



**Linnéuniversitetet**

Kalmar Vaxjö

Instruktion

# Räkna med ASP.NET Web Forms *(Old School)*

Introduktionsuppgift



*Författare:* Mats Looch

*Kurs:* ASP.NET Web Forms

*Kurskod:* 1DV406

## Upphovsrätt för detta verk

Detta verk är framtaget till kursen ASP.NET Web Forms (1DV406) vid Linnéuniversitetet.

### Du får använda detta verk så här:

Allt innehåll i detta verk av Mats Loock, förutom Linnéuniversitetets logotyp, symbol och kopparstick, är licensierad under:



Creative Commons Erkännande-IckeKommersiell-DelaLika 2.5 Sverige licens.  
<http://creativecommons.org/licenses/by-nc-sa/2.5/se/>

### Det betyder att du i icke-kommersiella syften får:

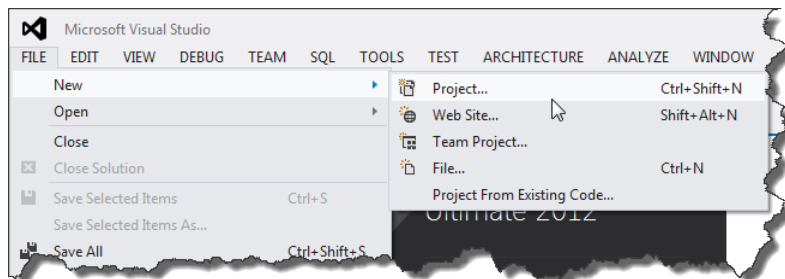
- kopiera hela eller delar av innehållet
- sprida hela eller delar av innehållet
- visa hela eller delar av innehållet offentligt och digitalt
- konvertera innehållet till annat format
- du får även göra om innehållet

Om du förändrar innehållet så ta inte med Linnéuniversitetets logotyp, symbol och/eller kopparstick i din nya version!

## Innehåll

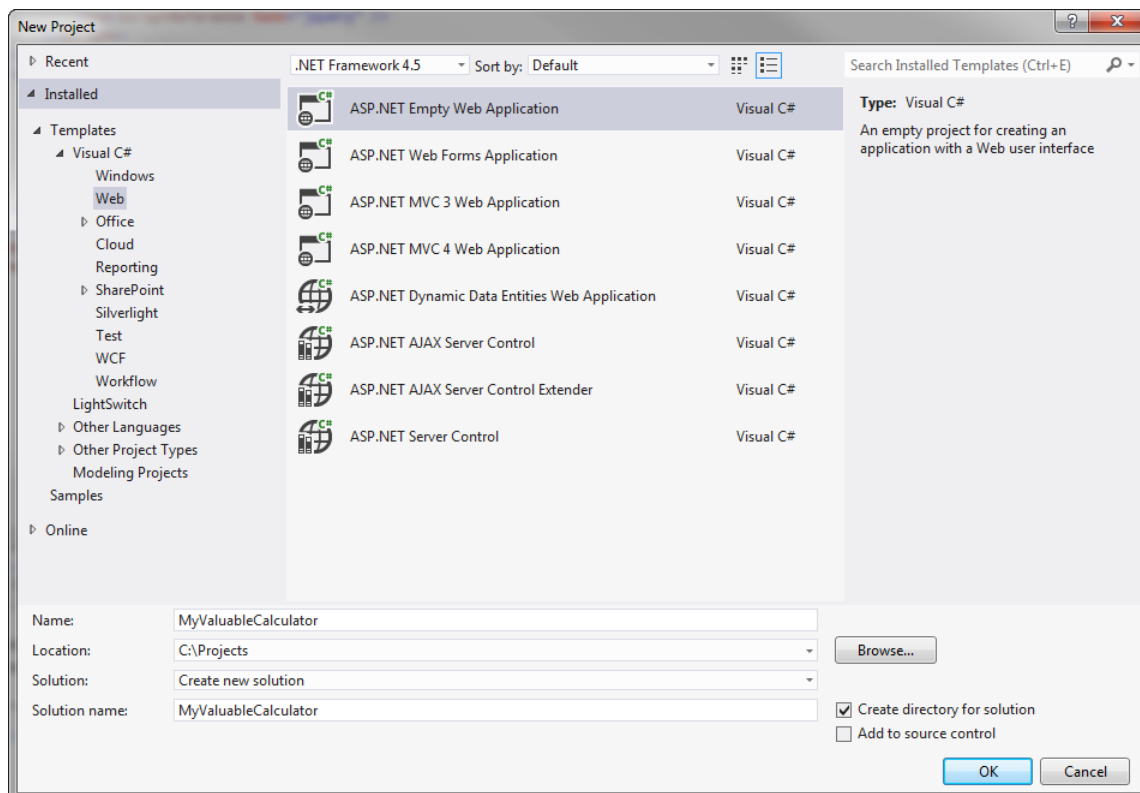
Du ska följa ”steg för steg”-instruktionen i denna introduktionsuppgift och skapa en ”ASP.NET Web Forms (*Old School*)”-applikation som kan addera två heltal en användare matar in i två textfält.

1. Starta Visual Studio 2012.
2. Du ska skapa ett nytt projekt för en webbapplikation, välj därför **File ► New ► Project...**



Figur 1.

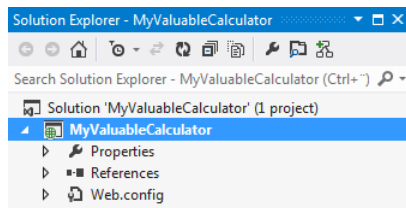
3. Dialogrutan **New Project** visas.



Figur 2.

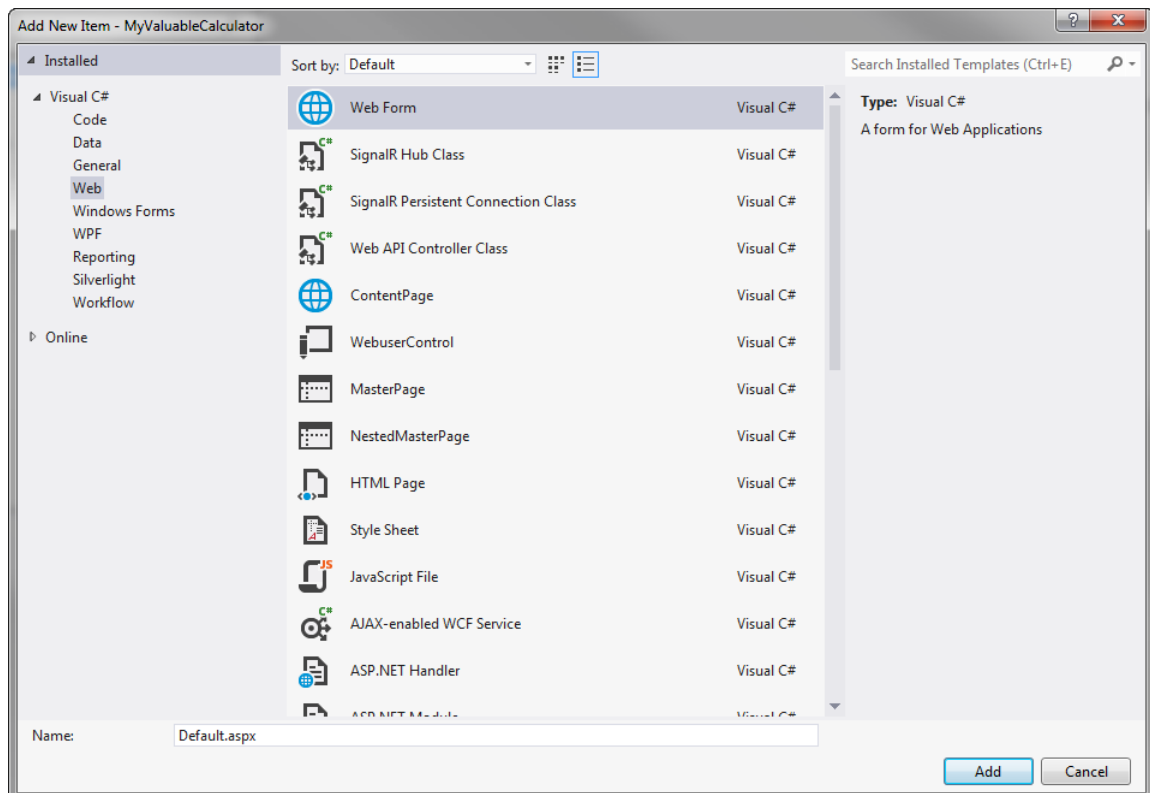
- a) Under **Installed ► Templates ► Visual C#** markera **Web**.
- b) Kontrollera så att **.NET Framework 4.5** visas i den nedrullningsbara listrutan.
- c) Markera **ASP.NET Empty Web Application**.
- d) Vid **Name** skriver du in projektets namn (MyValuableCalculator).
- e) Ange vid **Location** en lämplig katalog där projektet ska sparas (C:\Projects).
- f) Se till att kryssrutan vid **Create directory for solution** är markerad.

