel med en DataReader til at læse databasen.

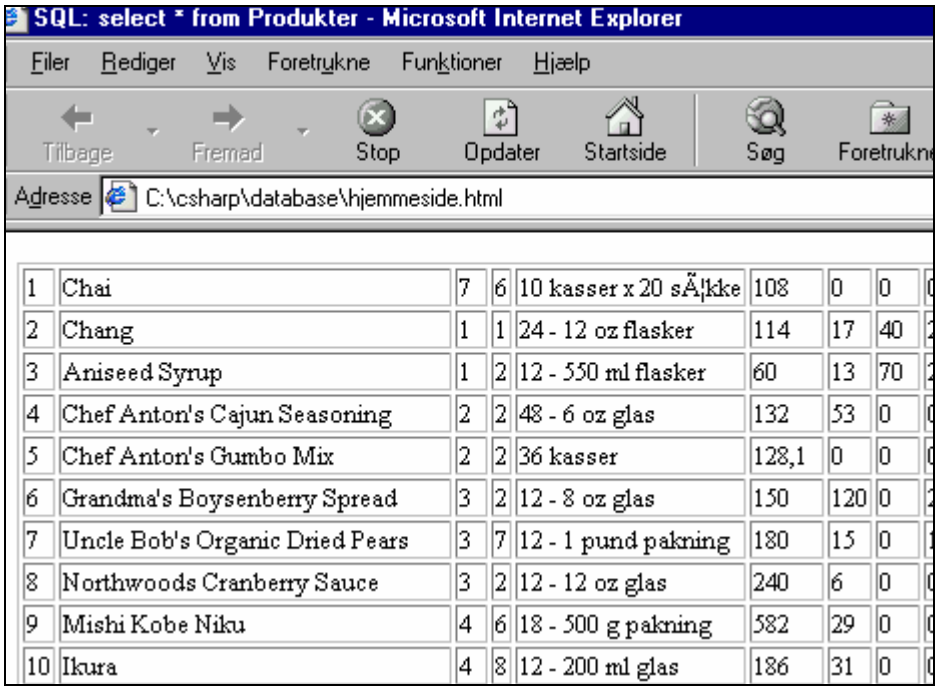
Klassen **HTMLData** kan så instantieres i et **serverprogram** og levere den side, som skal sendes tilbage til klienten – ud fra klientens anmodning:

```
// datareaderhtml.cs producerer en HTML side fra database:

//DataReader: er velegnet til blot at vise indhold i database som ikke skal opdateres

//viser data fra tabel i HTML tabel:



        using System;
        using System.IO;
        using System.Data;
        using System.Data.OleDb;//fx Access
        using System.Xml;

        public class HTMLData

        {
                private OleDbConnection cn;
                private static OleDbDataReader reader;
                private string sql;


                public HTMLData(string database,string sqlrequest){

                        sql=sqlrequest;

                        //DB connection: Kald op til Access databasen:
                        cn=new OleDbConnection();
                        cn.ConnectionString="provider = Microsoft.JET.OLEDB.4.0;data
source="+database;
                        cn.Open();

                        //SQL kommando: vælg alle kolonner i tabellen bog

                        OleDbCommand kommando=new OleDbCommand(sql,cn);

                        reader=kommando.ExecuteReader();
                        cn.Close();
                }

                //event handler som viser næste post:
                public          string producer_html(){

                        //er der flere poster: Read() returnerer true eller false:

                        string htmlside="<html><head><title>SQL:
"+sql+"</title><body><table border=1>";
                        while(reader.Read()){
                                htmlside+="<tr>";

                                for(int i=0;i<reader.FieldCount;i++){
```

```
                                                htmlside+= "<td>"
+reader[i].ToString() + "</td>";
                                        }
                                        htmlside+="</tr>";
                                }
                                htmlside+="</table></body></html>";
                                reader.Close();
                                return htmlside;


                        }
                }

                class app{


                        public static void Main(string[] args)
                        {
                                HTMLData html=new HTMLData ("c:\\dokumenter\\Northwind.mdb",
"select * from Produkter");
                                string hjemmeside=html.producer_html();

                                //kun til kontrol:
                                StreamWriter writer=File.CreateText("hjemmeside.html");
                                writer.Write(hjemmeside);
                                writer.Close();
                        }
                }
```

Klassen kaldes med 2 parametre og metoden producer_html() returnerer en HTML side. Her er valgt en tabel fra **Northwind**.mdb som leveres sammen med Windows.

Resultatet er måske ikke kønt, men effektivt:



1	Chai	7	6	10 kasser x 20 sÃ¦kke	108	0	0	
2	Chang	1	1	24 - 12 oz flasker	114	17	40	
3	Aniseed Syrup	1	2	12 - 550 ml flasker	60	13	70	
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz glas	132	53	0	
5	Chef Anton's Gumbo Mix	2	2	36 kasser	128,1	0	0	
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz glas	150	120	0	
7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 pund pakning	180	15	0	
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz glas	240	6	0	
9	Mishi Kobe Niku	4	6	18 - 500 g pakning	582	29	0	
10	Ikura	4	8	12 - 200 ml glas	186	31	0	

Remoting i Csharp:

System.Runtime.Remoting indeholder en samling klasser som kan bruges til at implementere 'RPC' (Remote Procedure Call' - som det for mange år siden blev opfundet i UNIX) eller 'Remoting'.

Herved forstås at en maskine fx i Hongkong kan bruge objekter og metoder på en maskine i New York – **SOM OM** de var 'lokale' eller befandt sig på den lokale maskine!

Remoting/RPC er så at sige **skjult** for brugeren!

Remoting kan gennemføres på flere forskellige måder. Vi vil her først se på nogle grundlæggende eksempler:

Hvis vi starter med **klienten** (som ønsker at kalde objekter/metoder på en fjern-komputer) kan vi forklare processen således:

Klient:

Kode til Klient.cs ser f.eks. sådan ud som **minimum**:

```
//RPC Remoting klient:

using System;
using System.Runtime.Remoting;

namespace olenyborg.rpc
{
   public class Klient
                    {
```

```
    private delegate String GDelegate();

    private static string svar;


                                public static void Main(string[] args)
                                {
                                        RemotingConfiguration.Configure("Klient.exe.config");

                                        RPC rpc=new RPC();
                                        if (rpc == null)
                                        {
                                                Console.WriteLine("Ingen forbindelse med
serveren!");
                                                return;
        }
                                //a-synkrone metoder ved kald til remoting serveren:

        GDelegate delegat = new GDelegate(rpc.data_retur);
        IAsyncResult ar = delegat.BeginInvoke(null, null);
        // ... evt anden kode, andre opgaver ...
        ar.AsyncWaitHandle.WaitOne();
        if (ar.IsCompleted)
        {
          svar = delegat.EndInvoke(ar);
        }
        Console.WriteLine(svar);
                                }
                }
}
```

Grundlæggende er denne kode **nøjagtig** lige som megen anden – ikke Remoting kode!

I klassen Klient instantieres en klasse RPC og metoden data_retur() kaldes på det nye objekt.

Her er anvendt delegater – men det er **IKKE** nødvendigt og **ikke** afgørende for selve strukturen i programmet. Men delegaten muliggør en 'threadet' implementering – hensigtsmæssigt fordi en Remoting KUNNE tage tid at etablere!!

I denne kode kan vi **ikke** se at RPC er en klasse på en fremmed server. Vi kan ikke se at det der reelt sker er at en klient i måske Hongkong henter et objekt i New York!! Jvf dog formlen if (rpc==null) som vi er nødt til at have hvis denne RPC ikke lykkes! Linjen:

```
                                        RemotingConfiguration.Configure("Klient.exe.config");
```

viser dog at Klient tilhører Remoting og har en Remoting config fil (XML fil). Configure() **betyder: Gå ud i XML filen og hent klientens data og egenskaber!**

NB XML filen SKAL **hedde exe navnet + ".config" altså her "Klient.exe.config" (!!):**

```
<configuration>
```

```
  <system.runtime.remoting>
    <application name="Klient">

      <client url="tcp:/localhost:3333/XYZ_Server">

        <wellknown type="olenyborg.rpc.RPC, rpc"
                   url="tcp://localhost:3333/XYZ_Server/Service" />

      </client>

      <channels>
        <channel ref="tcp" displayName="TCP Channel (Klient)" />
      </channels>

    </application>
  </system.runtime.remoting>
</configuration>
```

Config filen indeholder de nødvendige data:

1. Applikationen skal have et **navn** – vi KUNNE godt her have valgt noget andet end 'Klient' fx 'Remoting Klient' - hvis vi synes det er mere passende
2. Den fremmede server skal identificeres med **protokol** (tcp eller http osv), **adresse** og en **port**!! I dette Remoting system SKAL klienten altså kende disse data og også kende serverens/programmets **navn** her: **XYZ_Server**!
3. type beskriver dels den type/klasse som kaldes på den fremmede computer (her er det namespace + **RPC**) dels angives navnet på den **assembly** hvori typen/klassen findes - her **rpc** (findes i rpc.dll nemlig).
4. Desuden erklærer klienten nogle **kanaler** – her kun én: nemlig tcp – http ville også være en oplagt mulighed.

Alt i alt definerer config filen altså de nødvendige data for at kunne koble op på en fremmed maskine.

Her er valgt – som demo - at serveren er //localhost/, men serveren kunne selvfølgelig være en hvilken som helst server!!

Klientens rpc (DLL) stub:

Hvis vi nu prøver at **kompilere** vores Klient.cs får vi en hel del compiler fejl, fordi Klient.cs **refererer** til klasser, som compileren ikke kan finde!

Klient.cs skal nemlig have en **'stub'** (eller **skelet**) - dvs en 'mini udgave' af serverens program for at kunne køre. Klienten skal kende serverens klasser, metoder osv. Vi forsyner derfor vores klient med følgende stub-kode som kompileres som rpc.dll:

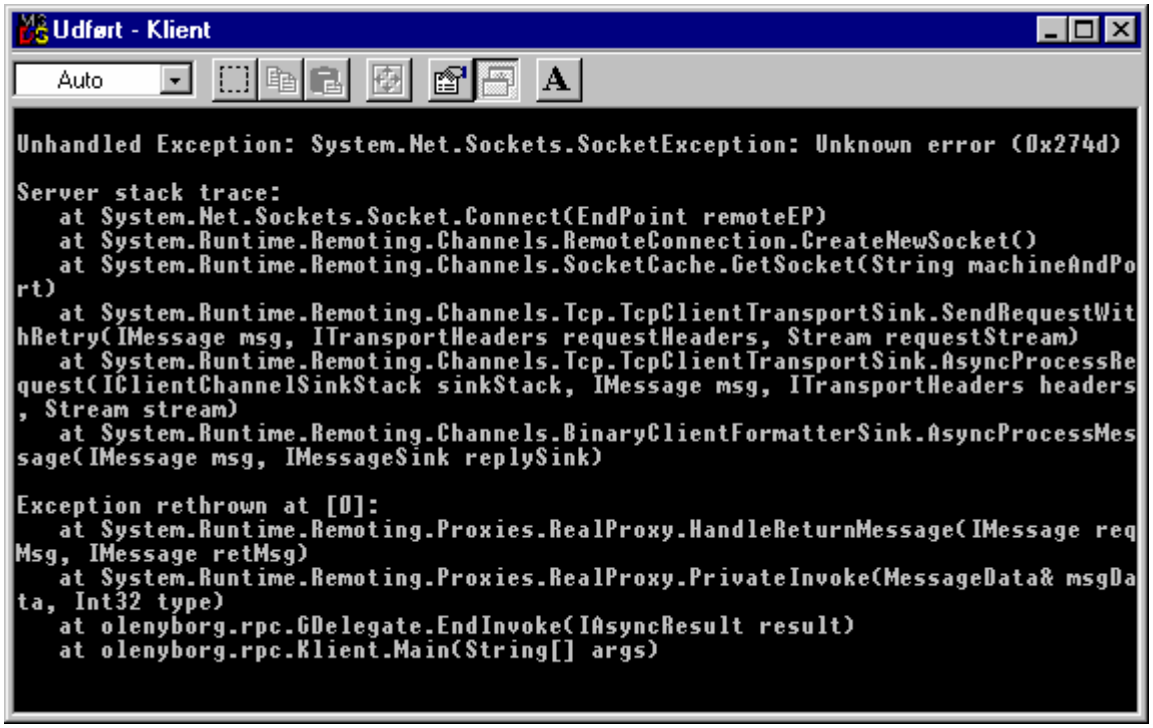
```
using System;
using System.Runtime.Remoting.Messaging;

namespace olenyborg.rpc
{
		public class RPC : System.MarshalByRefObject
		{
				public RPC()
				{
				}

				~RPC()
				{
				}

				//SERVICES:

				public string data_retur()
				{
						return "";
				}

		}
}
```

Som det ses er der her tale om formelle data som ikke er reelt implementeret!

Vi kan nu kompilere vores klient med en reference til denne stub DLL!

Vi har nu en velfungerende klient – men da vi er gået bagæns frem ! – mangler vi det væsentlige selve serveren!

Hvis vi prøver at starte Klient fås dette resultat:



Her kan vi se alle de metoder, som er mislykkedes – idet serveren jo slet ikke findes i dette tilfælde!

Vi kan se at runtime systemet prøver at etablere en **socket** forbindelse til en IP adresse - ved hjælp af TCP protokollen osv.

Serverens Service:

Vi kan fx på en enkel måde implementere serverens metode data_retur() således:

```
//RPC Remoting eksempel:

using System;
using System.Runtime.Remoting.Messaging;

namespace olenyborg.rpc
{
		public class RPC : System.MarshalByRefObject
		{
				public RPC()
				{
						Console.WriteLine("Constructor!");
				}

				~RPC()
				{
						Console.WriteLine("Destructor!");
				}

				//SERVICES:
```

```
                public string data_retur()
                {
                string data=@" <katalog>

                <bog>
                <forfatter>jens Jensen</forfatter>
                <titel>der var en gang en soldat</titel>
                <pris>213,50</pris>
                <aar>2001</aar>
                <beskrivelse>rigtig god</beskrivelse>
                </bog>


                <bog>
                <forfatter>hans Jensen</forfatter>
                <titel>der var en gang en pige</titel>
                <pris>233,50</pris>
                <aar>2002</aar>
                <beskrivelse>rigtig ringe</beskrivelse>
                </bog>
                <bog>
                <forfatter>ulla Jensen</forfatter>
                <titel>der var en gang en rigtig mand</titel>
                <pris>113,50</pris>
                <aar>2003</aar>
                <beskrivelse>rigtig interessant</beskrivelse>
                </bog>
        </katalog>";

                return data;
                }

        }
}
```

Klassen RPC har kun en metode eller service nemlig data_retur() som her returnerer en XML fil med en bog samling! Ved at bruge @ foran strengen kan vi sende strengen 'formatteret' dvs inklusive linje skift osv!

