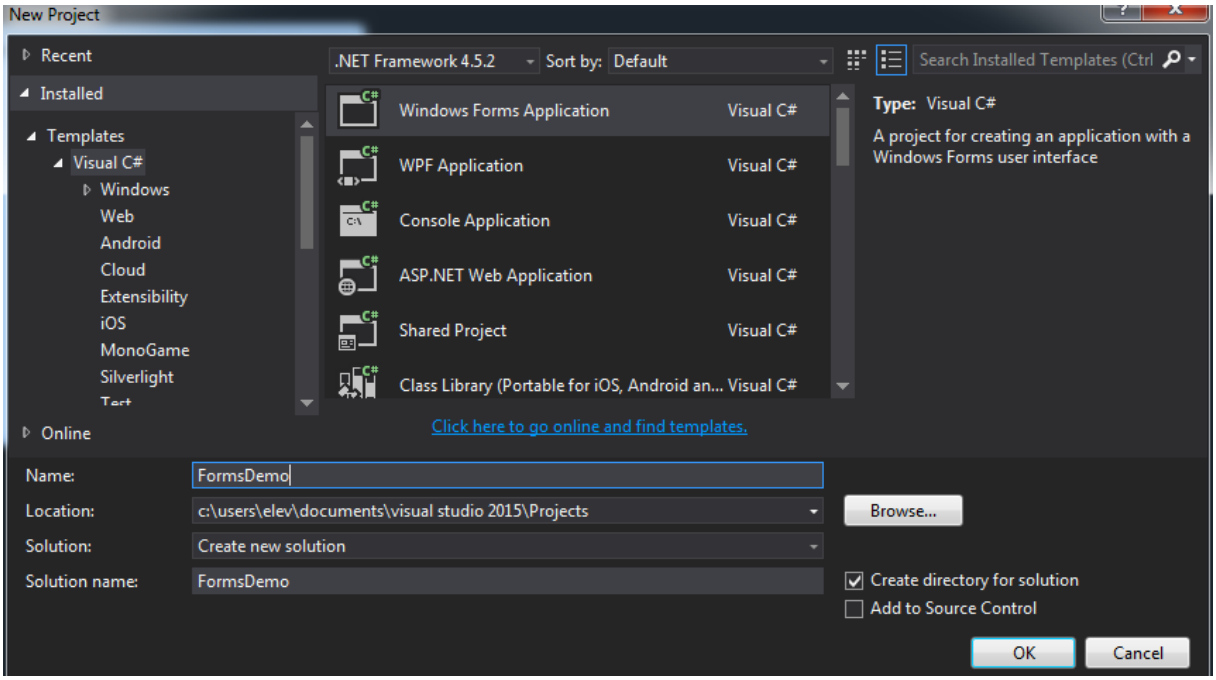
C#- Windows Forms, del 1

Från konsol till fönsterapplikationer

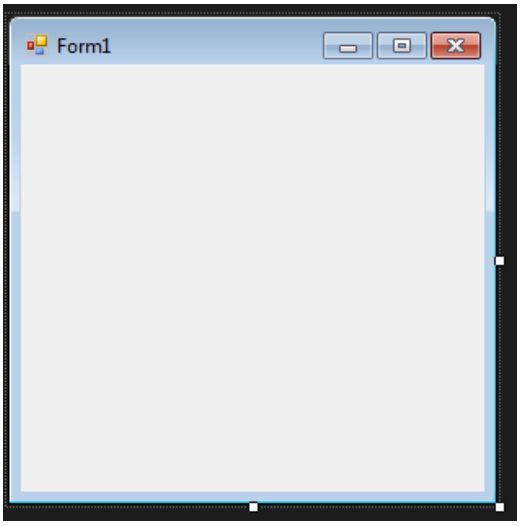
Vi ska nu övergå från konsolapplikationer till fönsterapplikationer. Detta gör vi genom att skapa Windows Forms projekt. Vi kommer använda oss av händelse baserad programmering, det vill säga att när vi trycker på en knapp så händer det något. I detta dokument kommer jag gå igenom hur vi skapar ett Windows Forms projekt, de vanligaste kontrollerna, hur man döper dessa och deras inbyggda lyssnare.