4. Visual Studio skapar ”*solution*” och projekt för den nya webbapplikationen. Den nya webbplatsen innehåller endast ett fåtal filer, som t.ex. Web.config.



Figur 3.

5. Högerklicka på projektnamnet i fönstret **Solution Explorer** och välj **Add ► New Item....** Dialogrutan **Add New Item** visas.

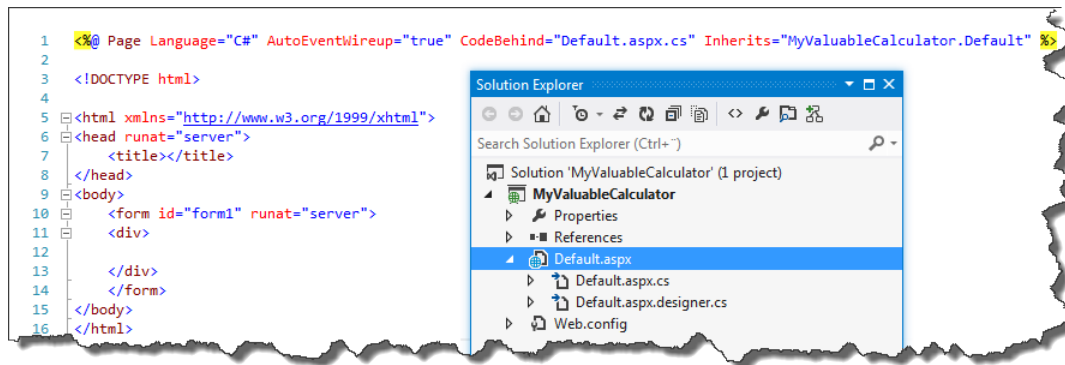


Figur 4.

- a) Välj **Visual C#** under **Installed**.
- b) Välj **Web**.
- c) Välj **Web Form**.
- d) Se till att de står Default.aspx vid **Name**.
- e) Klicka på knappen **Add**.

Webbformuläret **Default.aspx** och dess ”code-behind”-fil **Default.aspx.cs** skapas och läggs till projektet. (I och med att du initialt skapade ett projekt skapas även filen **Default.aspx.designer.cs** som används av Visual Studio men används inte direkt vid programmeringen av webbapplikationen.)

Webbformuläret kallade du **Default.aspx** vilket är det namn startsidans webbformulär i en ”Web Forms”-applikation brukar ges enligt den konvention som finns.



Figur 5.

6. För att en användare av webbapplikationen ska kunna addera två tal använder du dig lämpligen av två textfält, med mellanliggande +-tecken, och en kommandoknapp.

Dra ut två **TextBox**-kontroller (de renderas som textfält) från fliken **Standard** i fönstret **Toolbox** och placera dem i div-elementet som finns i det enda form-elementet. Det får bara finnas ett form-element och alla kontroller måste placeras i det.

Dra även ut en **Button**-kontroll och placera den på lämpligt ställe.

För att kunna presentera resultatet av additionen av talen använder du dig lämpligen av en **Label**-kontroll, som renderas ut som ett span-element.

Då du lagt till en kontroll kan dess egenskaper redigeras med hjälp av fönstret **Properties**. Det är lämpligt att ge en kontroll ett beskrivande ID då det är ID:et du använder i "code behind"-filen för att referera till kontrollen.

Om fönstret **Properties** inte visas kan du visa det genom att välja **View ► Properties Window**. Självklart kan du även redigera ID:et direkt i aspx-filen.

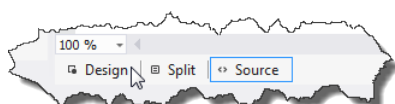
Passa även på att ge title-elementet innehållet **Min värdefulla kalkylator**.



Figur 6.

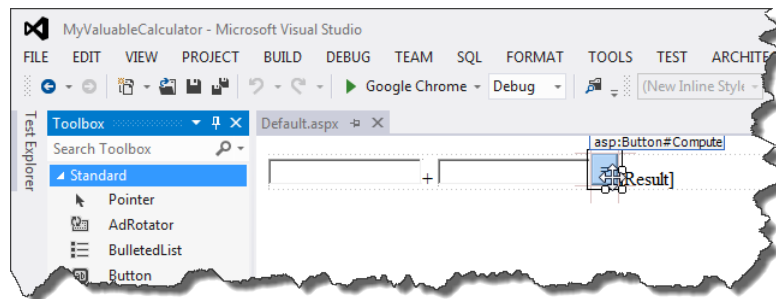
7. Då användaren klickar på kommandoknappen ska formulärets data skickas tillbaka till webbservern. Genom att skapa en hanterarmetod knuten till **Button**-kontrollens händelse **Click** sker en "postback" då användaren klickar på knappen.

För att skapa en hanterarmetod till händelsen **Click** byter du från **Source** till **Design**. Du kan göra det genom att klicka på knappen **Design** eller genom att trycka Shift + F7.




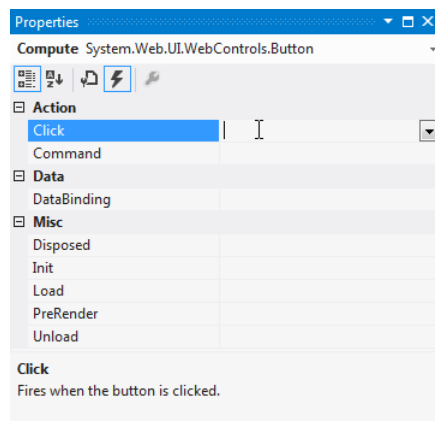
Figur 7.

Markera sedan **Button**-kontrollen.



Figur 8.

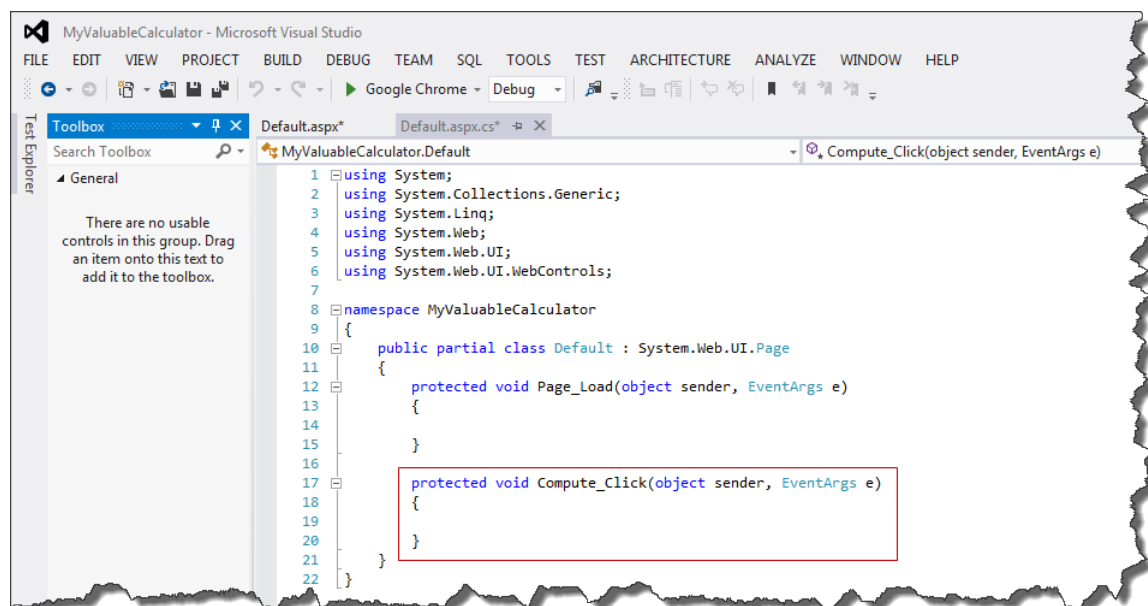
Du kan skapa hanterarmetoden till händelsen **Click** på två sätt. Antingen dubbelklickar du på knappen eller så klickar du på knappen **Events**  i fönstret **Properties** för att därefter dubbelklicka jämte **Click**.