Dette er naturligvis et meget enkelt eksempel, men tilstrækkeligt til at vise at systemet fungerer. Klassen RPC kunne implementere en hvilken som helst metode! Prøv bare!

Serveren:

Den nødvendige kode til Remoting serveren er nærmest ufattelig **kortfattet** – idet de nødvendige data indlæses fra en config fil, som vi også så det med klienten:

```
//Remoting server RPC til rpc.dll

//kompileres UDEN ref til rpc.dll!!

using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Contexts;
```

```
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
using System.Runtime.Remoting.Channels.Tcp;

namespace olenyborg.rpc
{
		class Server
		{
			public static void Main(string[] args)
			{
				//FORDI vi bruger System.remoting!!:

				//hent: kanaler, porte, klasser, dll'er, service definitioner:
				RemotingConfiguration.Configure("Server.exe.config");


				Console.WriteLine("Application: " +
RemotingConfiguration.ApplicationName);


				System.Console.WriteLine("\n... Tast Enter for STOP!");
				System.Console.ReadLine();
			}
	}
 }
```

Server.cs har egentlig **ikke** noget at gøre med rpc.dll og der skal **IKKE** refereres til rpc.dll når Server.cs skal kompileres!

Den tilhørende Server.exe.config ser sådan ud:

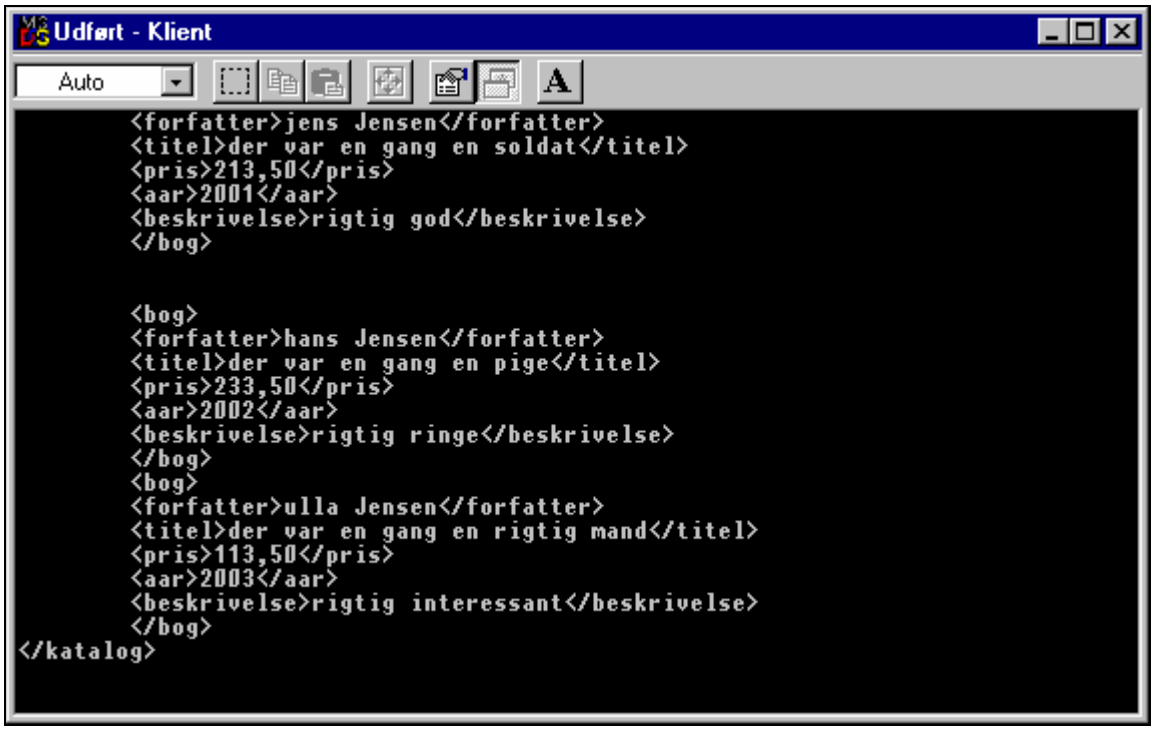
```
<configuration>
  <system.runtime.remoting>

    <application name="XYZ_Server">

      <service>
        <wellknown mode="SingleCall" type="olenyborg.rpc.RPC, rpc"
                   objectUri="Service" />
     </service>
      <channels>
        <channel ref="tcp" port="3333" displayName="TCP Channel (XYZ_Server)" />
        <channel ref="http" port="4444" displayName="HTTP Channel (XYZ_Server)" />
      </channels>
    </application>
  </system.runtime.remoting>
</configuration>
```

Som det ses ligner serverens og klientens config filer hinanden og skal selvfølgelig have data svarende til hinanden!!

Serveren kan altså nås dels via tcp dels via http **protokollen** – på 2 forskellige **porte**.

Vi kan nu kompilere serveren og starte den som exe fil. Serveren **åbner** kanalen således at klienter kan oprette et 'stik' eller 'socket' til den nævnte service. Som det kan ses er serveren **PASSIV** – det er **IKKE** serveren som arbejder!!

De fremmede klienter kalder SELV op til klassen RPC eller til applikationen 'XYZ_Server' og klienten gør selv hele arbejdet.



På den måde kan klienten få det **indtryk,** at han arbejder med en klasse der findes lokalt på hans egen maskine – hvilket er hele ideen med Remoting eller RPC!

Remoting bruges fx i forbindelse med **distribuerede** systemer. Omfattende og avanceret data behandling kan på denne måde fordeles over et stort antal maskiner, som samarbejder uden en fælles memory! Samarbejdet foregår udelukkende via en protokol og via Remoting/RPC.

Remoting kan ikke uden videre sammenlignes med andre systemer på netværk – fordi Remoting har et specifikt **formål** som omtalt! Men Remoting systemet kan selvfølgeligt bruges til 'hvad som helst' – **hvis** man vil. Alle mulige funktioner kan formidles via RPC eller Remoting! Almindelige server funktioner kan altså også køres via Remoting!

Vi kan skrive følgende metode og anvende den i klassen Server som et eksempel på hvordan det er muligt for en klasse som Server at aflæse hvordan config filen er defineret. Denne metode udskriver data fra Server.exe.config:

```
public static void ShowWellKnownServiceTypes()
{
```

```
                                        WellKnownServiceTypeEntry[] entries =
RemotingConfiguration.GetRegisteredWellKnownServiceTypes();

                                        foreach (WellKnownServiceTypeEntry entry in entries)
                                        {
                                                Console.WriteLine("\nAssembly: " +
entry.AssemblyName);
                                                Console.WriteLine("Mode: " + entry.Mode);
                                                Console.WriteLine("URI: " + entry.ObjectUri);
                                                Console.WriteLine("Type: " + entry.TypeName);

                                                IContextAttribute[] attributes =
entry.ContextAttributes;
                                                if (attributes != null)
                                                {
                                                        foreach (ContextAttribute attribute in
attributes)
                                                        {

        Console.WriteLine("Context attribute: " + attribute.Name);
                                                        }
                                                }
                                        }
                        }
```

Remoting uden config filer:

Vi kan opnå en tilsvarende virkning uden overhovedet at bruge config filer! I så fald skal data hard-kodes ind i selve cs koden.

F.eks. kan vi oprette en remoting server således:

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

public class Server
{
  public static void Main()
  {
    TcpServerChannel channel = new TcpServerChannel(3333);
    ChannelServices.RegisterChannel(channel);
    RemotingConfiguration.RegisterWellKnownServiceType(
                typeof(olenyborg.rpc.RPC), "Service", WellKnownObjectMode.SingleCall);

    Console.ReadLine();
  }
}
```

På samme måde kan en klient skrives på denne måde:

```
using System;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

public class Klient
{
  public static void Main()
  {
    ChannelServices.RegisterChannel(new TcpClientChannel());
    olenyborg.rpc.Rpc obj = (olenyborg.rpc.Rpc)Activator.GetObject(
                typeof(olenyborg.rpc.Rpc), "tcp://localhost:3333/Service");

    if(obj == null)
    {
      Console.WriteLine("Kan ikke forbinde til serveren!");
      return;
    }

    Console.WriteLine(obj.data_retur());
  }
}
```

Code Access Security (CAS):

Når .NET er installeret på en udgave af Windows eksisterer der grundlæggende 2 slags security eller sikkerheds systemer:

1. Windows har sit eget sikkerhedssystem som kan bestå af logins, brugerkonti og brugergrupper som tildeles bestemte rettigheder og en fil baseret security som definerer hvem der kan læse, skrive til eller eksekvere denne fil (Sml systemet i UNIX, LINUX, NT)
2. Oven på dette har .NET sin eget security system som dels er 'rolle baseret' (brugere, brugergrupper) dels er 'Code Access Security' hvor det enkelte program definerer regler for adgang og anvendelse. Når .NET installeres defineres en standard security som man kan studere i filer som machine.config (XML filer).

Der er en lang række forskellige grunde til at spørgsmål om sikkerheden er væsentlige:

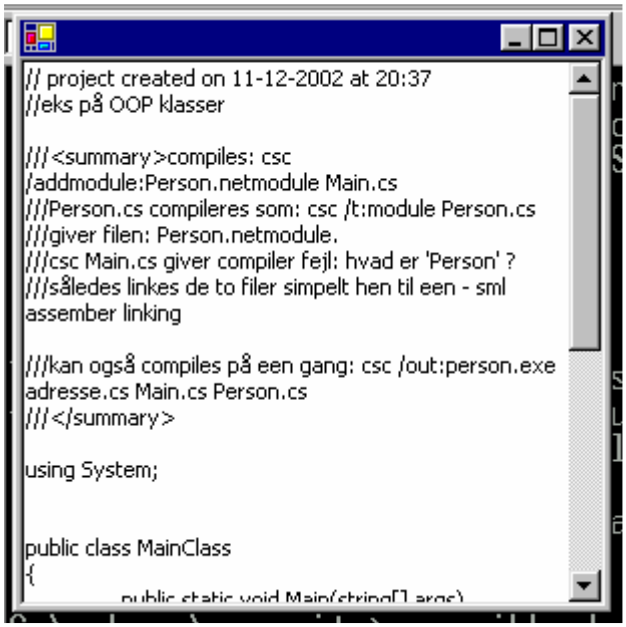
1. Jeg ønsker at skrive sikre programmer som ikke ved et uheld kan komme til at ødelægge min egen maskine
2. Programmer der ligger på min maskine kan måske kaldes af andre på et net – og jeg ønsker ikke at fremmede skal kunne hacke sig ind på min maskine eller forårsage ulykker
3. Programmer jeg downloader er måske ikke helt pålidelige og jeg ønsker derfor at sikre mig at de ikke kan forårsage skader

Begrebet 'sandkasse programmer' blev opfundet om fx Java Applets eller programmer i JavaScript. Disse kan tit fungere som rigtige programmer men de kan ikke skrive eller læse fra filer dvs de har ingen 'access' til min filsystem efter at jeg har downloadet dem. De er altså 'sikre' at køre. Lignende effekt kan man få ved at anvende forskellige slags CAS i .NET – men her kan man meget mere præcist defineret hvad der er 'OK' og hvad der er 'forbudt'!

Det grundlæggende begreb er her 'Permission'. En assembly/program tildeles eller fratages eller udkræves en bestemt Permission for at det kan starte og køre.

Et lille eksempel på dette er følgende program:

```
// CAS eksempel Code Access Security:

using System.Windows.Forms;
using System;
using System.IO;
using System.Security;
using System.Security.Permissions;



public class S : Form {

		private static RichTextBox box;
		private static StreamReader reader;

		public S(string s){

				box=new RichTextBox();
				box.Dock=DockStyle.Fill;
				Controls.Add(box);
				get_fil(s);

		}


		private static void get_fil(string s){
				reader=File.OpenText(s);
				string fil=reader.ReadToEnd();
				box.Text=fil;


		}
		public static void Main(string[] args){
				S s=new S(args[0]);
				s.ShowDialog();
		}
}
```



Dette simple program startes med en fil som parameter og filens indhold vises i formens tekst boks. Som det ser ud nu vil det kunne åbne en hvilken som helst fil. Man kan opfatte Main() som en klient der kalder op til en server (klassen S) og jeg forestille mig at jeg er S! Måske ønsker jeg så ikke at den 'fremmede' computer skal kunne se alle mine filer uden videre. For at styre dette kan jeg anvende CAS til at skabe en Permission tilstand. (Klienten har pt ubegrænsede Permissions så at sige!).

En begrænsning af klientens rettigheder kan ske på flere måder fx ved at bruge en **attribut** – såkaldt 'declarative security' - ved den metode som henter filen sådan her (her er det relevante uddrag af koden):

```
public S(string s){

                box=new RichTextBox();
                box.Dock=DockStyle.Fill;
                Controls.Add(box);
                get_fil(s);

}

[FileIOPermission(SecurityAction.PermitOnly, Read="C:\\csharp\\")]

private static void get_fil(string s){
                reader=File.OpenText(s);
                string fil=reader.ReadToEnd();
                box.Text=fil;


}
```

Lige før metoden get_fil() indsættes en attribut med en Permission som består af to dele:

1. Klienten har **kun** adgang til filer i mappen c:\csharp (og alle undermapper herunder) !! (Dette svarer til at have en brugerkonto på en maskine som bruges af mange brugere).
2. Klienten kan kun **læse** filer, ikke skrive til filsystemet eller eksekvere filer!! (PermitOnly).

Hvis programmet kaldes med en 'ikke tilladt' fil fås en lang fejl meddelelse (en Security Exception hvoraf uddrag vises her):

```
tackCrawlMark& stackMark)
   at System.Security.CodeAccessPermission.Demand()
   at System.IO.FileStream..ctor(String path, FileMode mode, FileAcc
FileShare share, Int32 bufferSize, Boolean useAsync, String msgPath,
omProxy)
   at System.IO.FileStream..ctor(String path, FileMode mode, FileAcc
FileShare share, Int32 bufferSize)
   at System.IO.StreamReader..ctor(String path, Encoding encoding, B
EncodingFromByteOrderMarks, Int32 bufferSize)
   at System.IO.StreamReader..ctor(String path)
   at System.IO.File.OpenText(String path)
   at S.get_fil(String s)
   at S.get_fil(String s)
   at S..ctor(String s)
   at S.Main(String[] args)

The state of the failed permission was:
IPermission class="System.Security.Permissions.FileIOPermission, ms
on=1.0.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
            version="1"
            Read="c:\autoexec.bat"/>
```

Som det ses har jeg her prøvet at åbne c:\autoexec.bat og jeg får at vide at jeg ikke har den nødvendige CodeAccessPermission og at der kastes en Exception i metoden get_fil() ved sætningen File.OpenText() osv.
At indsætte en attribut før en metode er altså et ret effektivt sikkerheds instrument.

Ovenstående attribut (FileIOPermission) kan udformes på forskellige måder. Vi kan f.eks. ændre attributten til følgende:

```
[FileIOPermission(SecurityAction.Assert, Write="C:\\csharp\\")]
private static void get_fil(string s){
                reader=File.OpenText(s);
                string fil=reader.ReadToEnd();
                box.Text=fil;

}
```

Assert betyder at klienten tildeles en rettighed uanset om han evt. har den på forhånd! I dette tilfælde får klienten altså evt flere rettigheder – han kan nu skrive til filer i mappen! Men han kan

også åbne filer alle andre steder (hvis han allerede har denne rettighed – ingen af hans rettigheder tages fra ham i al fald!).

Hvis vi ønsker at checke/kontrollere om klienten har bestemte rettigheder i forvejen skal attributten anvende demand sådan:

```
[FileIOPermission(SecurityAction.Demand, Read="C:\\csharp\\")]

private static void get_fil(string s){
        reader=File.OpenText(s);
        string fil=reader.ReadToEnd();
        box.Text=fil;


}
```

Endelig kan vi fratage brugeren/klienten hans rettigheder sådan ved brug af Deny:

```
[FileIOPermission(SecurityAction.Deny, Read="C:\\")]
```

Virkningen af denne attribut er at klienten **ikke** kan læse **nogen** fil på C-drevet. MEN klienten har frie rettigheder mht filer på andre drev fx på A-drevet eller D-drevet!!

At anvende attributter – som vist ovenfor – er ofte den simpleste metode - men der eksisterer en anden metode – såkaldt **'imperative** security' – som ikke anvender attributter. Her er et eksempel som gør det samme som det første attribut eksempel ovenfor:

```
private static void get_fil(string s){

        FileIOPermission p = new FileIOPermission
        (FileIOPermissionAccess.Read, "c:\\csharp\\");
        p.PermitOnly();
        reader=File.OpenText(s);
        string fil=reader.ReadToEnd();
        box.Text=fil;


}
```

I stedet for en attribut forud for metoden indsættes kode inden i metoden som instantierer en Permission klasse og kalder den med Assert(), Demand(), PermitOnly() osv. Som det ses er deklarativ og imperativ security metoderne næsten identiske! Resultatet af denne kode er igen at klienten kan læse filer fra mappen c:\csharp samt undermapper – men **ikke** nogen anden fil overhovedet!

Fordelen ved at bruge attributter er at man ved hjælp af Reflection kan se på forhånd hvilken 'security' der er defineret i en assembly (dens 'security policy')!

Den modsatte sikkerheds situation kan defineres sådan:

Jeg har downloadet en applikation som jeg ikke helt stoler på. Jeg kan derfor starte programmet med egne security definitioner fx som vist i dette eksempel:

```
// CAS eksempel Code Access Security:

using System.Windows.Forms;
using System;
using System.IO;
using System.Security;
using System.Security.Permissions;



public class S : Form {

		private static RichTextBox box;
		private static StreamWriter writer;

		public S(){

				box=new RichTextBox();
				box.Dock=DockStyle.Fill;
				box.Text="Velkommen og god fornøjelse!";
				Controls.Add(box);
				skriv_filer();

		}


		private static void skriv_filer(){
				for(int i=0;i<10;i++){
				writer=File.CreateText("nr "+i+".txt");
				writer.Close();
				}
		}


		public static void Main(string[] args){
				S s=new S();
				s.ShowDialog();
		}
}
```

Man skal her forestille sig at jeg er klienten Main() og at jeg ikke har nogen viden om hvad klassen S egentligt laver! Når jeg kører programmet – som jeg måske har downloadet – ses dette:

Det ser jo ikke så farligt ud! Problemet er selvfølgeligt at programmet 'skjult' begynder at skrive til min harddisk!

For at sikre mig mod den slags overraskelser kalder jeg programmet med en **security** definition før Main() således:

```
[FileIOPermission(SecurityAction.PermitOnly, Read="C:\\temp\\")]

public static void Main(string[] args){
                S s=new S();
                s.ShowDialog();
}
```

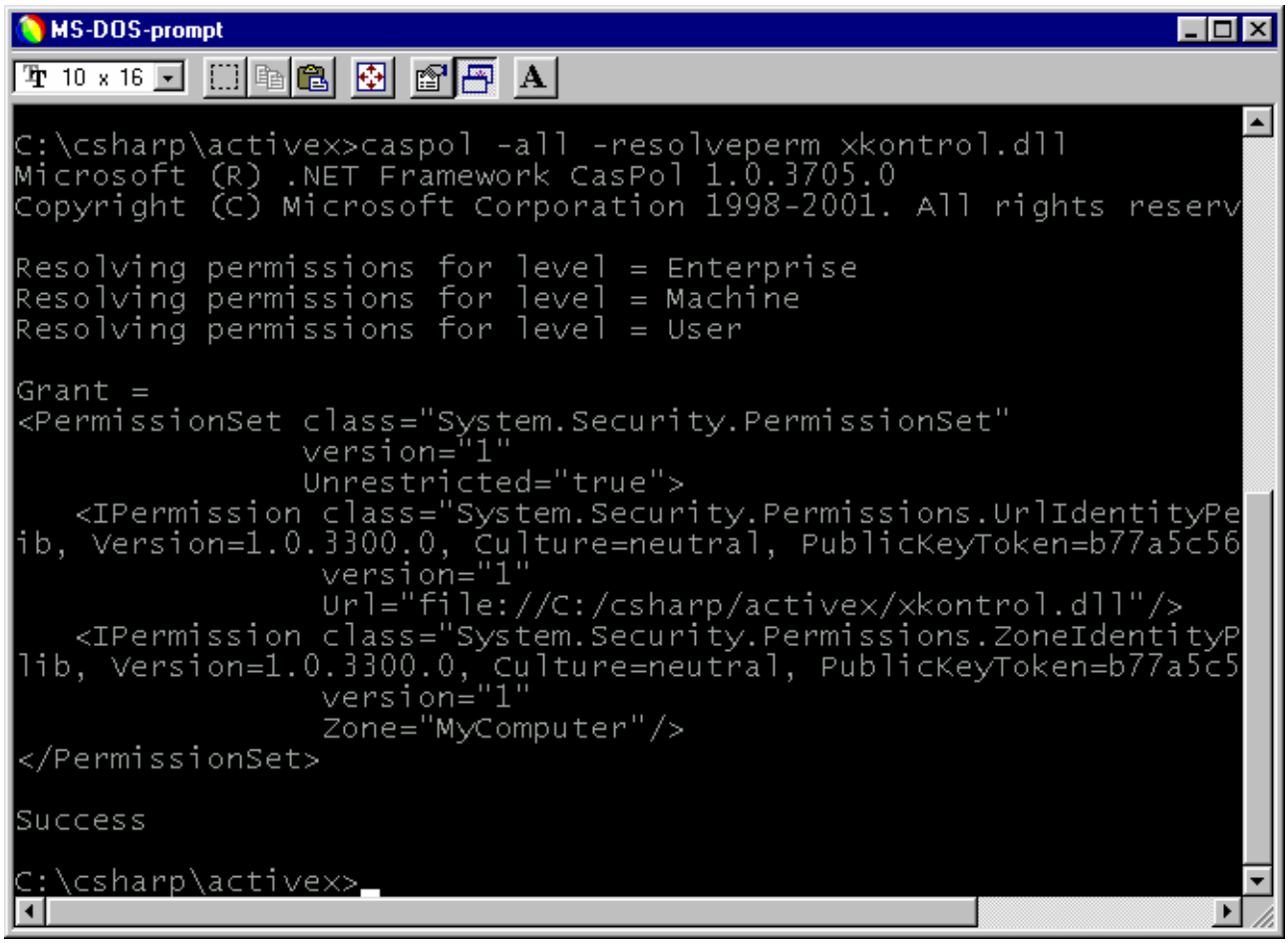
Det **eneste,** som det ikke helt troværdige fremmede program nu kan, er at **læse** filer og kun i mappen c:\temp. Det kan ikke skrive, slette, ændre el. lignende. Attributten er altså en **meget** voldsom indskrænkning af privilegier og rettigheder!!

Attributten er en slags beskyttelse af mig selv mod et ukendt, fremmed program som måske har onde hensigter! Derfor skal attributten anbringes lige før Main()!

Hvis jeg nu prøver at køre programmet vil det kaste en SecurityException og stoppe! Der sker altså ingen skader på mit fil system!

caspol:

På kommandolinjen kan caspol bruges til at vise og ændre CAS. caspol kan kaldes med en række forskellige parametre – se hjælpen! F.eks. giver caspol –list en MEGET lang liste over kodegrupper og caspol –all –resolveperm xkontrol.dll giver en liste over en bestemt assemblys permissions:



caspol –af (eller:)
caspol -addfulltrust <assembly_name>

tildeler en assembly fulde privilegier og

caspol -rf
caspol -remfulltrust <assembly_name>

fjerner disse fulde privilegier.

Profilering af C# programmer:

En **'profiler'** er et program som analyserer et andet program og som kan udskrive en række data om programmets brug af memory, kald af metoder og assemblies, tidsforbrug osv.

Profiler-programmer kan vise helt **utrolige** data: Hvis en profiler køres på et forholdsvis simpelt Windows program kan det vise, at i alt 41.488 forskellige metoder kaldes alt i alt og at en af metoderne blev kaldt 18.256 gange i løbet af den korte tid programmet kørte!! En profiler kan altså vise data som normalt er 'skjult' og som ikke kan ses i den blotte C# kode!

Ved at køre et C# program gennem en profiler kan man se hvor programmet 'går i stå' eller hvor man evt skal gøre noget for at få det til at køre mere effektivt og hurtigere (det som på engelsk kaldes programmets **'performance'**).

Vi skal her se på et simpelt eksempel på en profiler-klasse som kan måle tidsforbruget i et C# program (et slags 'stop-ur'). Vi har tidligere i forbindelse med DateTime klassen set lidt på dette. Det er også muligt at bruge DateTime.Now.Ticks til at måle en start og slut tid – men denne måling er kun rimelig nøjagtig hvis tiden der måles er et halvt sekund eller mere! Hvis vi skal måle mikro eller millisekunder er DateTime klassen simpelt hen for upræcis!

1 millisekund = 1/1000 sekund
1 mikrosekund = 1/1000000 sekund
1 nanosekund = 1/1000000000 sekund

En meget mere præcis måling – fx af mikrosekunder - kan fås ved at kalde metoder i Windows API'en (ikke .NET kode). I nedenstående eksempel – klassen IntervalTimer - kalder vi 2 metoder i Windows-filen **kernel32**.dll således (kernel32.dll ligger i C:\Windows\System, men det er ikke nødvendigt at angive nogen sti hertil):

```
using System;
using System.Runtime.InteropServices;


        ///<summary>Klassen IntervalTimer - anvender Windows API og DllImport</summary>

        public class IntervalTimer
        {
                [DllImport("kernel32.dll")]
                static extern private int QueryPerformanceCounter(out long count);

                [DllImport("kernel32.dll")]
                static extern private int QueryPerformanceFrequency(out long count);

        ///<summary>TimerState enum har 3 tilstande: Started, Stopped, NotStarted</summary>

                public enum TimerState {NotStarted, Stopped, Started}

                private TimerState state;
                private long ticksAtStart;
                private long intervalTicks;
```

```
                private static long frequency;
                private static int decimalPlaces;
                private static string formatString;
                private static bool initialized = false;

        ///<summary>Constructor kalder Windows API: QueryPerformanceFrequency(out
frequency)</summary>

                public IntervalTimer()
                {
                        if (!initialized)
                        {
                                QueryPerformanceFrequency(out frequency);
                                decimalPlaces = (int)Math.Log10(frequency);
                                formatString = String.Format("Interval: {{0:F{0}}}
seconds ({{1}} ticks)", decimalPlaces);
                                initialized = true;
                        }
                        state = TimerState.NotStarted;
                }

                ///<summary>public void Start() starter timeren</summary>

                public void Start()
                {
                        state = TimerState.Started;
                        ticksAtStart = CurrentTicks;
                }
                ///<summary>public void Stop() stopper timeren</summary>

                public void Stop()
                {
                        intervalTicks = CurrentTicks - ticksAtStart;
                        state = TimerState.Stopped;
                }
                ///<summary>public float GetSeconds(): EFTER at timeren er stoppet!</summary>

                public float GetSeconds()
                {
                        if (state != TimerState.Stopped)
                                throw new TimerNotStoppedException();
                        return (float)intervalTicks/(float)frequency;
                }
                ///<summary>public long GetTicks(): EFTER stop!</summary>

                public long GetTicks()
                {
                        if (state != TimerState.Stopped)
                                throw new TimerNotStoppedException();
                        return intervalTicks;
                }
                ///<summary>property: CurrentTicks af typen long</summary>

                private long CurrentTicks
                {
                        get
                        {
                                long ticks;
```

```
                                        QueryPerformanceCounter(out ticks);
                                        return ticks;
                                }
                        }
                        ///<summary>ToString() EFTER stop: udskriver sekunder og ticks </summary>

                        public override string ToString()
                        {
                                if (state != TimerState.Stopped)
                                        return "Interval timer, state: " + state.ToString();
                                return String.Format(formatString, GetSeconds(), intervalTicks);
                        }

}
                        ///<summary>Exception: TimerNotStoppedException()</summary>

public class TimerNotStoppedException : ApplicationException
{
                        public TimerNotStoppedException()
                                : base("Timer is either still running or has not been started")
                        {
                        }
}
```

De 2 metoder 'wrappes' på denne måde – med en attribut [DllImport()]. Derefter kan vi bruge disse ikke-DotNet metoder i C# koden! Den første metode returnerer et antal ticks, den anden returnerer hvor mange ticks maskinen kører pr sekund (frekvensen er afhængig af maskinens hardware!).