Skapa Windows Form projekt

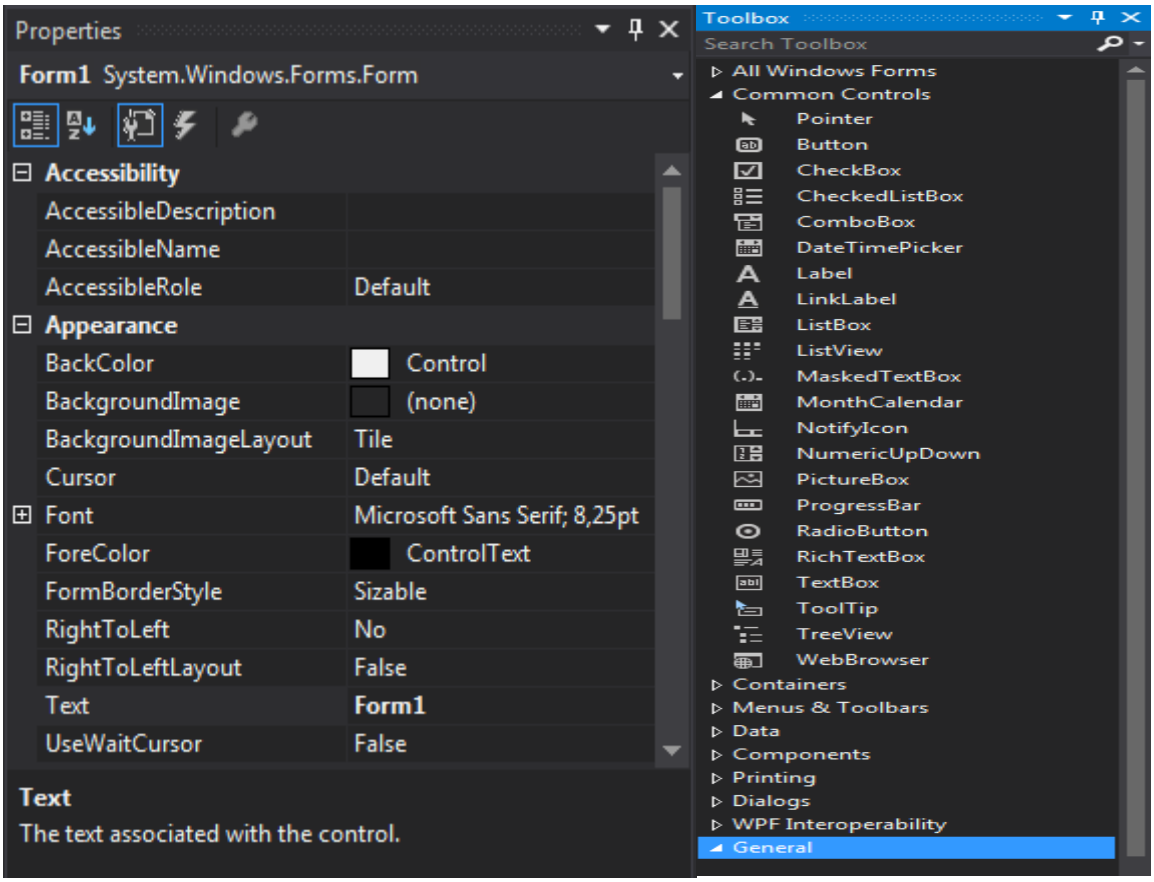
Vi börjar med att skapa ett nytt projekt, väljer Windows Forms och ger den ett namn.



Nu kommer en designer dyka upp med några nya tabbar att hålla koll på. Framför er ser ni er form, denna kommer vi dra in kontroller och designa utseendet för.



Under solution explorer ser ni en ny tab för Properties, och på vänster sida har ni Toolbox.

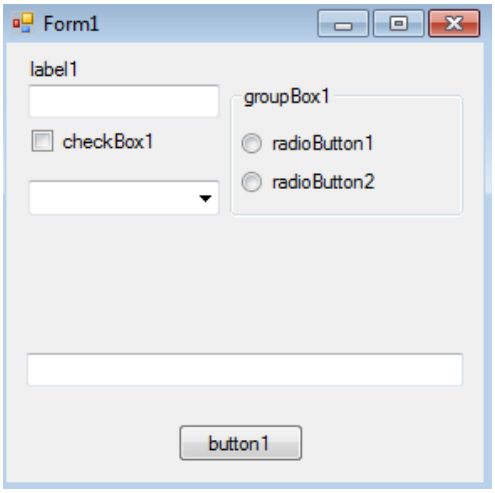


(Har ni inte toolbox så kan ni få fram den via View > Toolbox)