Figur 9.

Oavsett vilken metod du väljer kommer Visual Studio att automatgenerera en hanterarmetod i "code behind"-filen `Default.aspx.cs` och därefter öppna filen.

8. Hanterarmetoden som genereras får ett namn sammanslaget av ID:et på kontrollen (Compute) samt namnet på händelsen (Click), vilket resulterar i `Compute_Click`.



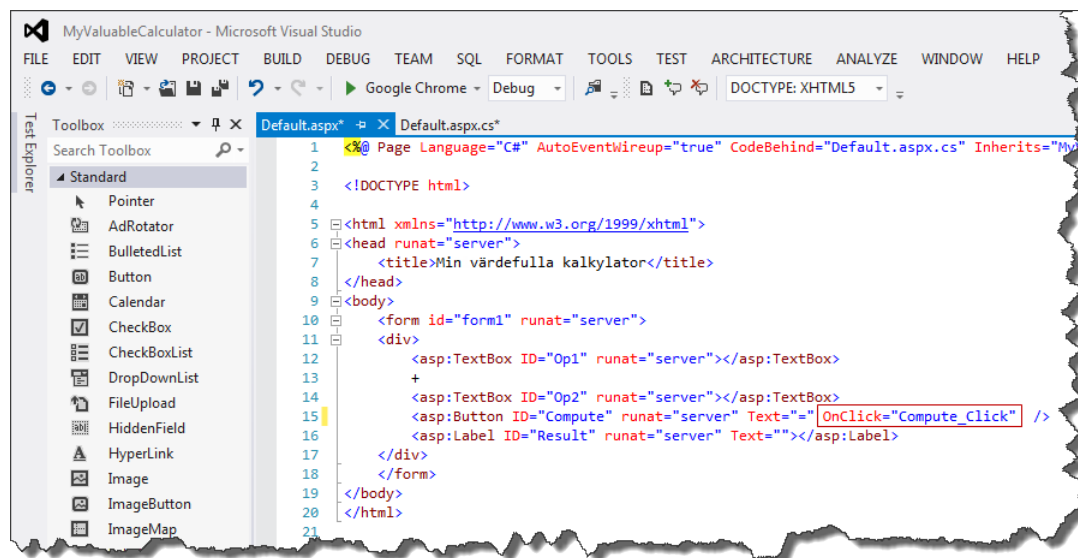
Figur 10.

Det är i hanterarmetoden `Compute_Click` du ska skriva koden du vill ska köras då användaren

klickar på kommandoknappen.

(Du behöver inte i den här applikationen bry dig om metoden `Page_Load`. Denna metod körs alltid då en sida genereras, oavsett om det är fråga om en "get" eller "post".)

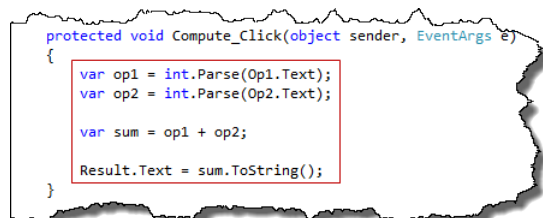
Även `Default.aspx` modifierades då hanterarmetoden skapades. Genom attributet `OnClick` knyts kontrollen `Compute` till hanterarmetoden `Compute_Click`.



Figur 11.

9. Textrutorna innehåller text som måste tolkas från text till heltal av typen `int`. Talen ska därefter adderas och summan presenteras med hjälp av **Label**-kontrollen.

**TextBox**- och **Label**-kontroller har egenskapen `Text` som du använder dig av för att hämta respektive sätta textinnehållet i kontrollerna.

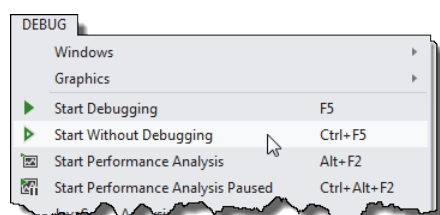


Figur 12.

10. Hur många gånger har du hittills sparat de ändringar du gjort? Inte någon gång? Huuu! Hög tid att klicka på knappen **Save All** som sparar alla öppna filer som har osparade ändringar. Ta för vana att spara ofta. `Ctrl + S`, som sparar filen som har fokus, ska sitta i ryggmärgen.
11. Webbapplikationen är nu färdig att provköras. Du kan köra en applikation på flera olika sätt. Enklast är kanske att trycka `Ctrl + F5`, som startar applikation utan "debug"-läge.

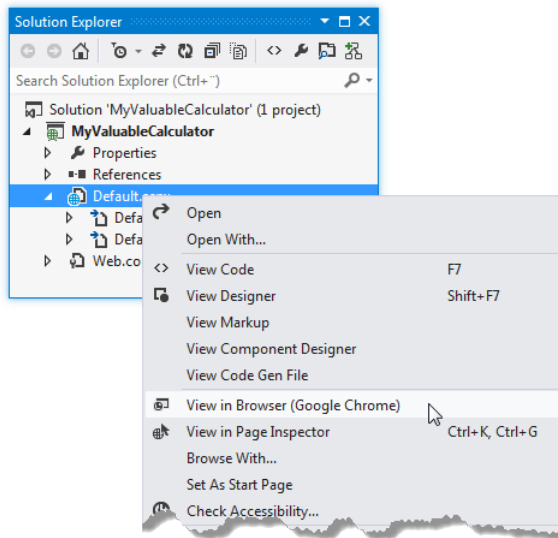
Då du kör en applikation startar automatiskt IIS Express och därefter öppnas sidan i den webbläsare som är vald som standard.

Andra sätt att starta en webbapplikation är att välja **Debug ► Start Without Debugging**.



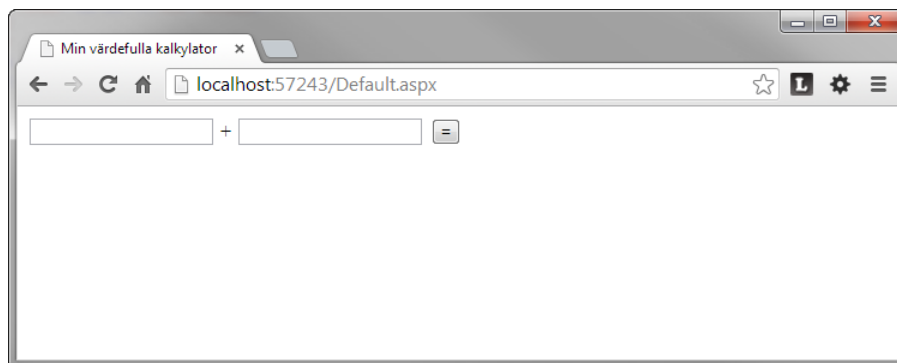
Figur 13.

Du kan också högerklicka på sidan du vill köra i fönstret **Solution Explorer** och välja **View in Browser**.



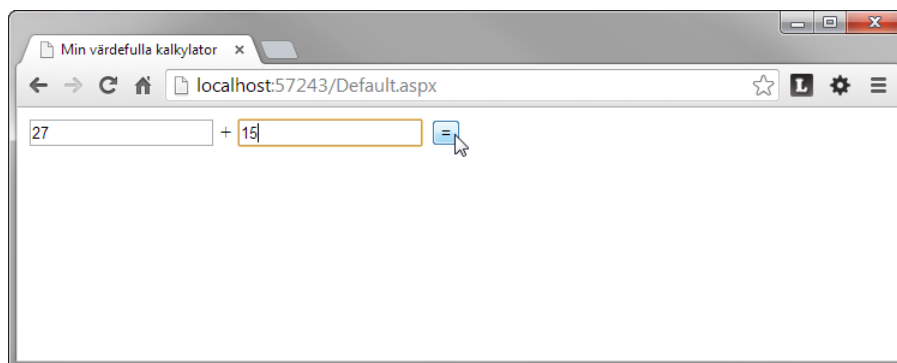
Figur 14.