```
[DllImport("kernel32.dll")]
                        static extern private int QueryPerformanceCounter(out long count);

                        [DllImport("kernel32.dll")]
                        static extern private int QueryPerformanceFrequency(out long count);
```

Koden kompileres til en dll fil: intervaltimer.dll.

Vi kan nu meget nemt bruge denne klasse i andre programmer til at måle tidsforbuget i hele programmet eller i de enkelte metoder eller i dele af en metode! (NB det sidste kan ikke lade sig gøre med en professionel profiler – så vi får lidt mere fleksibilitet på denne måde!).

Vi skal selvfølgelig compilere programmet med en reference til intervaltimer.dll, som skal ligge i samme mappe som programmet der skal 'profileres' eller testes!

Vi kalder simpelt hen metoden Start(), kører koden i et program, kalder metoden Stop() og tilsidst metoden timer.ToString(). Vi kan illustrere dette med et database program som vi tidligere har brugt (der er oprettet en private IntervalTime timer, følgende er kun uddrag af koden):

```
                        public DataForm(){

                                //start timer til at maale performance:
                                timer=new IntervalTimer();
                                timer.Start();
```

```
                                        Text="Data fra boghandel.mdb";

                                        //DB connection:
                                        cn=new OleDbConnection();
                                        cn.ConnectionString="provider=Microsoft.JET.OLEDB.4.0;data
source=c:\\dokumenter\\boghandel.mdb";

                                        cn.Open();
                                        adapter=new OleDbDataAdapter("select * from bog",cn);
                                        dataset=new DataSet();
                                        adapter.Fill(dataset,"bog");
                                        //tabellen oprettes først her

                                        grid.SetDataBinding(dataset,"bog");
                                        grid.CaptionText="Boghandel.mdb";

                                        //NB hver adapter sin command builder!
                                        builder=new OleDbCommandBuilder(adapter);
                                        dataset.WriteXml("boghandel.xml");
                                        dataset.WriteXmlSchema("boghandel_skema.xml");
                                        cn.Close();

                                        timer.Stop();
                                        MessageBox.Show ("Opstarten varede: "+timer.ToString(),
"IntervalTimer:");

                                }
```

Når programmet startes fås en boks i stil med denne:

De faktiske værdier vil selvfølgelig være afhængige af maskinens hardware (CPU) mv.

I stedet kunne også måles hvor lang tid metoderne der skriver til XML filer tager:

```
                                        timer=new IntervalTimer();
                                        timer.Start();

                                        dataset.WriteXml("boghandel.xml");
                                        dataset.WriteXmlSchema("boghandel_skema.xml");

                                        timer.Stop();

                                        cn.Close();

                                        MessageBox.Show ("WriteXml() og WriteXmlSchema():
"+timer.ToString(),  "IntervalTimer:");
```

På Internettet kan downloades forskellige shareware eller demo modeller af profiler-programmer. Eksempler: http://www.gotdotnet.com eller http://www.red-gate.com.

Kryptering:

Kryptering er en del af området 'Security'. Formålet med at kryptere en tekst er at kun visse personer kan læse (dekryptere) teksten.

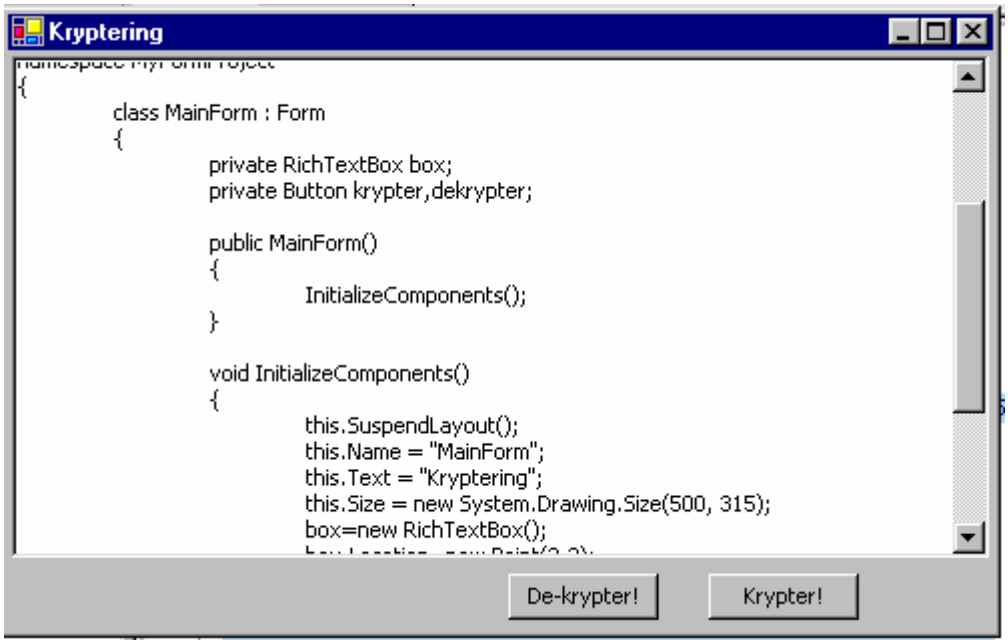
Kryptering findes i to udgaver: symmetrisk (shared key) og asymmetrisk (public key/private key) kryptering. Vi vil i dette afsnit se på symmetrisk kryptering, som indebærer at modtager og afsender begge anvender den samme nøgle til både kryptering og dekryptering. Ulempen ved dette system er at begge skal holde nøglen hemmelig for andre (!) og at nøglen – inden processen kan starte – skal overgives mellem parterne. Problemet her er at en sådan nøgle jo ikke kan sendes fx over internettet – for så kan den jo komme i forkerte hænder! Men når først begge parter (eller flere parter) har nøglen – fungerer symmetrisk kryptering OK. En nøgle er på et vist antal bytes/bits og en nøgle på 64 bits (eksemplet nedenfor) er **meget** svær at bryde! Systemet er altså pålideligt!

I .NET er defineret en mængde klasser og metoder mht kryptering i mscorlib.dll i namespace System.Security.Cryptography. De metoder som vi vil anvende findes bl a i klasserne ICryptoTransform og SymmetricAlgorith i dette namespace. Den mest kendte symmetriske algoritme er DES – Digital Encryption Standard, men i .NET findes en række andre algoritmer (klasser), som kan anvendes fuldstændigt svarende til DES. De matematiske formler der 'oversætter' et tegn i originalteksten til et tegn i den krypterede (chiffrede) tekst er **meget** komplicerede – og vi vil derfor ikke gå ind i dem.

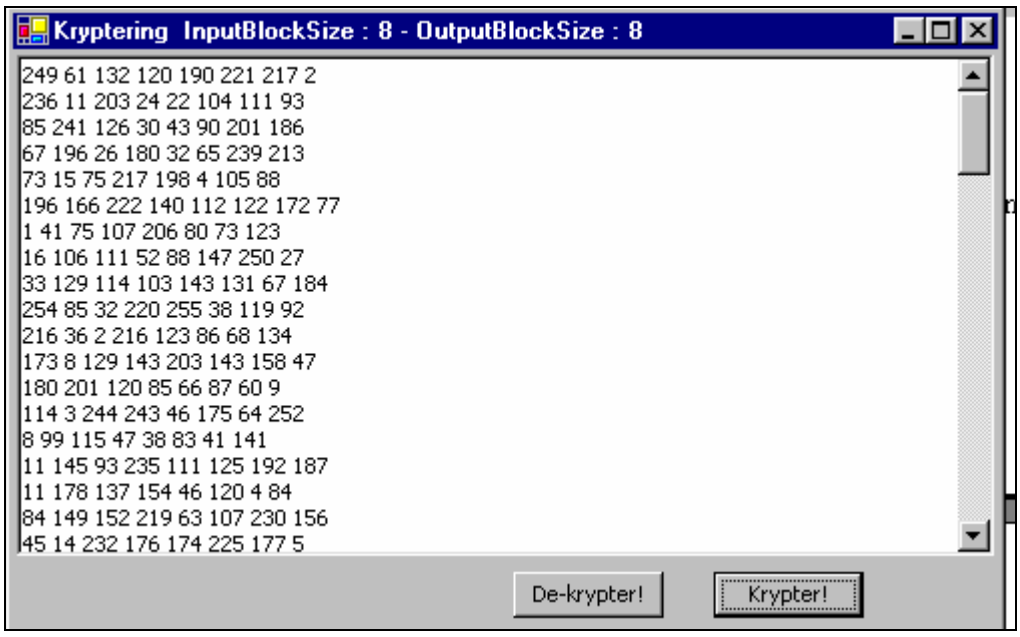
I princippet fungerer krypteringen som nogle af os gjorde som børn: Fx et 'a' i teksten krypteres til et 'c', et 'f' til et 'h' osv (ryk 2 bogstaver frem-algoritmen!!). DES fungerer ved at algoritmen læser en blok af tegn (her 8 tegn/bytes), krypterer dem, læser næste blok og krypterer den osv.

Dette har den konsekvens at hvis jeg indtaster fx 11 tegn vil den krypterede tekst indeholder 16 tegn, hvis jeg indtaster 766 tegn fylder den kodede tekst 768 bytes osv, fordi der altid arbejdes med en blok på 8 tegn (bytes)!

Når nedenstående program kører ser det således ud:

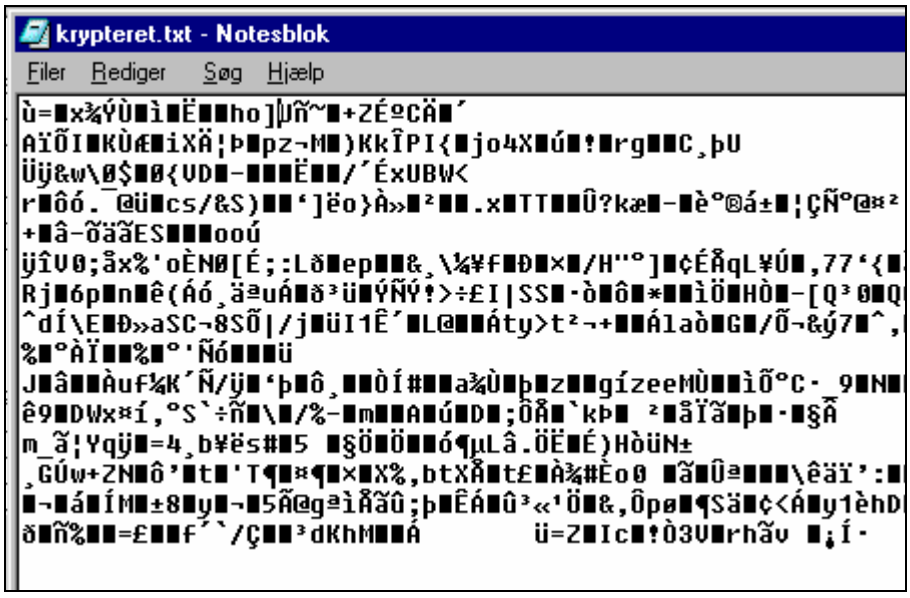


Teksten i tekstboksen kan enten skrives i hånden eller kopieres fra et andet sted og indsættes med Control+V! (Her er indkopieret en del af C# koden til programmet som eksempel).
Når der klikkes på Krypter knappen fås:

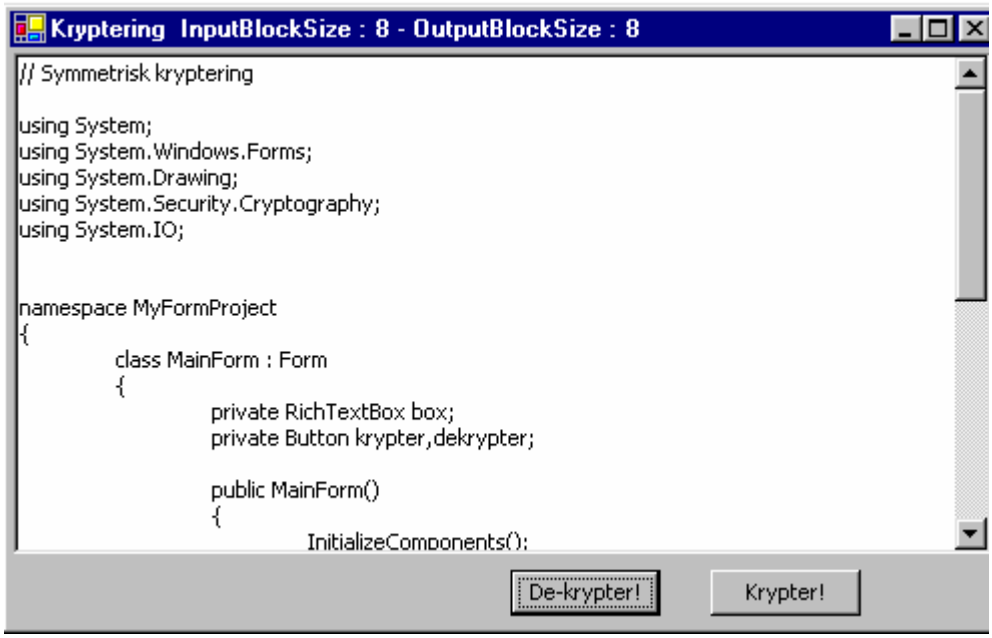


Den krypterede tekst vises som bytes linje for linje idet hver linje udgør en blok i chiffer teksten.

Den krypterede tekst kan også åbnes i Notesblok:



Når der derefter klikkes på Dekrypter knappen fås:



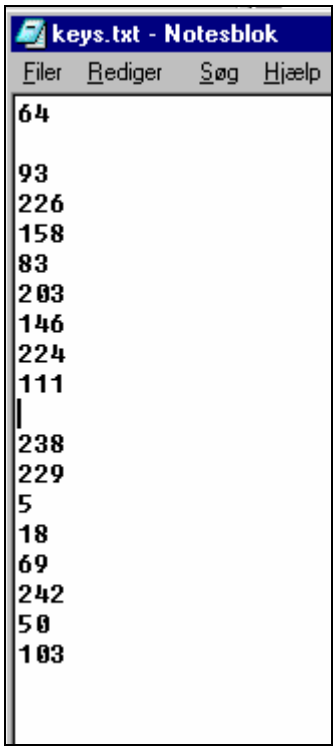
Programmet i sin helhed vil blive vist nedenfor men vi vil først se på dele af koden. Det logisk første skrift er at generere en nøgle – som i DES består af en Key på 8 byes og en byte array IV (initialiserings vektor) på 8 bytes. Disse oprettes ved at computeren simpelt hen finder nogle random værdier (bytes) (dele af koden er her oversprunget):

```
private void krypt(object o, EventArgs a){

    Text="Kryptering ";

    DESCryptoServiceProvider des = new DESCryptoServiceProvider();
```

```
                    //Provideren har disse indbyggede metoder
                    //til at finde en key og en initialiserings vektor:
                    //disse findes som random heltal (0..255):
                    //hver gang GenerateKey() kaldes skabes en NY key!!:

                    des.GenerateKey();
                    des.GenerateIV();

                    gem_key_vektor(des);
}

//Gem data til kryptering/de-kryptering i fil:

private void gem_key_vektor(DES des)

{
                    StreamWriter ud_fil = new StreamWriter("keys.txt", false);

                    //KeySize er her 64 bits dvs 8 bytes (8*8 bits):
                    //Den krypterede tekst består af multipler af 8
                    //fx 8, 16, 24, 32 osv bytes (aldrig fx af 19 bytes)!!

                    ud_fil.WriteLine(des.KeySize);

                    //Key bestaar af 8 bytes (tal 0..255)
                    for (int i=0 ; i< des.KeySize/8 ; i++)
                                        ud_fil.WriteLine(des.Key[i]);