Toolbox

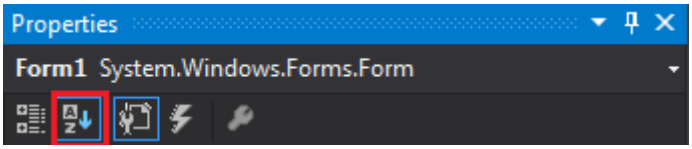
I denna del kommer vi använda oss av **Buttons**, **Textboxes**, **Checkbox**, **Label**, **Combobox**, **GroupBox** samt **Radio Buttons**.

Vi börjar med att dra ut två **textboxes**, en **button**, en checkbox, en **label**, en **combobox**, en **groupbox** samt två radio **button** som vi placerar inuti vår **groupbox**. Som ni märker så får vi hjälp med att placera ut dessa symmetriskt utan att behöva räkna pixlarna själv. Därefter drar vi i objektens fönster för att ändra storleken så att allt passar i vår form.

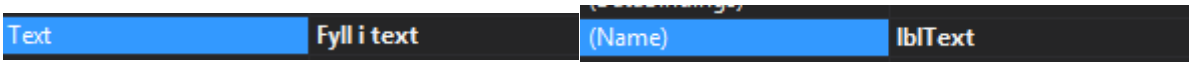


Properties

Nu ska vi ändra namn och text för våra kontroller i **Properties**. Vi börjar med att ändra vår **view** så att vi får alla egenskaper i alfabetisk ordning.



Namn-egenskapen för en kontroll är vårt variabelnamn för kodningen och text-egenskapen är texten som syns för användaren. Vi börjar med att ändra namn och text för vår **label**. Vi börjar med att leta på ”**(name)**” egenskapen och ger den namnet ”lblText”, sedan letar vi på text-egenskapen och skriver in ”Fyll i text”.

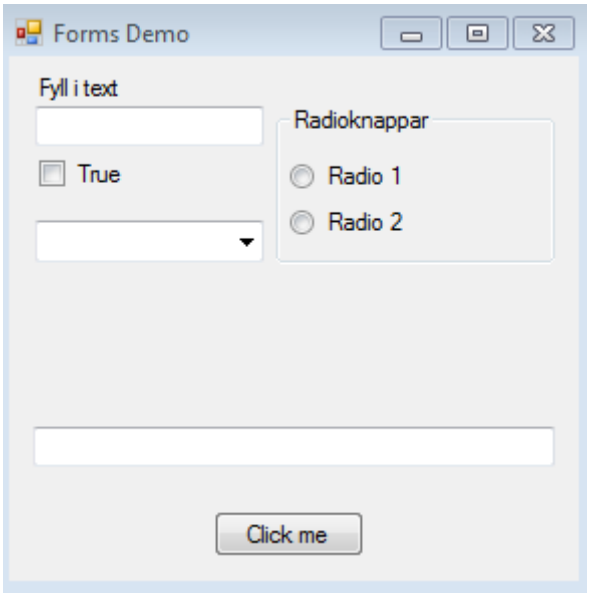


Enligt sed när man döper en kontroll börjar man namnet med en förkortning av kontrollen och följer med dess syfte. I detta fall blev det ”lblText”, lbl för label och Text för syfte.

Döp nu resten av kontrollerna.