Vill du ändra standardwebbläsare väljer du **Browse With....** Du har då en möjlighet att bestämma vilken webbläsare du vill använda och dessutom storleken som webbläsarfönstret ska ha.



Figur 15.

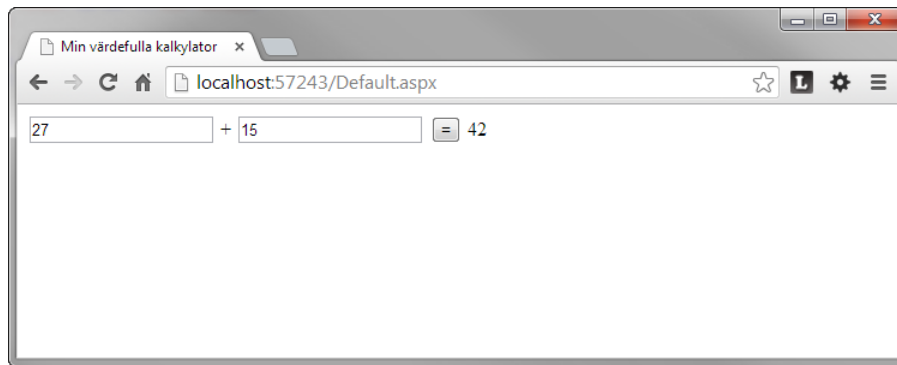
12. Nu när applikationen är färdig(?) är det hög tid att testa den. Testa med värden du vet summan av.



Figur 16.

... och klicka på kommandoknappen.

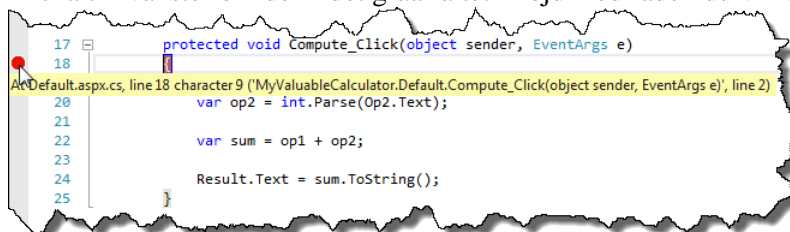




Figur 17.

Allt verka fungerar som det ska. Men det är nog bäst att du kontrollerar med några fler värden...


13. Vad är det som händer då du klickar på kommandoknappen? Ett enkelt sätt för dig att ta reda på det är att använda dig av ”debuggern”. Sätt en brytpunkt i början av hanterarmetoden genom att klicka till vänster om den i det gråa fältet i höjd med raden du vill att exekveringen ska stanna på.



Figur 18.

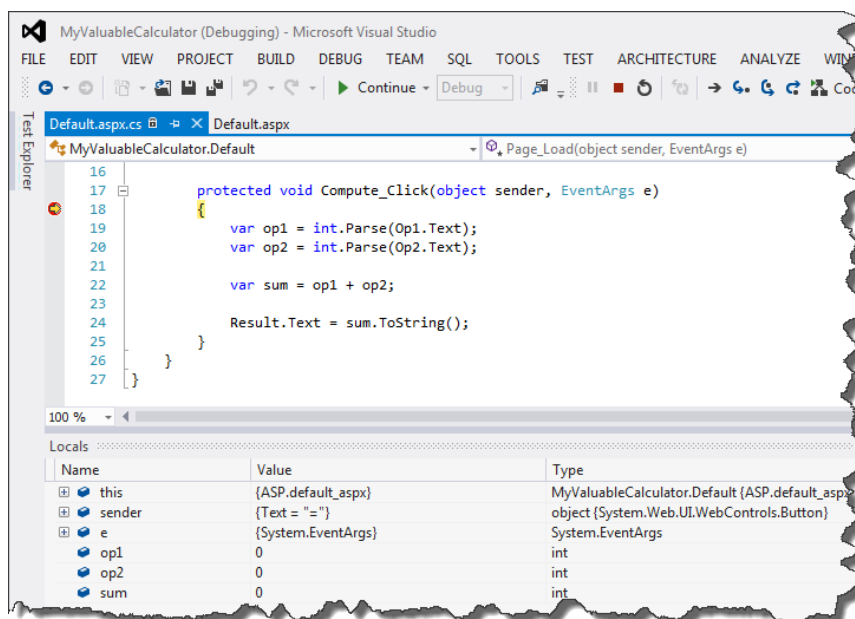
Andra sätt att skapa en brytpunkt är att trycka på F9 eller välja **Debug ► Toggle Breakpoint**.

Du tar bort en brytpunkt genom att klicka på brytpunkten, trycka F9 eller välja **Debug ► Toggle Breakpoint**. Du kan också välja **Debug ► Delete All Breakpoints**, som tar bort alla brytpunkter.

För att köra applikationen i ”debug”-läge klickar du på knappen  i verktygsfältet, trycker på F5 eller väljer **Debug ► Start Debugging**.

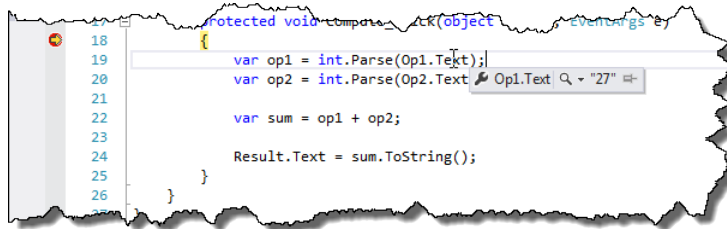
14. Fyll i några värden i textrutorna och klicka på kommandoknappen.

Exekvering stannar vid raden som är markerad med en brytpunkt. Du kan nu exempelvis inspektera variablers värden i fönstret **Locals**.



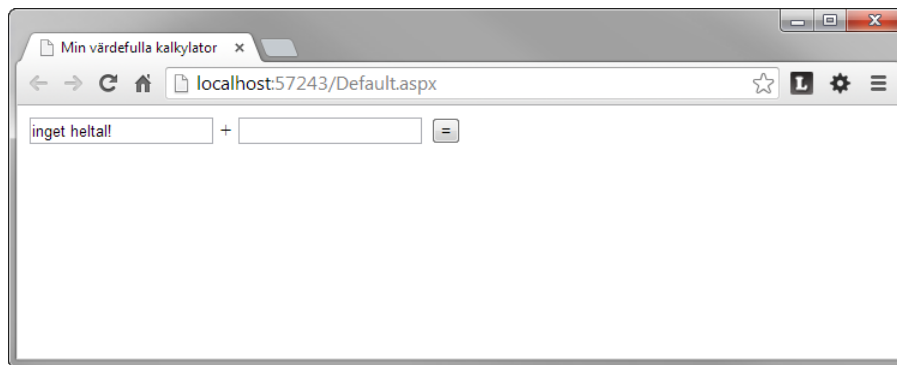
Figur 19.

Du kan även direkt i koden genom att hålla muspekaren över variabler eller egenskaper, som t.ex. Text, se vilka värden de har.



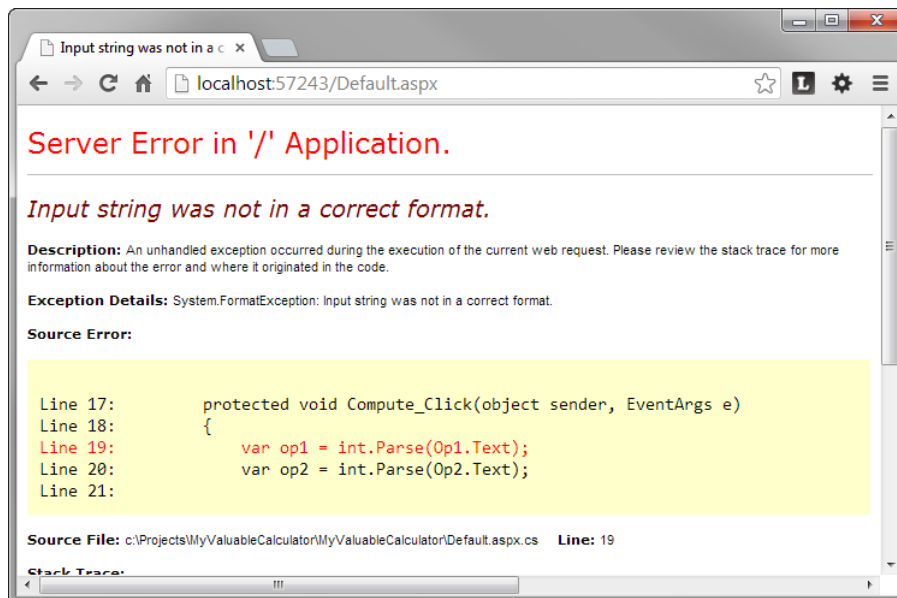
Figur 20.