                    //Init Vektoren bestaar af 8 bytes:
                    for (int i=0 ; i< des.KeySize/8 ; i++)
                                        ud_fil.WriteLine(des.IV[i]);
                    ud_fil.Close();
}
```

Hvis programmet skal bruges i praksis skal det altså først oprette en keys.txt som indeholder en key size, en key og en vektor. Programmet nedenfor opretter en ny key hver gang programmet kører – men sådan skal det selvfølgelig ikke køre i praksis! I virkeligheden skal nøglen jo kun oprettes een gang for alle og encryptor og decryptor skal så begge bruge denne samme nøgle!

keys.txt kan fx se således ud:



Her er teksten kunstigt delt bagefter for at vise at den første værdi (64) er key size dvs at denne nøgle er en 64 bit nøgle. De næste 8 værdier er **key** (8 sub keys) og de sidste 8 bytes er **vektoren,** som bruges til at initialisere krypterings algoritmen. Hver gang metoden kører oprettes en NY key!!

Det er denne keys.txt som afsender og modtager **begge** skal råde over hvis de skal kommunikere krypteret!

Krypterings program eksempel:

Her følger hele koden til eksempel programmet:

```
// Symmetrisk kryptering

using System;
using System.Windows.Forms;
using System.Drawing;
using System.Security.Cryptography;
using System.IO;


namespace MyFormProject
{
		class MainForm : Form
		{
				private RichTextBox box;
				private Button krypter,dekrypter;

				public MainForm()
				{
```

```
                        InitializeComponents();
                }

                void InitializeComponents()
                {
                        this.SuspendLayout();
                        this.Name = "MainForm";
                        this.Text = "Kryptering";
                        this.Size = new System.Drawing.Size(500, 315);
                        box=new RichTextBox();
                        box.Location=new Point(2,2);
                        box.Size=new Size(488,250);
                        Controls.Add(box);

                        krypter=new Button();
                        krypter.Text="Krypter!";
                        krypter.Location=new Point(350,260);
                        krypter.Click+=new EventHandler(krypt);
                        Controls.Add(krypter);

                        dekrypter=new Button();
                        dekrypter.Text="De-krypter!";
                        dekrypter.Location=new Point(250,260);
                        dekrypter.Click+=new EventHandler(dekrypt);
                        Controls.Add(dekrypter);

                        this.ResumeLayout(false);
                }

                private void dekrypt(object o, EventArgs a){

                        DESCryptoServiceProvider des = new DESCryptoServiceProvider();
                        ReadKeyAndIV(des);

                        ICryptoTransform decryptor = des.CreateDecryptor();

                        FileStream inFile = new FileStream("krypteret.txt", FileMode.Open);
                        FileStream outFile = new FileStream("dekrypteret.txt",
FileMode.Create);
                        int inSize = decryptor.InputBlockSize;
                        int outSize = decryptor.OutputBlockSize;
                        byte [] inBytes = new byte[inSize];
                        byte [] outBytes = new byte[outSize];
                        int numBytesRead, numBytesOutput;
                        do
                        {
                                numBytesRead = inFile.Read(inBytes, 0, inSize);
                                if (numBytesRead == inSize)
                                {
                                        numBytesOutput =
decryptor.TransformBlock(inBytes, 0, numBytesRead, outBytes, 0);
                                        outFile.Write(outBytes, 0,
numBytesOutput);
                                }
                                else
                                {
                                        byte [] final =
decryptor.TransformFinalBlock(inBytes, 0, numBytesRead);
```

```
                                                    outFile.Write(final, 0, final.Length);
                                }
                } while (numBytesRead > 0);
                inFile.Close();
                outFile.Close();

                //Vis dekrypteret tekst i tekstboksen:
                StreamReader reader=File.OpenText("dekrypteret.txt");
                box.Text=reader.ReadToEnd();
                reader.Close();
}


//NB DES er basis klassen for provideren (DESCryptoServiceProvider):

//NB Normalt vil encryptor ogsaa laese key data fra en fil
//og ikke producere en NY key HVER GANG!!:

private void ReadKeyAndIV(DES des)
{
                StreamReader inFile = new StreamReader("keys.txt");
                int keySize;

                //tallet i 1 linje i filen angiver antal bits i key (her 64):
                keySize = int.Parse(inFile.ReadLine());

                //keySize/8 giver antal bytes i key og vektor:

                byte [] key = new byte[keySize/8];
                byte [] iv = new byte[keySize/8];

                //find de 8 sub-keys:
                for (int i=0 ; i< des.KeySize/8 ; i++)
                                key[i] = byte.Parse(inFile.ReadLine());

                //find de 8 elementer i init vektoren:
                for (int i=0 ; i< des.KeySize/8 ; i++)
                                iv[i] = byte.Parse(inFile.ReadLine());

                inFile.Close();

                //indstil de 3 properties i DES provideren:
                des.KeySize = keySize;

                //property Key og IV (vektor) er 2 byte arrays:
                des.Key = key;
                des.IV = iv;
}


private void krypt(object o, EventArgs a){

                Text="Kryptering ";

                //Der findes mange forskellige algoritmer fx:
                //RijndaelManaged des=new RijndaelManaged();
                //resten af koden er stort set den samme!!
```

```
                                        DESCryptoServiceProvider des = new DESCryptoServiceProvider();

                                        //Provideren har disse indbyggede metoder
                                        //til at finde en key og en initialiserings vektor:
                                        //disse findes som random heltal (0..255):
                                        //hver gang GenerateKey() kaldes skabes en NY key!!:

                                        des.GenerateKey();
                                        des.GenerateIV();

                                        gem_key_vektor(des);
                                        ICryptoTransform encryptor = des.CreateEncryptor();

                                        //krypter det bruger har skrevet i tekst boksen:
                                        StreamWriter w=File.CreateText("eksempel.txt");
                                        w.Write(box.Text);
                                        w.Close();
                                        box.Text="";

                                        FileStream ind_fil = new FileStream(@"eksempel.txt",
FileMode.Open);
                                        FileStream ud_fil = new FileStream(@"krypteret.txt",
FileMode.Create);


                                        //BlockSize er det antal bytes som kodes ad gangen (her 8 bytes):
                                        int ind_size = encryptor.InputBlockSize;

                                        Text+=" InputBlockSize : "+ind_size;

                                        int ud_size = encryptor.OutputBlockSize;

                                        Text+=" - OutputBlockSize : "+ud_size;

                                        byte [] bytes_ind = new byte[ind_size];
                                        byte [] bytes_ud = new byte[ud_size];
                                        int antal_bytes_ind, antal_bytes_ud;
                                        do
                                        {
                                                antal_bytes_ind = ind_fil.Read(bytes_ind, 0, ind_size);
                                                if (antal_bytes_ind == ind_size)
                                                {
                                                        antal_bytes_ud =
encryptor.TransformBlock(bytes_ind, 0, antal_bytes_ind, bytes_ud, 0);
                                                        ud_fil.Write(bytes_ud, 0,
antal_bytes_ud);
                                                }
                                                else if (antal_bytes_ind > 0)
                                                {

                                                        //den sidste blok er maaske mindre
end 8 bytes:
                                                        byte [] sidste_bytes =
encryptor.TransformFinalBlock(bytes_ind, 0, antal_bytes_ind);
                                                        ud_fil.Write(sidste_bytes, 0,
sidste_bytes.Length);
                                                }
                                        } while (antal_bytes_ind > 0);
```

```
                                        ind_fil.Close();
                                        ud_fil.Close();

                                        FileStream stream=new
FileStream("krypteret.txt",FileMode.Open,FileAccess.Read);
                                        BinaryReader reader=new BinaryReader(stream);
                                        int linje=0;
                                        while(reader.PeekChar()!=-1){

                                                //Evt kan tekstboksen vise tegn i stedet for bytes (tal):
                                                //box.Text+=(char)reader.ReadByte();

                                                box.Text+=reader.ReadByte();
                                                box.Text+=" ";

                                                linje++;
                                                if(linje%8==0)box.Text+="\n";
                                        }
                                        stream.Close();
                                        reader.Close();

                        }

                        //Gem data til kryptering/de-kryptering i fil:

                        private void gem_key_vektor(DES des)

                        {
                                        StreamWriter ud_fil = new StreamWriter(@"keys.txt", false);

                                        //KeySize er her 64 bits dvs 8 bytes (8*8 bits):
                                        //Den krypterede tekst består af multipler af 8
                                        //fx 8, 16, 24, 32 osv bytes (aldrig fx af 19 bytes)!!

                                        ud_fil.WriteLine(des.KeySize);

                                        //Key bestaar af 8 bytes (tal 0..255)
                                        for (int i=0 ; i< des.KeySize/8 ; i++)
                                                ud_fil.WriteLine(des.Key[i]);

                                        //Init Vektoren bestaar af 8 bytes:
                                        for (int i=0 ; i< des.KeySize/8 ; i++)
                                                ud_fil.WriteLine(des.IV[i]);
                                        ud_fil.Close();
                        }




                        public static void Main(string[] args)
                        {
                                        Application.Run(new MainForm());
                        }
                }
}
```

På http://csharpkursus.subnet.dk ligger et lignende eksempel, som anvender algoritmen **Rijndael** og en Crypto stream. Denne algoritme krypterer teksten med en 128 bit nøgle.

Opgaver:

1. Skriv 3 **separate** programmer som opretter en nøgle, krypterer en tekst og de-krypterer en tekst. Programmerne skal fungere som Windows programmer på den måde som du synes er mest effektivt set fra brugerens synspunkt.
2. Undersøg om systemet virker: Send krypteret tekst over internettet til en anden evt til dig selv og undersøg om den kan dekrypteres korrekt.
3. Du kan evt. sende teksten og nøglen til mig!

Attributter:

En attribut er i C# og i .NET en anmærkning som sættes på et element fx på en klasse eller metode. I C++ og i Java findes ikke attributter, men i COM og Visual Basic bruges attributter (defineret i IDL – Interface Definition Language).

At anvende attributter kaldes 'aspekt orienteret programmering'. Attributter definerer bestemte generelle aspekter ved koden.

Der findes flere hundrede forud definerede attributter i .NET – fordelt i de forskellige DLL filer og namespaces.

En assembly består grundlæggende af to dele: først en sektion med metadata og dernæst en sektion med kode. Attributter gemmes i metadata. Når jeg tilføjer en attribut til en assembly eller klasse lægger jeg altså data ind i metadata.
En attribut er i C# simpelthen en klasse som arver fra System.Attribute og som instantieres i attribut erklæringen.
Der findes i C# grundlæggende to typer af attributter:

1. Visse forud definerede attributter som systemet genkender **og** hvor systemet **automatisk** vil foretage visse handlinger hvis attributten optræder.
2. Alle øvrige attributter inklusive attributter, som jeg selv har defineret

Et simpelt eksempel på anvendelsen af attributter er følgende:

```
// Attributter i C#

using System;
using System.Reflection;

[assembly: AssemblyVersion("1.2.*")]
[assembly: AssemblyTitle("C# Attributter")]
[assembly: AssemblyDescription("Dette program illustrerer C# attributter")]
[assembly: AssemblyCompany("World Digital Inc.")]
```

```
public class A{

		public A(){}

}
class app {

		public static void Main(){

				Console.WriteLine("Attribut demo.");

		}
}
```

Her defineres 4 attributter dvs de 4 attribut klasser instantieres med passende parametre. Attributterne er her forsynet med et 'scope' som definerer at denne attribut gælder hele denne assembly. De 4 attributter er alle attributter som C# genkender – MEN systemet foretager IKKE bestemte handlinger når disse attributter forekommer. Deres data lægges blot ind i metadata hvor de så kan hentes frem via reflection og bruges. (Men dette skal altså eksplicit kodes – systemet er for så vidt ligeglad med disse attributter).
Hvis et nyt projekt åbnes i **SharpDevelop** vil der blive oprettet en række af disse standard attributter automatisk og de kan defineres i filen AssemblyInfo.cs som kompileres sammen med cs filen.

Hvis filen kompileres kan man fx i Windows Stifinder se exe filens **egenskaber**:

Versionen angives altid som 4 tal i formatet:

major.minor.build.revision

Da vi blot har angivet **major** (den overordnede version) og **minor** (under-versionen) (”1.2.*”) - udfylder systemet selv resten. Efterhånden som vi ændrer og gen-kompilerer filen vil de to sidste værdier build og revision automatisk blive opdateret af systemet!

Vi kan også se at ALLE C# filer automatisk af systemet tildeles visse attributter!

Vi vil du se på nogle attributter som systemet automatisk genkender og handler på:

Vi skriver koden om således:

```
public class A{

		public A(){}

		//false giver warning, true giver error
		[Obsolete("Denne metode er håbløst forældet!",false)]
		public int metode(){
				Console.WriteLine("Metode i klassen A!");
				return 1;
		}

}
class app {

		public static void Main(){

				Console.WriteLine("Attribut demo.");
				A a=new A();
				a.metode();

		}
}
```

Vi har nu sat en attribut (Obsolete – ’forældet’) på metoden i klassen A. Hvis den anden parameter sættes til false sker følgende:

```
C:\csharp\attr>csc attr1.cs
Microsoft (R) Visual C# .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.

attr1.cs(30,3): warning CS0618: 'A.metode()' is obsolete: 'Denne metode er
        håbløst forældet!'
```

Når vi kompilerer programmet får vi en **warning** – **men** hvis programmet kører vil metoden blive udført:

Hvis vi i stedet sætter attributten til **true** – vil vi få en error og metoden vil **ikke** blive udført!

System.ObsoleteAttribute er altså et eksempel på en attribut som systemet genkender og hvor systemet ved hvad det skal gøre.

Andre eksempler herpå er:

1. Security (se afsnittet om Code Access Security)
2. Serializable (som bevirker at objektet kan serialiseres – se tidligere i kurset)
3. CLSCompliant (som chekker om koden KUN anvender de grundlæggende basis typer i .NET – en uint (int uden fortegn) er fx ikke CLSCompliant)
4. DllImport (som bruges til at kalde native metoder – altså ikke .NET kode - i Windows API)
5. StructLayout (som definerer fx ANSII eller Unicode for en assembly)
6. Flags som – anvendt foran en enum – betyder at værdierne i enum'en fungerer som bits som kan OR'es (dvs en instans kan sættes til flere værdier i enum'en samtidigt)

Hvis disse attributter sættes på et element i koden vil .NET reagere **automatisk**.

Hvis vi omskriver koden kan vi se et andet eksempel på at systemet reagerer 'automatisk' på en attribut:

```
public class A{

		public A(){}

		[Conditional("TEST")]
		public void test_metode(){
				Console.WriteLine("Test metoden kaldes i A.");
		}

}
class app {

		public static void Main(){

				Console.WriteLine("Attribut demo.");
				A a=new A();
				a.test_metode();

		}
}
```

Desuden skal tilføjes en ny using:

using System.Diagnostics;//JVF attributten Conditional, som defineres i dette namespace!!

Hvis vi kompilerer koden med: csc attr1.cs og eksekverer programmet fås:

Meoden test_metode() bliver IKKE kompileret og bliver IKKE eksekveret!

Hvis vi i stedet kompilerer filen med csc /d:TEST attr1.cs (**eller**: csc /define:TEST attr1.cs - altså med et define flag) fås:

Attributten Conditional (i System.Diagnostics) kan altså bruges i debugging.

Attributter kan opføres hver for sig (i hver deres skarpe parenteser) **eller** samlet f.eks. således:

```
[AttributeUsage(AttributeTargets.Delegate |
AttributeTargets.Interface | AttributeTargets.Method,
AllowMultiple=true)]
```

Ny attribut klasse: DataAttribute:

Vi kan skrive vores egen attribut klasse således:

```
using System;
using System.Reflection;

//Vores egen hjemmelavede attribut klasse - skal gemmes i DLL fil:
//Attributten SKAL have en AttributeUsage attribut!!
//Kan anvendes paa alle elementer - assembly, class, metode, felt osv
//Kan anvendes flere gange i samme fil:

[AttributeUsage(AttributeTargets.All,AllowMultiple=true)]

public class DataAttribute : Attribute {

        public string person;
        public string dato;
        public string kommentar;

        //Klassen instantieres i [] attribut erklæringen med 3 parametre
        //ALTID i samme orden!!:
        public DataAttribute(string p,string d,string k){
                person=p;
                dato=d;
                kommentar=k;
```

```
		}

}
```

Klassen DataAttribute ligner enhver anden klasse i C# - bortset fra:

1. Navnet på klassen angives normalt med et suffix 'Attribute' (men dette er ikke tvunget)
2. Klassen arver fra System.Attribute
3. Klassen har en AttributeUsage attribut der definerer om attributten kan anvendes flere gange i samme fil og på hvilke elementer den kan anvendes.

Filen kan kompileres fx til 'dataattribut.dll' og herefter kan den nye attribut anvendes på alle filer om refererer til dataattribut.dll!

Koden kan nu skrives således:

```
// Attributter i C#

using System;
using System.Reflection;
using System.Diagnostics;//JVF attributten Conditional

[assembly: AssemblyVersion("1.2.*")]
[assembly: AssemblyTitle("C# Attributter")]
[assembly: AssemblyDescription("Dette program illustrerer C# attributter")]
[assembly: AssemblyCompany("World Digital Inc.")]
[assembly:CLSCompliant(true)]

[assembly:Data("marie-louise","11-2-2003","Koden skal være klar 1-3-03!")]

public class A {

		private string s="Anden metoden kaldes i A.";

		public A(){}

		[Conditional("TEST")]
		public void test_metode(){
				Console.WriteLine("Test metoden kaldes i A.");
		}
		[Data("erik jensen","1-1-2003","Koden er ikke helt klar endnu!")]
		public void ny_metode(){
				Console.WriteLine("Ny metoden kaldes i A.");
		}
		public void anden_metode(){
				Console.WriteLine(s);
		}


}
class app {

		public static void Main(){
```

```
                Console.WriteLine("Attribut demo.");
                A a=new A();
                a.test_metode();
                a.ny_metode();
                a.anden_metode();

        }
}
```

Vi har her sat nogle nye attributter af typen DataAttribute ind. Som det ses kan suffix'et 'Attribute' overspringes – men **kan** også medtages sådan fx:

```
        [DataAttribute("erik jensen","1-1-2003","Koden er ikke helt klar endnu!")]
```

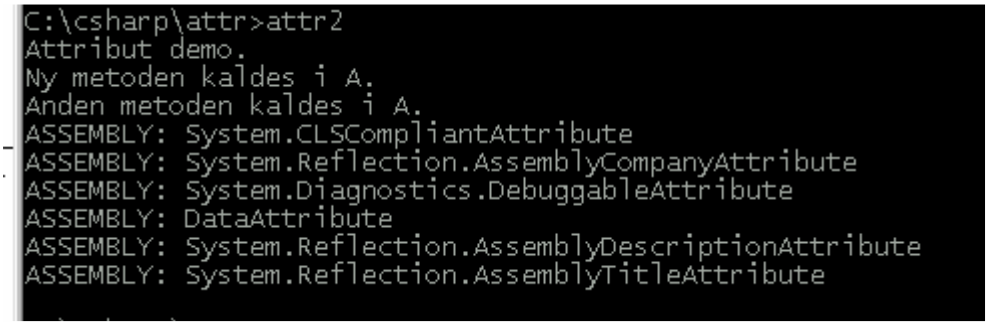
Finde og bruge attribut værdier:

Vi finder de forskellige attributter og deres værdier ved at bruge klasser og metoder i System.Reflection.

Main() kan fx omskrives således:

```
        public static void Main(){

                Console.WriteLine("Attribut demo.");
                A a=new A();
                a.test_metode();
                a.ny_metode();
                a.anden_metode();
                //Find attributter:
                //hent alle attributter for hele assembly'en:
                Attribute[] attributter = Attribute.GetCustomAttributes
(Assembly.GetExecutingAssembly());
                foreach(Attribute at in attributter){
                        Console.WriteLine("ASSEMBLY: "+at.GetType().ToString());
                }


        }
```

Metoden henter nu alle **'Custom'** attributter som er defineret i assemblyen:



Det ses at nogle attributter **automatisk** sættes af systemet: DebuggableAttribute i dette tilfælde. Debuggable har 2 egenskaber som kan sættes til true eller false når et program debugges:

```
DebuggableAttribute: Property: IsJITTrackingEnabled
DebuggableAttribute: Property: IsJITOptimizerDisabled
```

Når et program debugges sættes oftest den sidste property **`IsJITOptimizerDisabled`** til true (dvs ingen automatisk optimering af koden).

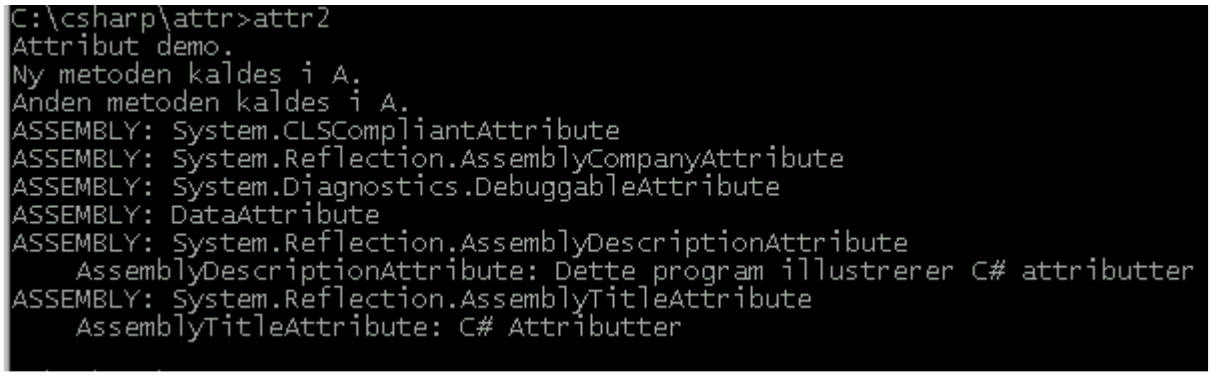
Metoden GetCustomAttributes() – overloadet mange gange – er en **static** metode som kaldes på klassen Attribute!

Det er altså rimeligt simpelt at få udskrevet en liste over **alle** de attributter som er defineret. Det er lidt mere besværligt at hente værdien af disse attributter fordi dette skal ske enkeltvis og med en **eksplicit** angivelse af attributtens **parametre**.

Hvis vi fx vil have udskrevet værdien af attributterne Title og Description skal vi omskrive koden i Main() således:

```
        public static void Main(){

                Console.WriteLine("Attribut demo.");
                A a=new A();
                a.test_metode();
                a.ny_metode();
                a.anden_metode();
                //Find attributter:
                //hent alle attributter for hele assembly'en:
                Attribute[]
attributter=Attribute.GetCustomAttributes(Assembly.GetExecutingAssembly());
                foreach(Attribute at in attributter){
                        Console.WriteLine("ASSEMBLY: "+at.GetType().ToString());
                        if(at.GetType()==typeof(AssemblyTitleAttribute)){
                                AssemblyTitleAttribute
attitel=(AssemblyTitleAttribute)at;

                                //OBS hver enkelt attribut har sine egne
felter/properties!!:
                                Console.WriteLine("    AssemblyTitleAttribute:
{0}",attitel.Title);
                        }
                        else if(at.GetType()==typeof(AssemblyDescriptionAttribute)){
                                AssemblyDescriptionAttribute
atdes=(AssemblyDescriptionAttribute)at;
                                Console.WriteLine("    AssemblyDescriptionAttribute:
{0}",atdes.Description);
                        }

                }


        }
```

Programmet kan nu hente værdierne således:



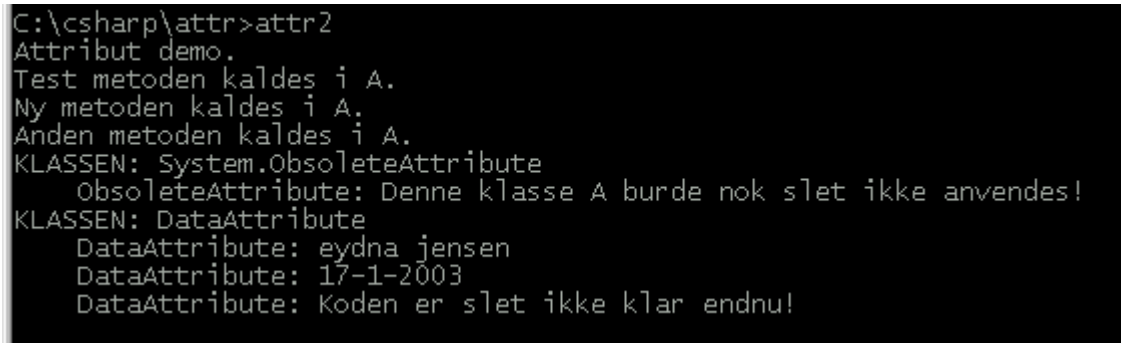
Vi kan hente attributter på klassen A således. Vi sætter først disse attributter forud for klasseerklæringen:

```
[Data("eydna jensen","17-1-2003","Koden er slet ikke klar endnu!")]
[Obsolete("Denne klasse A burde nok slet ikke anvendes!",false)]

public class A {}
```

I Main() kan derefter indsættes:

```
                    Attribute[] attributter=Attribute.GetCustomAttributes(typeof(A));
                    foreach(Attribute at in attributter){
                              Console.WriteLine("KLASSEN: "+at.GetType().ToString());
                              if(at.GetType()==typeof(ObsoleteAttribute)){
                                        ObsoleteAttribute atobs=(ObsoleteAttribute)at;
                                        Console.WriteLine("    ObsoleteAttribute:
{0}",atobs.Message);
                              }
                              else if(at.GetType()==typeof(DataAttribute)){
                                        DataAttribute atdata=(DataAttribute)at;
                                        Console.WriteLine("    DataAttribute:
{0}",atdata.person);
                                        Console.WriteLine("    DataAttribute:
{0}",atdata.dato);
                                        Console.WriteLine("    DataAttribute:
{0}",atdata.kommentar);

                              }

                    }
```

Main() henter nu alle attributter for klassen A og chekker derefter 2 af disse:



Der er 3 public felter i DataAttribute og vi får altså adgang til dem direkte med punktum-notationen:

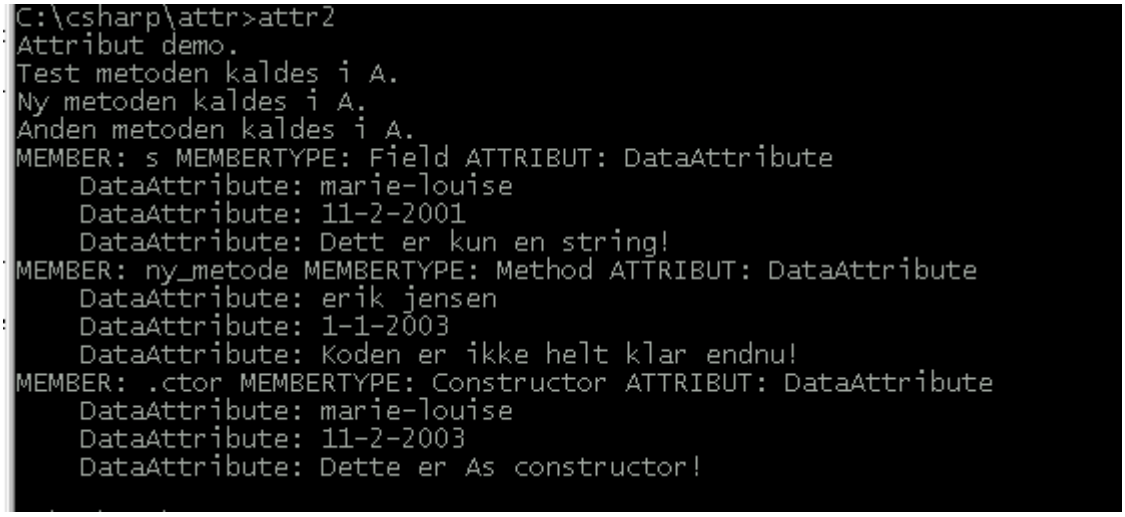
```
Console.WriteLine("    DataAttribute: {0}",atdata.kommentar);
```

Det er altså nødvendigt at kende disse egenskaber ved en attribut for at få adgang til dens data! Med System.Reflection kan man finde attribut klasserne og se hvilke felter de indeholder.

Hvis vi ønsker at hente attributter der er sat på metoder osv inde i A klassen kan vi bruge følgende kode:

```
Type entype=typeof(A);
MemberInfo[] medlemmer=entype.GetMembers();
foreach(MemberInfo m in medlemmer){
        Attribute[] attrib=Attribute.GetCustomAttributes(m);
        foreach(Attribute atr in attrib){
                Console.WriteLine("MEMBER: {0} MEMBERTYPE: {1} ATTRIBUT: {2}",m.Name,m.MemberType, atr.GetType().ToString());
                if(atr.GetType()==typeof(DataAttribute)){
                DataAttribute atdata=(DataAttribute)atr;
                Console.WriteLine("    DataAttribute: {0}",atdata.person);
                Console.WriteLine("    DataAttribute: {0}",atdata.dato);
                Console.WriteLine("    DataAttribute: {0}",atdata.kommentar);