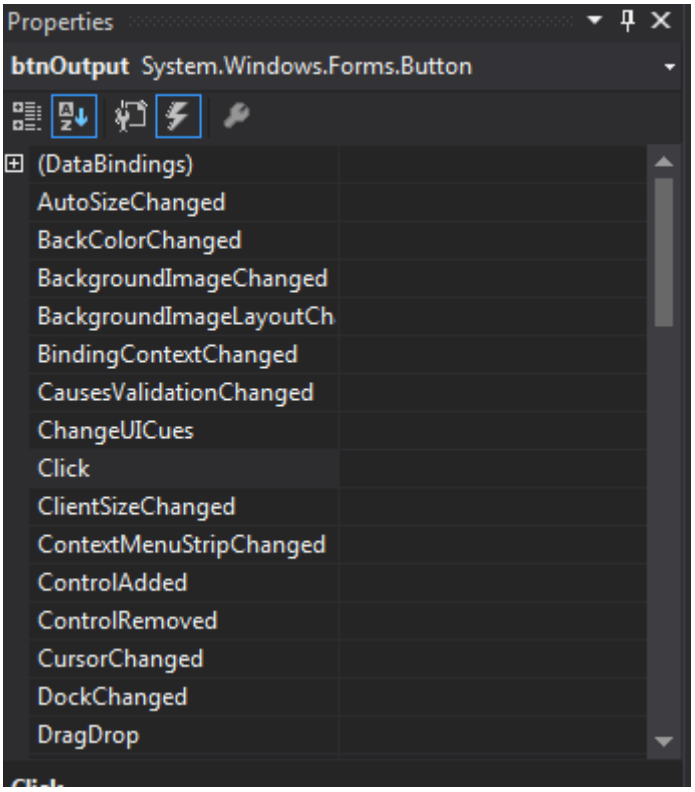
- **Textbox1** namn: **txtInput**, här ska text-egenskapen vara tom
- **Textbox2** namn: **txtOutput**, text-egenskapen tom
- **Checkbox1** namn: **chkTrue**, text: True
- **Combobox1** namn: **cboLista**, text tom
- **Groupbox1** namn: **gbRadio**, text: Radioknappar
- **RadioButton1** namn: **optRadio1**, text: Radio 1
- **RadioButton2** namn: **optRadio2**, text: Radio 2
- **Button** namn: **btnOutput**, text: Click me
- **Form1** namn: **frmMain**, text: Forms Demo

Resultat:



Properties Events

Varje kontroll har händelser(events) som vi kan anropa kod på. Vilka händelser som finns för de olika kontrollerna finns i **properties** under Events-tabben(blixt-ikonen). **Button**-events:



Vi ska nu använda oss av **Click**-händelsen, denna skapar vi genom att antingen dubbelklicka i listan eller på knappen i formen. **Notis:** Dubbelklickar ni på andra kontroller förutom exempelvis knapp så kommer ni få en **click**-händelse för dessa också vilket vi ibland kan råka få av misstag. Så var noga med var ni trycker på. I slutet visar jag hur man tar bort händelser om man råkat av misstag lägga till en.

Nu när vi dubbelklickat antingen på knappen eller i listan får vi fram detta:

```
private void btnOutput_Click(object sender, EventArgs e)
{

}
```

Nu skriver vi en rad kod för att demonstrera hur **textboxes** fungerar. Vi börjar med att skriva **txtOutput.Text** = **txtInput.Text.ToUpper**(); När vi arbetar med **textboxes** och vill ha värdet eller sätta värde så måste vi använda oss av ”**.Text**” för att peka mot strängen. Kod raden vi nyss skrev tar texten i **txtInput**, gör om till stora bokstäver och lägger in den i **txtOutput**. Testa nu att köra programmet och skriv en rad i **txtInput** och tryck på knappen.

Initiera våra kontroller

Som ni märkte när ni körde programmet så var vår **combobox** tom och ingen av våra **radiobuttons** var valda. Detta ska vi ändra på i vår kod och skapa vår egen metod initierar våra kontroller när programmet startas.

```
public frmMain()
{
    InitializeComponent();

    InitializeControls(); // Initiera våra controls
}

// Initierar startvärden till våra controls
private void InitializeControls()
{
    optRadio1.Checked = true;
    chkTrue.Checked = false;

    initCheckbox();
}

// Tilldelar items till vår combobox
private void initCheckbox()
{
    for (int i = 0; i < 3; i++)
    {
        cboLista.Items.Add(i + 1); // lägg till 3 items(siffror)
    }
    cboLista.SelectedIndex = 0; // Sätt SelectedIndex till första indexet
}
```

Här har vi tre metoder. **frmMain**() är metoden som körs när programmet startas. I den här metoden har vi anropat vår egen metod **InitializeControls**() som vi sätter värden till våra kontroller i. Vi vill att vår första radioknapp ska vara ikryssad och att vår checkbox inte ska vara ikryssad. Därefter anropar vi en metod vars uppgift är att lägga in värden i vår **combobox**, det vill säga att vi lägger in objekt i listan. Därefter använder vi **SelectedIndex** för att sätta valt objekt att vara första positionen.