15. Vad händer om du anger något som inte kan tolkas som ett heltal av typen int?



Figur 21.

Ett undantag kastas av `int.Parse(Op1Textbox.Parse)` som ASP.NET fångar och sidan nedan visas.

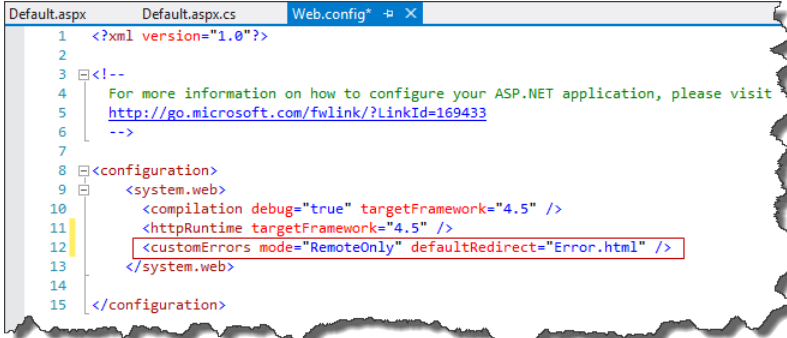


Figur 22.

Ingen trevlig sida<sup>1</sup> att visa för en användare av applikationen. Självklart kan mycket lösas genom att validera datat innan det skickas till webbservern. Men fortfarande ska inte ovanstående sida visas om det kastas ett undantag.

I konfigurationsfilen `web.config` kan du deklarera vilka sidor som ska visas då olika typer av fel inträffar. Elementet `customError` saknas vilket är det första du måste åtgärda.

<sup>1</sup> Benämns "gulfylt sida" under kursen.



```

1 <?xml version="1.0"?>
2
3 <!--
4 For more information on how to configure your ASP.NET application, please visit
5 http://go.microsoft.com/fwlink/?LinkId=169433
6 -->
7
8 <configuration>
9   <system.web>
10     <compilation debug="true" targetFramework="4.5" />
11     <httpRuntime targetFramework="4.5" />
12     <customErrors mode="RemoteOnly" defaultRedirect="Error.html" />
13   </system.web>
14 </configuration>
15

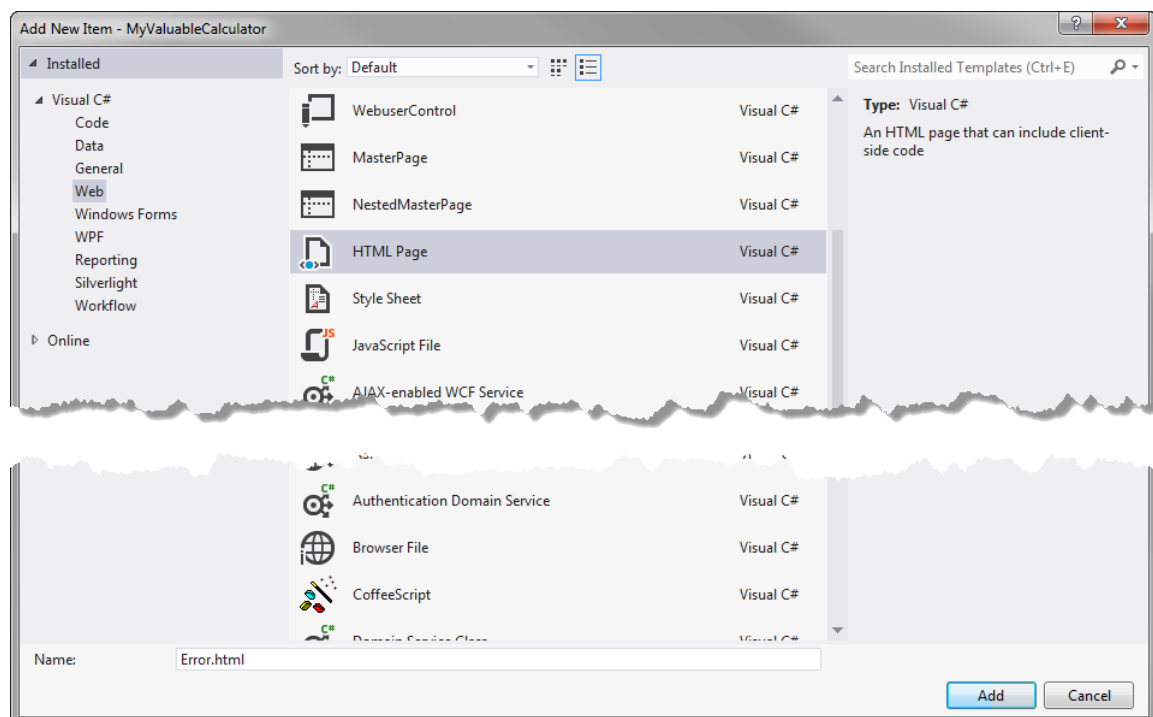
```

Figur 23.

Du kan ta hand olika typer av fel som inträffar. I detta fall kan du dock nöja dig med att ha en enda felsida som visas för alla fel.

Felsidan kan ha vilket namn som helst, och vara placerad i vilken katalog som helst. I detta fall måste du skapa en HTML-sida med namnet `Error.html` i roten på webbapplikationen.

Högerklicka på applikationsroten och välj **Add New Item...** Dialogrutan **Add New Item** visas. Välj **HTML Page** och ge sidan samma namn som det står i konfigurationsfilen `Web.config` och klicka på **Add**.



Figur 24.

16. Redigera den nya sidan så att den innehåller lämpligt meddelande.



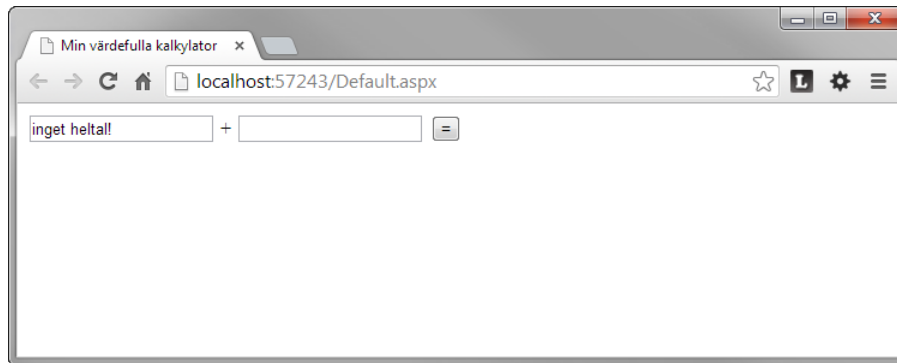
```

1 <!DOCTYPE html>
2 <html xmlns="http://www.w3.org/1999/xhtml">
3 <head>
4   <title>Min värdefulla kalkylator</title>
5 </head>
6 <body>
7   <h1>
8     Fel
9   </h1>
10  <p>
11    Ett fel inträffade då förfrågan behandlades.
12  </p>
13 </body>
14 </html>

```

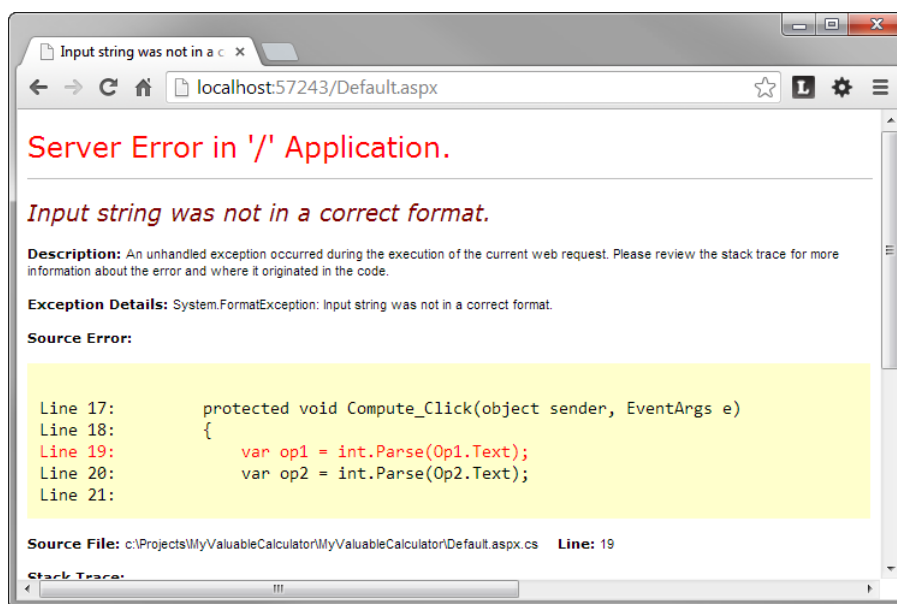
Figur 25.