        }

}
```

Vi finder altså først alle medlemmer af klassen A med metoden GetMembers(). Derefter kan vi kalde metoden GetCustomAttributes() på hvert medlem – metoden har mange forskellige signaturer bl.a. en som tager en MemberInfo!
Derefter kan vi – som vi gjorde ovenfor – chekke medlemmet for bestemte attributter! NB kun public felter kan chekkes på denne måde – så feltet er gjort public (public string s):



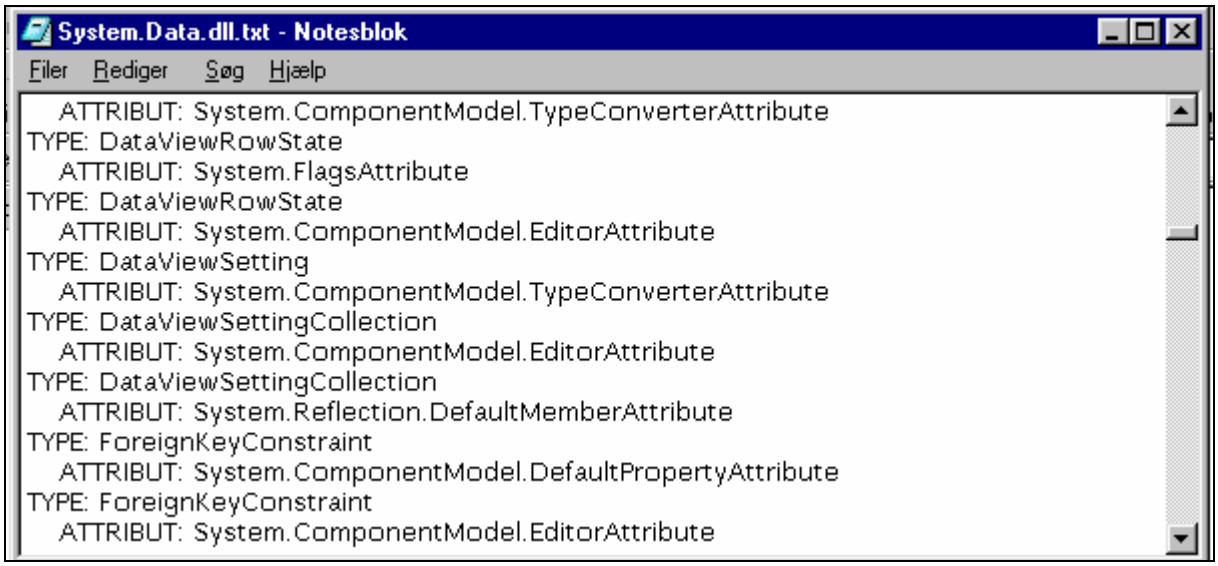
Som en **alternativ** metode til ovenstående kan anvendes denne formel, som bruger den statiske metode **Attribute**.**IsDefined**() som chekker om et klassemedlem har en bestemt attribut:

```
foreach(MemberInfo md in medlemmer){
                                        if(Attribute.IsDefined(md,typeof(DataAttribute))){
                                                Console.WriteLine("IsDefined: {0}
{1}",typeof(DataAttribute).ToString(),md.Name);
                                        }

        }
```

Følgende kode kan udskrive alle attributter for alle klasser i fx System.Data.dll (eller en hvilken som helst anden DLL fil):

```
        public void find_attributter(string s){
                Assembly ass=Assembly.Load(s);
                Type[] typer=ass.GetTypes();
                foreach(Type type in typer){
                        Attribute[] attributter=Attribute.GetCustomAttributes(type);
                        foreach(Attribute at in attributter){
                                Console.WriteLine("TYPE: {0} ",type.Name);
                                Console.WriteLine ("    ATTRIBUT: {0}",
at.ToString());
                        }
                }
        }
```

Her er resultatet **re-directed** til en tekst fil:



UsageAttribute:

Der findes en 'generel' attribut, som kan bruges til at indsætte forklaringer eller kommentarer, nemlig UsageAttribute som kan bruges sådan f.eks.:

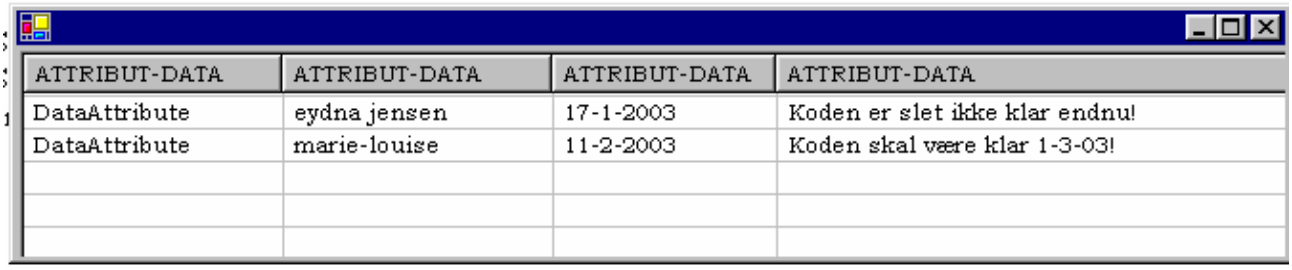
```
[Usage("Computes the density of air")]
  public double GetDensity()
  {
    return p*0.02885/(8.3144*t);
  }
```

Værdien af denne attribut hentes med property'en **Desc** i stil med følgende:

```
    foreach ( UsageAttribute att in attributes )
      {
        Console.WriteLine("      Description: "+att.Desc);
      }
```

Opgaver

1. Skriv nogle nye attributter som du mener kan være nyttige!
2. Skriv et Windows program som kan loade en DLL eller EXE fil og vise dens attributter fx i en ListView kontrol!



ATTRIBUT-DATA	ATTRIBUT-DATA	ATTRIBUT-DATA	ATTRIBUT-DATA
DataAttribute	eydna jensen	17-1-2003	Koden er slet ikke klar endnu!
DataAttribute	marie-louise	11-2-2003	Koden skal være klar 1-3-03!

Konvertere fra C# til COM:

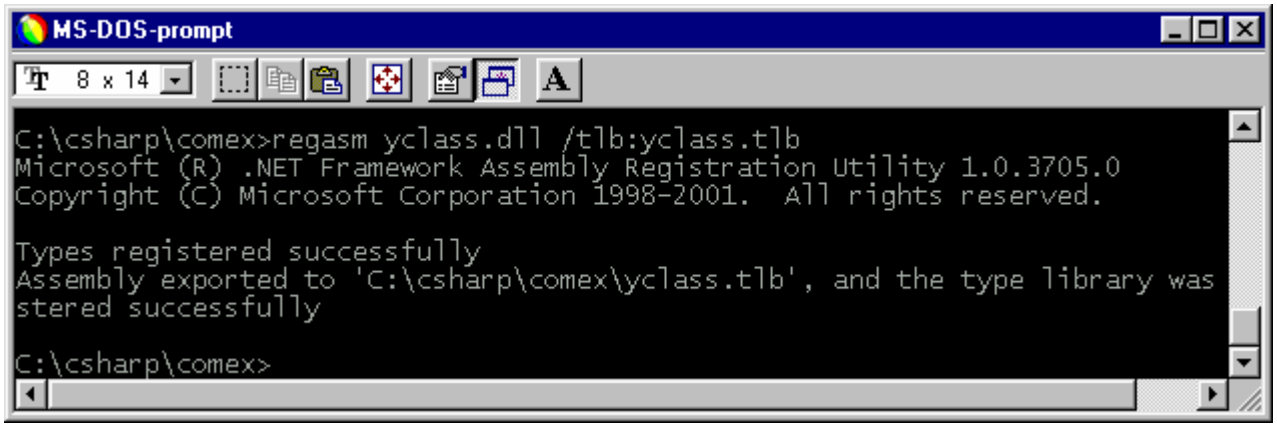
En C# DLL fil kan konverteres til en COM komponent således:

Skriv først en almindelig C# fil som kompileres som en DLL med csc /t:library yclass.cs:

```
using System;

public interface IKlasser{
		string retur();
}

namespace klasser {

public class YClass : IKlasser
{
		public YClass(){}
		public string retur(){
			return "Dette er klassen Y!";
		}

}
```

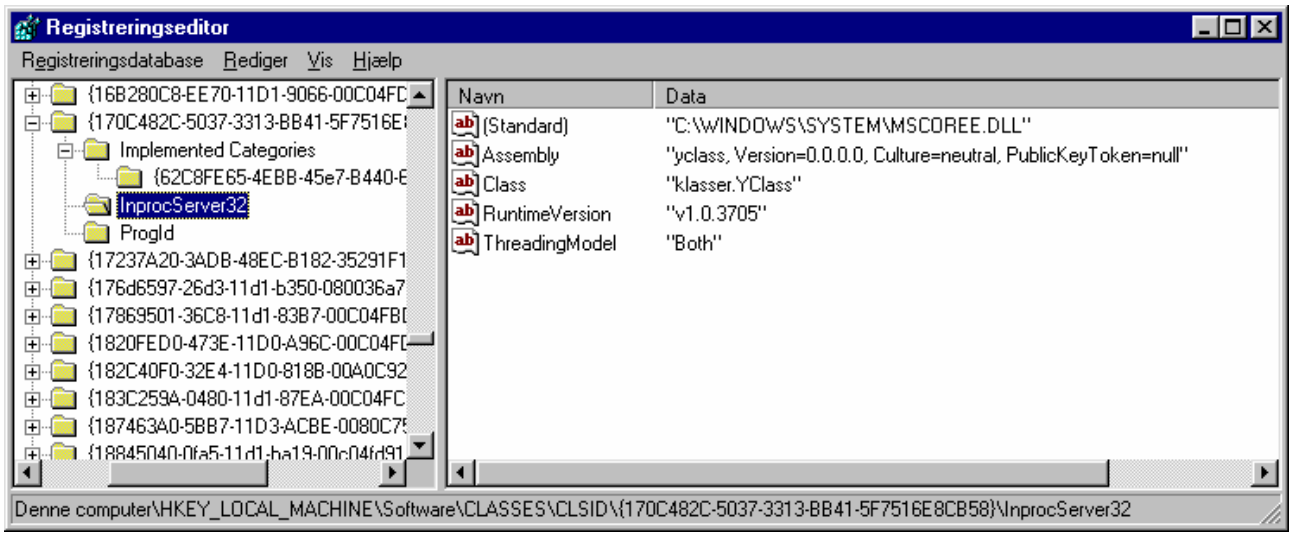
Denne DLL kan nu registreres som COM komponent i System Registrerings databasen således:

regasm yclass.dll /tlb:yclass.tlb



Der oprettes et type-library 'yclass.tlb' (en binær fil der definerer alle typer/klasser i dette library) som gemmes i programmets mappe.

Hvis vi søger i Windows registrerings databasen efter 'YClass' finder vi – blandt andet (for den nye klasse er registreret flere steder!):

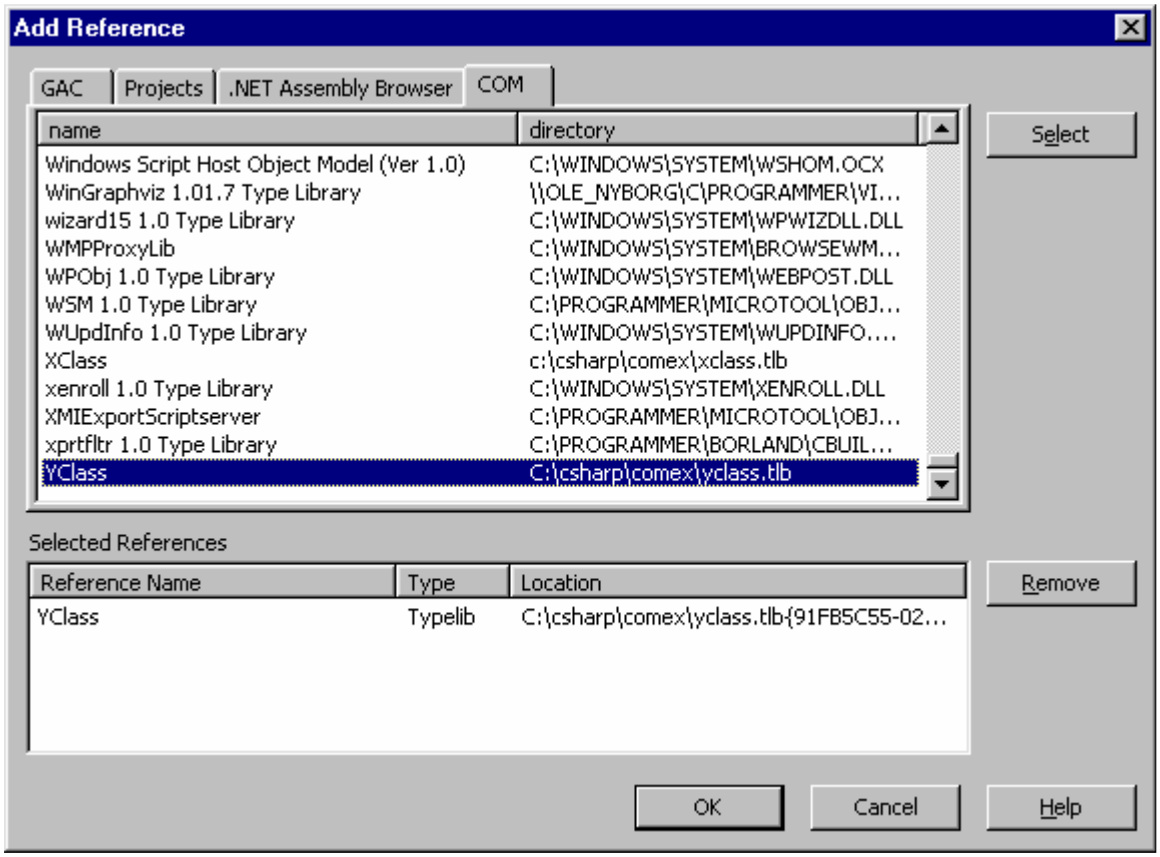


C# komponenten er nu registreret med en CLSID (klasse ID), ProgId ('klasser.YClass') osv.

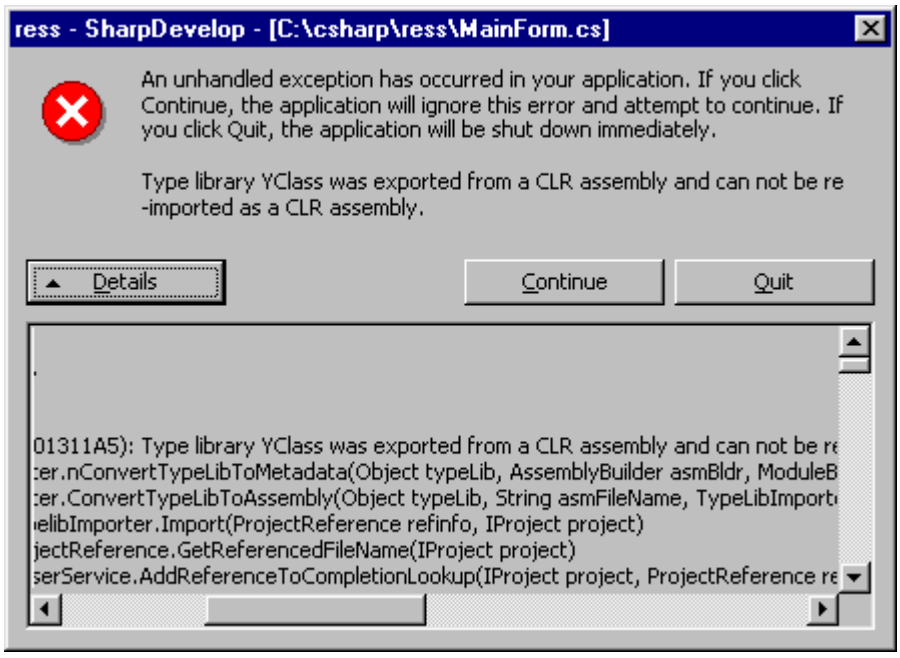
Komponenten er registreret i nøglen:

HKEY_LOCAL_MACHINE\Software\CLASSES\CLSID\{170C482C-5037-3313-BB41-5F7516E8CB58}

Hvis vi ønsker at tilføje en COM reference til et C# projekt kan vi nu finde den nye YClass komponent:



YClass kan dog **ikke** tilføjes som reference, fordi den oprindeligt er eksporteret fra en C# assembly!!:



Vi kan også anvende klasser i System.Runtime.InteropServices (fx klassen TypeLibConverter) til at konvertere frem og tilbage mellem C#/.NET og COM.

Dynamisk kald af COM komponent:

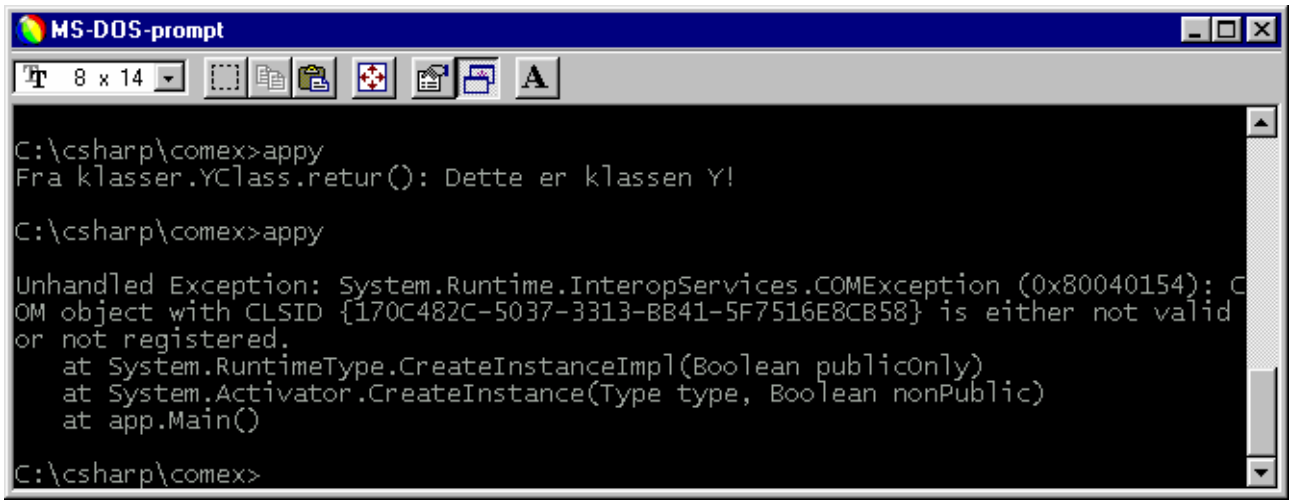
Vores nye centralt registrerede COM komponent 'klasser.yClass' kan kaldes dynamisk i C# kode således:

```
using System;
using System.Reflection;
using System.Runtime.InteropServices;

class app {

		public static void Main(){

				//Vi kalder den registrerede YClass – søger i registrerings databasen:
				Type type=Type.GetTypeFromProgID("klasser.YClass");

				//objektet oprettes dynamisk/runtime, i stedet for new:
				object o=Activator.CreateInstance(type);

				//kald metoden retur() i objektet – late binding:
				string
s=type.InvokeMember("retur",BindingFlags.Default|BindingFlags.InvokeMethod,null,o,null).ToString();
				Console.WriteLine("Fra klasser.YClass.retur(): {0}",s);



		}
}
```

Vi kan instantiere komponenten dynamisk ved at gå ind i system registrerings databasen og finde det objekt som har 'klasser.YClass' som **'ProgId'**! (I virkeligheden går programmet ind og leder efter en bestemt **CLSID**!).

YClass komponenten oprettes altså runtime - også kaldet **'late binding'**:

At vi kan gøre dette, skyldes **KUN** at komponenten er registreret!! Hvis vi – som test – **sletter** ovenstående nøgle for klasser.YClass i databasen og derefter prøver at starte applikationen får vi en fejl:



Vi kan se at systemet faktisk leder efter det rigtige CLSID (klasse ID), men ikke kan finde det nogen steder!

En COM Person klasse:

Et lidt mere anvendeligt eksempel på det samme er en Person klasse skrevet således i C#:

```
// eksempel paa eksport af C# klasse til COM:
//csc /t:library person.cs
//regasm person.dll /tlb:person.tlb:

using System;
using System.Runtime.InteropServices;

namespace Personer {

        public interface IPersoner {

                string Navn {get;set;}
                string Telefon {get;set;}
        }

        [ClassInterface(ClassInterfaceType.AutoDual)]

        public class Person : IPersoner {

                private string navn,telefon;

                public Person(){}

                public string Navn {

                        get{
                                return navn;
                        }
                        set {
```

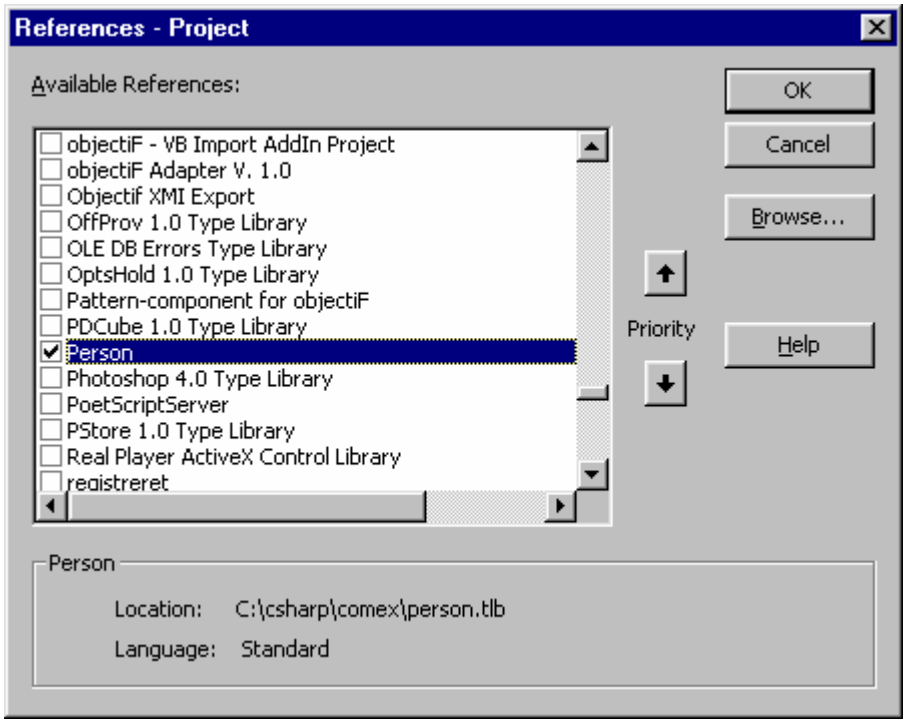
```
                                        navn=value;
                    }
            }
            public string Telefon {

                    get{
                                        return telefon;
                    }
                    set {
                                        telefon=value;
                    }
            }

    }
}
```

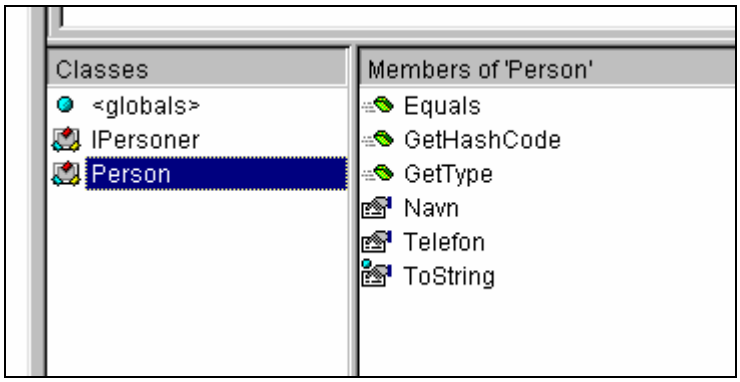
Vi har her sat en attribut på klassen:

```
[ClassInterface(ClassInterfaceType.AutoDual)]
```

der sikrer, at C# klassen gnidningsfrit kan registreres som COM klasse. NB Det er strengt taget ikke nødvendigt at erklære et interface – men i mange tilfælde er det nødvendigt hvis klassen siden skal bruges som COM objekt.

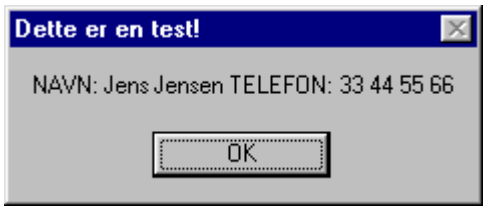
Vi kan nu bruge denne Person klasse fx i programmering i Visual Basic (VBA) i et Microsoft Office Program:





Hvis vi kopierer DLL og TLB filen til Windows mappen, kan vi bruge Person klassen i en windows script fil person.vbs således:

```
Dim obj
Set obj = CreateObject("Personer.Person")
obj.navn="Jens Jensen"
obj.Telefon="33 44 55 66"
MsgBox "NAVN: "& obj.Navn & " TELEFON: " & obj.Telefon,,"Dette er en test!"
```



COM objekter (eller konverterede C# objekter) kan også indsættes i en HTML fil med en <object classid=...> tag.

COM+ eller ComPlus:

En række udvidede typer service i Windows teknologien 'COM+' kan refereres fra C#. Vi kan således registrere en C# DLL som en COM+ komponent således:

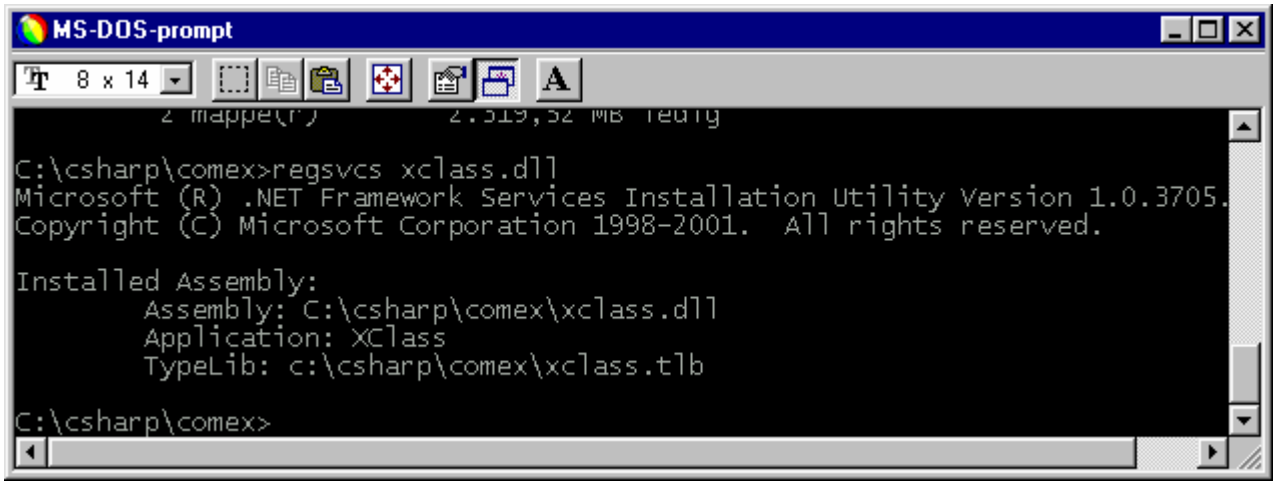
Vi starter med en almindelig C# fil som kompileres som xclass.dll:

```
// eksempler: C# og COM og COM+

using System;
using System.Reflection;

//COM+:
using System.EnterpriseServices;

[assembly:ApplicationName("XClass")]
```

```
//NB skal have en Public Key for at kunne registreres!:
[assembly:AssemblyKeyFile("eks.snk")]

namespace complus {

public class XClass:ServicedComponent
{
                public XClass(){}

}
```

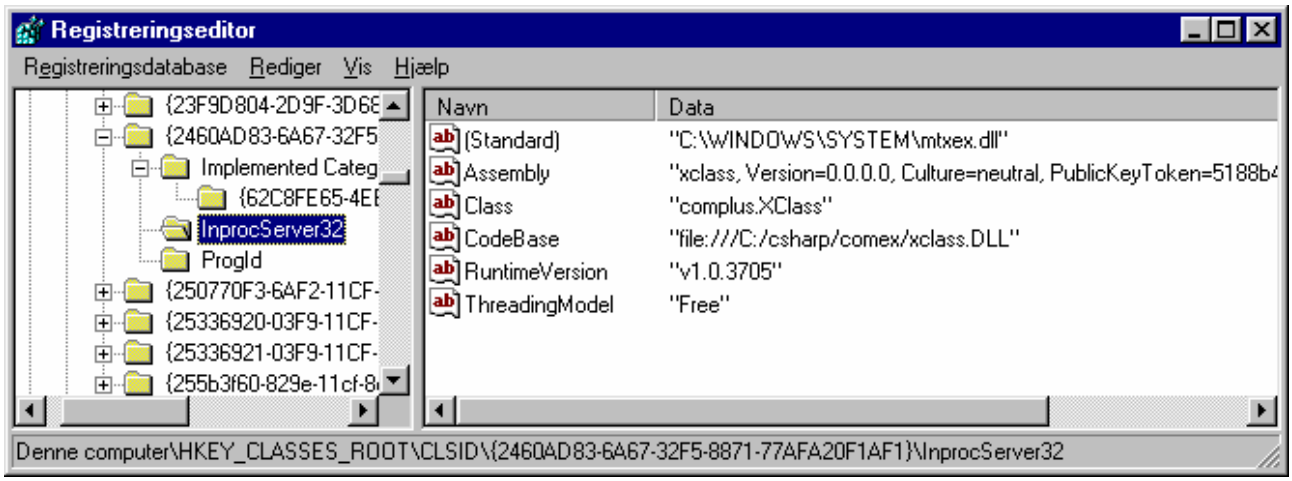
COM+ klasserne ligger i System.EnterpriseServices som rummer et væld af klasser. Filen xclass.dll kan nu registreres med:

regsvcs xclass.dll



Vi har nu fået vores C# klasse registreret i Windows System registrerings databasen! Samtidigt har vi fået oprettet et type-library 'xclass.tlb' (en binær fil som rummer klasse definitioner).

Hvis vi i registreringsdatabasen søger efter vores XClass ses dette:



Vores C# dll er nu registreret og har fået et 'CLSID' (klasse ID) og et 'ProgId' (namespace.klassenavnet):