Lyssnare till radioknapparna och vår checkbox

Vi börjar med att använda en händelse för en av våra radioknappar som heter **CheckedChanged**. I denna metod kollar vi om **optRadio1** är ikryssad med hjälp av egenskapen **Checked** och i så fall vill vi att bakgrundsfärgen på vår form ska vara standardfärgen. Är den inte ikryssad, det vill säga att vi har kryssat i **optRadio2**, så ändrar vi bakgrundsfärgen till blå. Koden ser ut så här:

```
// När värdet i optRadio1 ändras
private void optRadio1_CheckedChanged(object sender, EventArgs e)
{
    // Om radio1 är checked, använd standardbakgrundsfärgen
    // annars ändra till blå bakgrundsfärg
    if (optRadio1.Checked)
        this.BackColor = frmMain.DefaultBackColor;
    else
        this.BackColor = Color.Blue;
}
```

(This hänvisar till formen)

Vi använder samma händelse för vår **checkbox** men denna gång så vill vi att ifall användaren kryssar i denna så vill vi inaktivera **txtInput** objektet. Samma här kollar vi om den är ikryssad med egenskapen **Checked** och använder oss av egenskapen **Enabled** för att ge ett värde. Är den ikryssad vill vi sätta **Enabled** till **false**, och om den inte är ikryssad vill vi att värdet ska vara **true**. Koden ser ut så här:

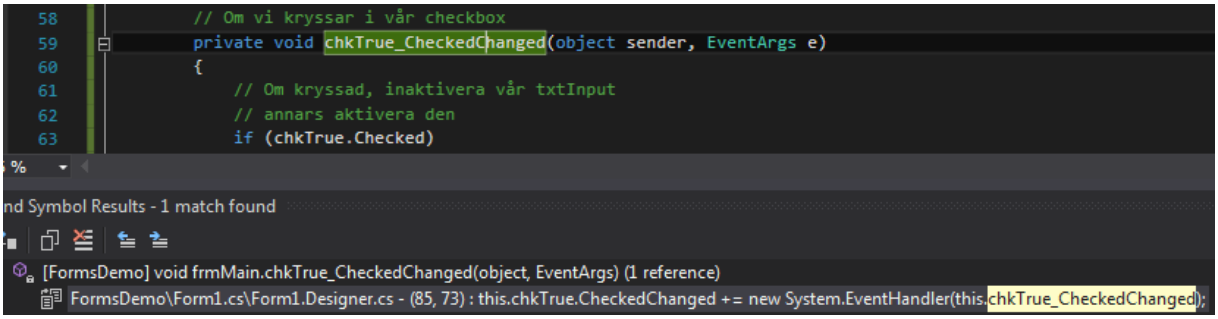
```
// Om vi kryssar i vår checkbox
private void chkTrue_CheckedChanged(object sender, EventArgs e)
{
    // Om kryssad, inaktivera vår txtInput
    // annars aktivera den
    if (chkTrue.Checked)
        txtInput.Enabled = false;
    else
        txtInput.Enabled = true;
}
```

Del 1 är nu klar! Kör programmet och se vad som händer.

Hjälp! Jag dubbelklickade på fel händelse!

Om vi råkar skapa fel händelse och vi tar bort koden för händelsen i Form.cs så kommer vi få ett fult felmeddelande och vår form är borta. Detta händer för att själva koden för händelsen inte är borta, utan bara metoden vi vill ska anropas när händelsen inträffar. Vi måste därför in i Form.Designer.cs och ta bort händelsen själva.

Innan vi tar bort metoden i Form.cs kan vi högerklicka på metodnamnet och trycka find all references och dubbelklicka referensen. (Leta efter += new System.EventHandler…)



Nu kommer vi se koden för just det objektet framför oss och vi vill ta bort +=new System.EventHandler(this.objektNamn_Händelsenamn);

```
//
// chkTrue
//
this.chkTrue.AutoSize = true;
this.chkTrue.Location = new System.Drawing.Point(15, 51);
this.chkTrue.Name = "chkTrue";
this.chkTrue.Size = new System.Drawing.Size(48, 17);
this.chkTrue.TabIndex = 4;
this.chkTrue.Text = "True";
this.chkTrue.UseVisualStyleBackColor = true;
this.chkTrue.CheckedChanged += new System.EventHandler(this.chkTrue_CheckedChanged);
```

Därefter kan ni ta bort koden för metoden i vår Form.cs och allt är löst.