17. Prova på nytt att ange något som inte kan tolkas som ett heltal av typen int.



Figur 26.

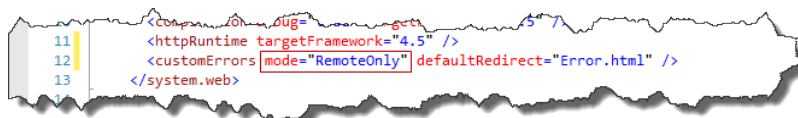
Konstatera att...



Figur 27.

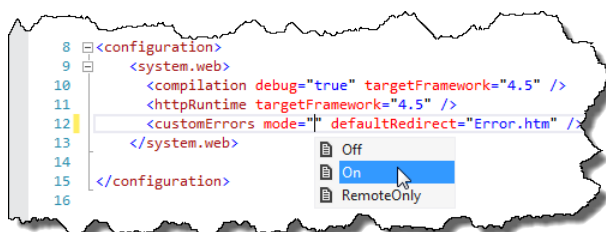
..det inte förändrade någonting.

Problemet är att i konfigurationsfilen Web.config har attributet mode värdet RemoteOnly, vilket får till följd att du som utvecklare inte får se felsidan utan istället visas den av ASP.NET automatgenererade sidan.



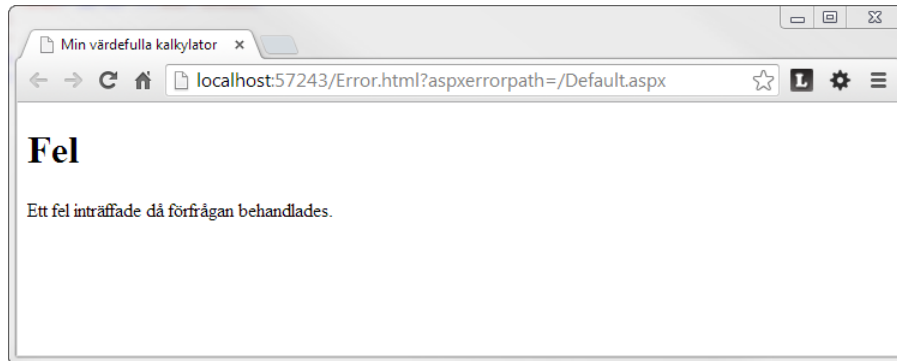
Figur 28.

Genom att sätta mode till On visas din felsida. (Om du raderar RemoteOnly, har insättningspunkten mellan citattecknen och trycker Ctrl + Mellanslag visas fönstret med giltiga värden för attributet mode.)



Figur 29.

Provat du på nytt med ett värde som inte kan tolkas som ett heltal av typen `int` visas nu din felsida.



Figur 30.

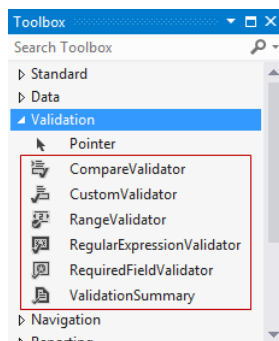
Det är ett allmänt felmeddelande som inte ger användaren någon återkoppling vad som gick fel. Genom att använda validering av datat innan det skickas till webbservern kan du ge användaren bättre felmeddelanden.

18. Det finns i princip bara en typ av fel som kan inträffa i detta fall och det är att textfälten innehåller något som inte kan tolkas som ett heltal av typen `int`. Ur ASP.NET:s synpunkt är det två typer av fel som kan inträffa.

- Användaren matar inte in något värde i ett eller båda textfälten.
- Användaren matar inte in ett eller två värden som inte kan tolkas som heltal.

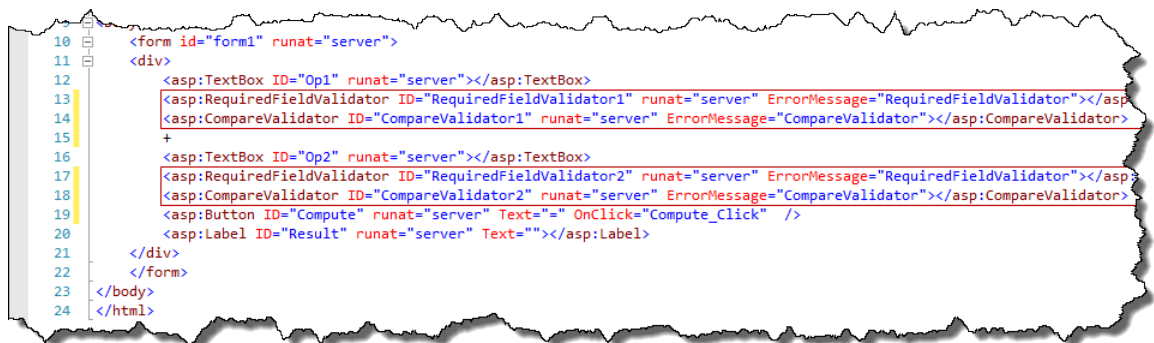
Du kan genom att använda dig av valideringskontrollerna **RequiredFieldValidator** och **CompareValidator** på ett enkelt sätt validera datat både på klienten och på webbservern.

Du hittar de valideringskontrollerna under fliken **Validation** i fönstret **Toolbox**.



Figur 31

Dra ut och släpp kontrollerna i anslutning till respektive **TextBox**-kontroll. En valideringskontroll kan bara validera en annan kontroll, så i detta fall krävs det två **RequiredFieldValidator**-kontroller och två **CompareValidator**-kontroller.

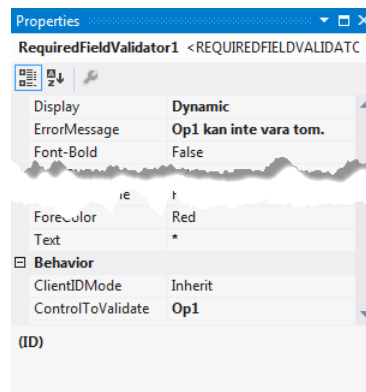


Figur 32.

Hur **RequiredFieldValidator**- och **CompareValidator**-kontrollerna ska fungera bestämmer du med hjälp av deras egenskaper.

Egenskaper du ska sätta för **RequiredFieldValidator**-kontrollerna är:

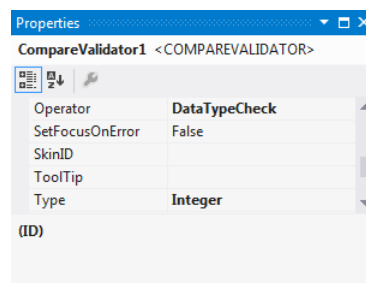
- **Display** sätts till **Dynamic**
- **ErrorMessage** sätts till **Op1 kan inte vara tom..**
- **Text** sätts till **\***.
- **ControlToValidate** sätts till **Op1** respektive **Op2**.



Figur 33.

Egenskaper du ska sätta för **CompareValidator**-kontrollerna är:

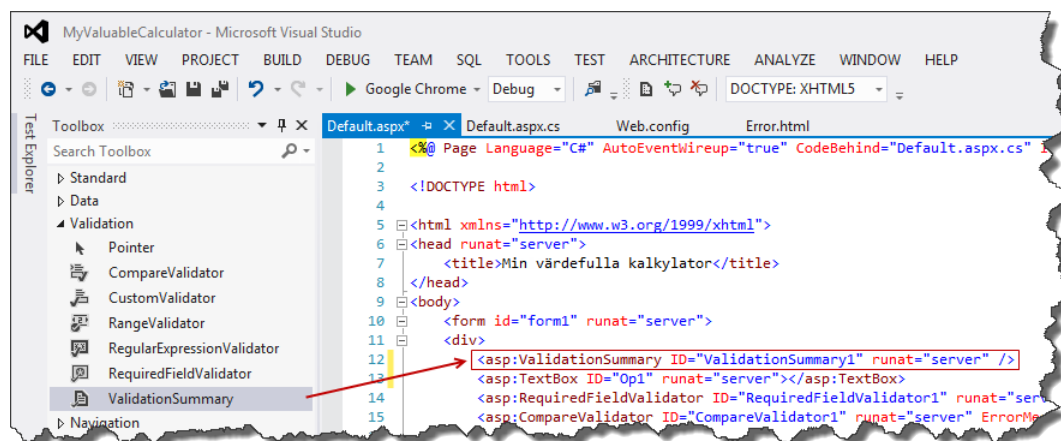
- **Display** sätts till **Dynamic**
- **ErrorMessage** sätts till **Op1 måste innehålla ett heltal..**
- **Text** sätts till **\***.
- **ControlToValidate** sätts till **Op1** respektive **Op2**.
- **Operator** sätts till **DataTypeCheck**.
- **Type** sätts till **Integer**.



Figur 34.

Valideringskontroller använder sig av automatgenererade JavaScript på klienten för att validera datat. Du behöver med andra ord inte skriva en enda rad JavaScript för att validera på klienten.

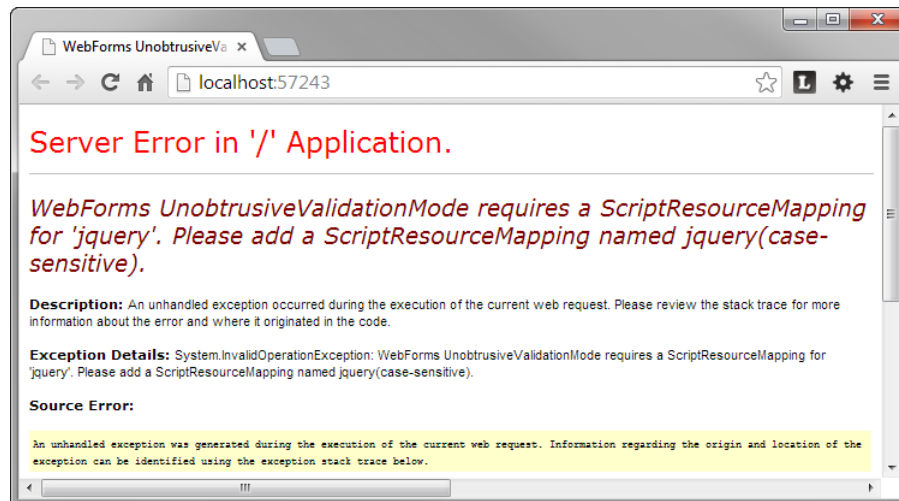
Felmeddelanden kan presenteras på klienten på olika sätt. Vill du visa en lista med felmeddelanden använder du dig av kontrollen **ValidationSummary**.



Figur 35.

Du behöver inte ändra på några egenskaper för att en lista med fel ska visas. Innehållet i listan bestäms av vad egenskapen **ErrorMessage** har för värde i de tidigare valideringskontrollerna.

Provkör du nu får du ett felmeddelande beroende på att från och med version 4.5 används "unobtrusive JavaScript" vid validering på klienten.

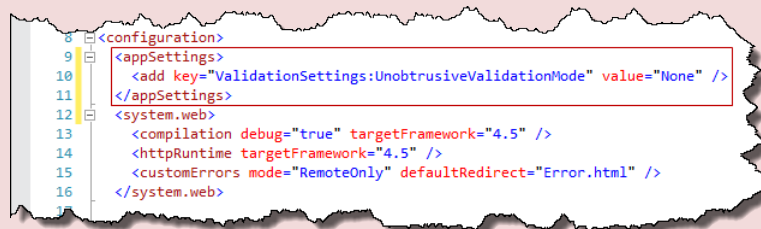


Figur 36.

Det finns två (nåja) lösningar på problemet:

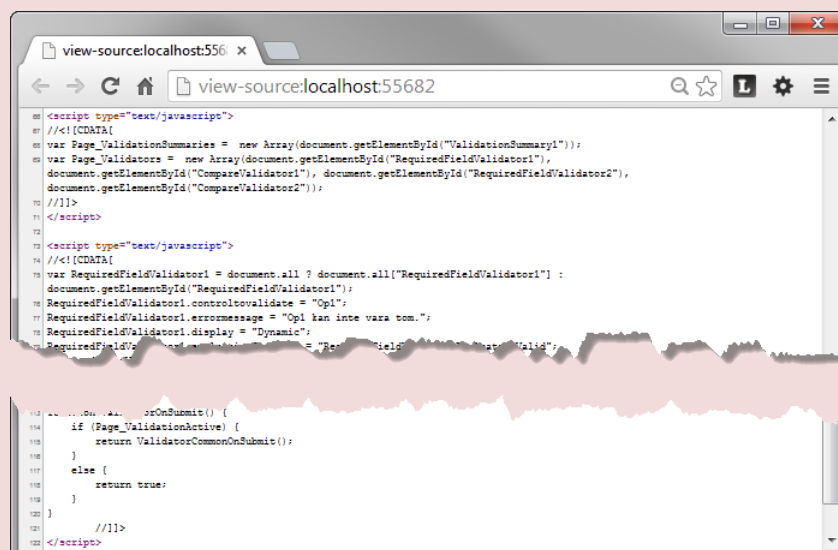
- Snabbast och sämst är ”old school”-lösningen som innebär att applikationen inte använder sig av ”unobtrusive validation” (punkt 19).
- Bästa lösningen för att få valideringen på klienten att fungera är att se till att jQuery-filer läggs till webbapplikationen. Dessa måste även definieras och registreras med namnet ”jquery” (punkt 20).

19. ”Old School”-lösningen innebär att du stänger av ”unobtrusive validation” genom att lägga till attribut i Web.config.



Figur 37.

Lösningen innebär att valideringen på klienten sker på det gamla sättet vilket medför att en icke oansenlig mängd JavaScript bäddas in i sidan.



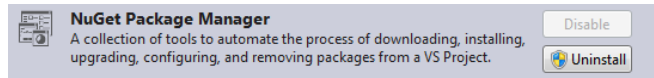
Figur 38.

En bättre lösning är att istället se till att ”unobtrusive validation” fungerar på klienten enligt punkt



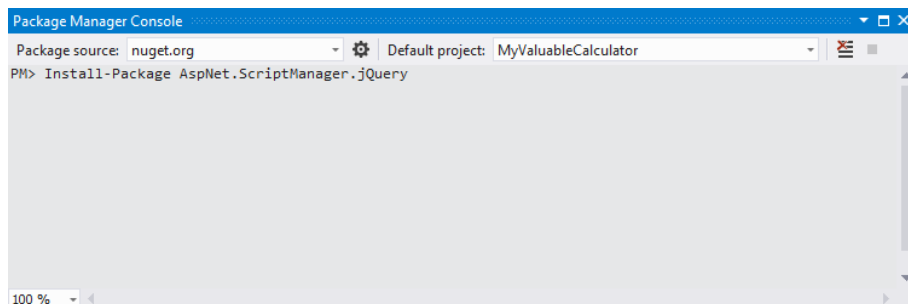
20.

20. Du måste ha *NuGet Package Manager* installerad, och helst minst version 2.7.41101.299, för att kunna följa anvisningen som följer i denna punkt. Du lägger till den med **Tools ► Extensions and Updates....**



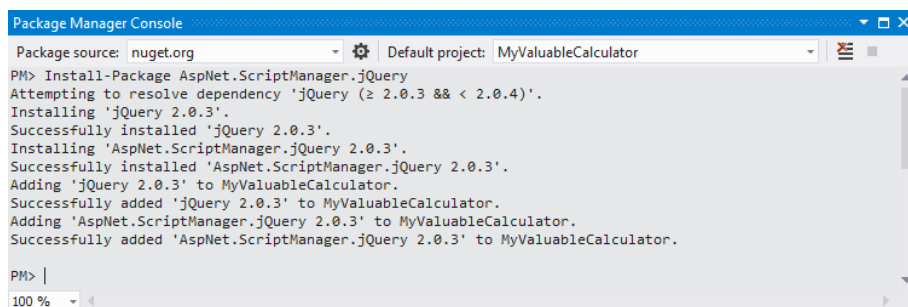
Figur 39.

För att lägga till jQuery-filer välj **Tools ► Library Package Manager ► Package Manager Console** och skriv in `Install-Package ASPNet.ScriptManager.jQuery` vid prompten.

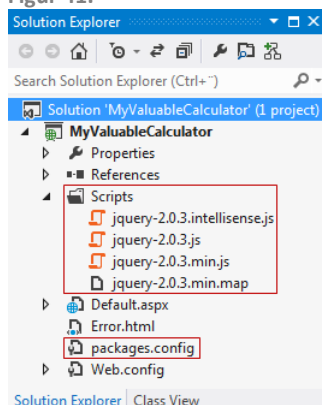


Figur 40.

Nödvändiga filer hämtas hem och läggs till webbapplikationen.



Figur 41.



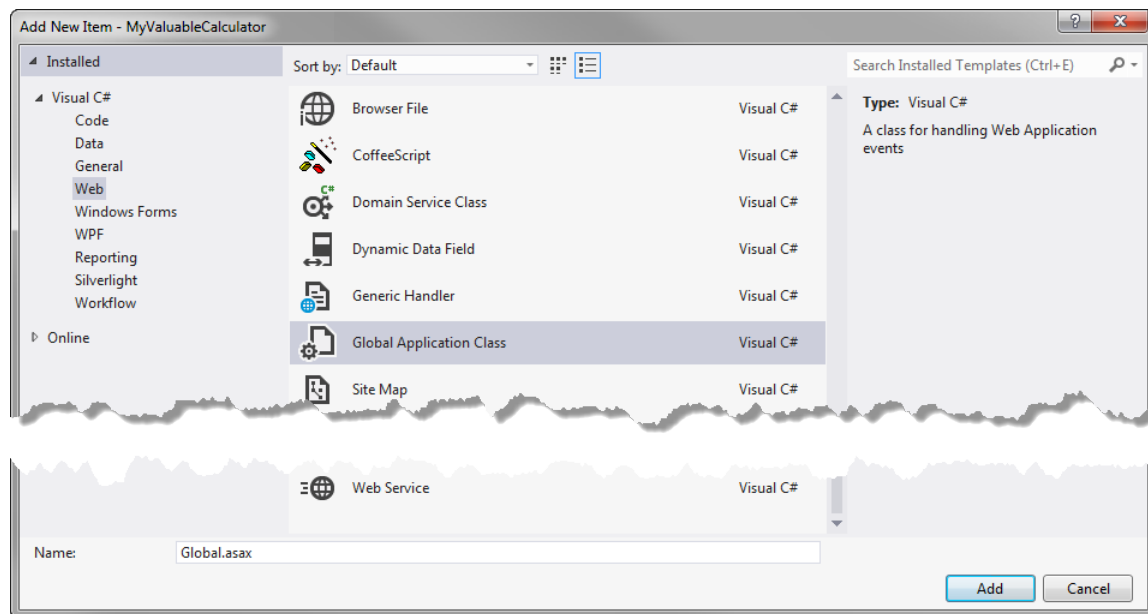
Figur 42.

Nödvändiga filer för validering på klienten placeras i katalogen Scripts. Filen `packages.config` används av *Package Manager* för att hantera paket som lagts till webbapplikationen.

21. För att registrera och definiera jQuery-filerna måste du lägga till filen `Global.asax`.

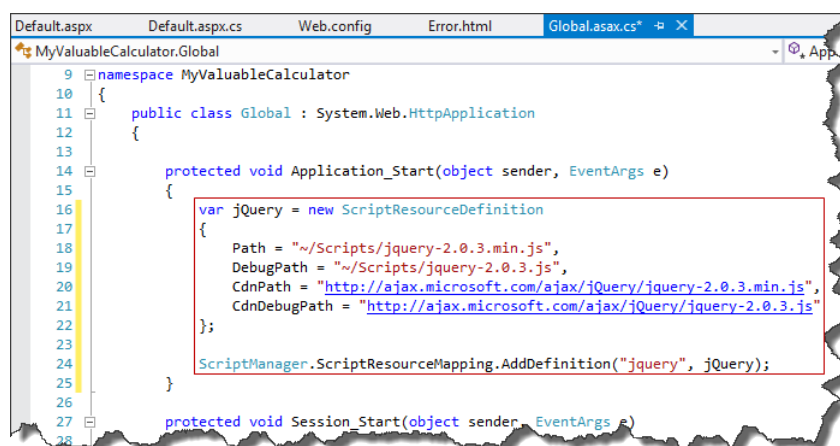
Högerklicka på applikationsroten och välj **Add New Item....** Dialogrutan **Add New Item** visas. Välj **Global Application Class** och klicka på **Add**.





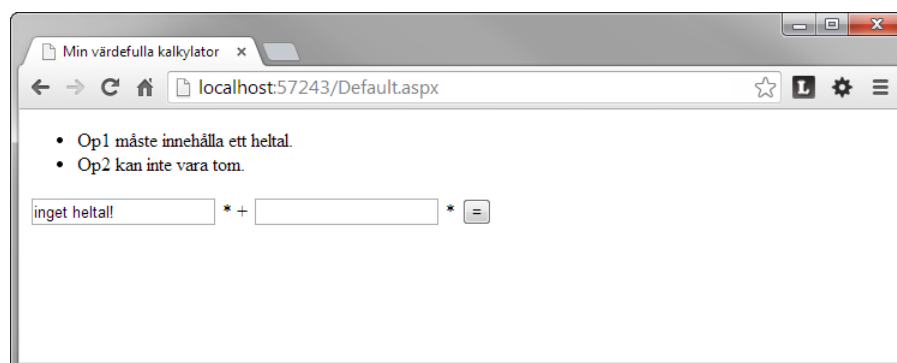
Figur 43.

I hanterarmetoden `Application_Start` registrerar du jQuery.



Figur 44.

Provkör du nu visas en lista med fel utan att något postas till server.



Figur 45.

22. Du måste även se till att webbapplikationen verkligen arbetar med data som är validerat även på servern. Du behöver inte skriva något på servern heller, så när på att du måste undersöka om datat klarat valideringen eller inte.

Har egenskapen `IsValid` värdet `true` har formulärdatat klarat valideringen.

```

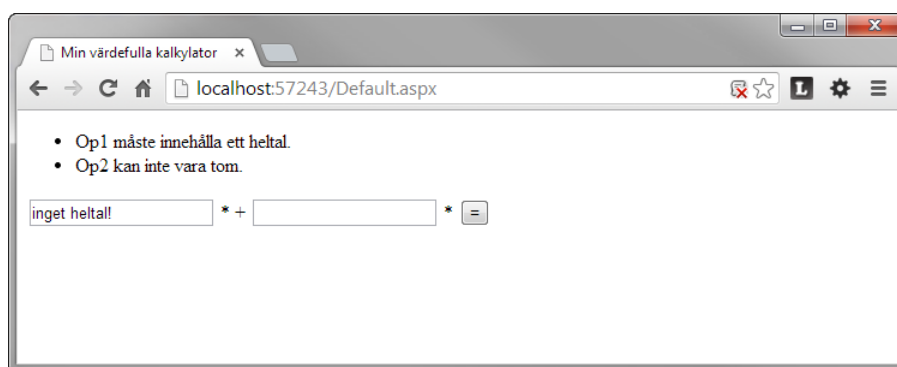
17 protected void Compute_Click(object sender, EventArgs e)
18 {
19     if (IsValid)
20     {
21         var op1 = int.Parse(Op1.Text);
22         var op2 = int.Parse(Op2.Text);
23
24         var sum = op1 + op2;
25
26         Result.Text = sum.ToString();
27     }
28 }

```

Figur 46.

Hanterarmetoden `Compute_Click` körs oavsett om datat klarat valideringen eller inte. Därför måste du alltid undersöka om egenskapen `IsValid` är `true` innan du på något sätt hanterar datat.

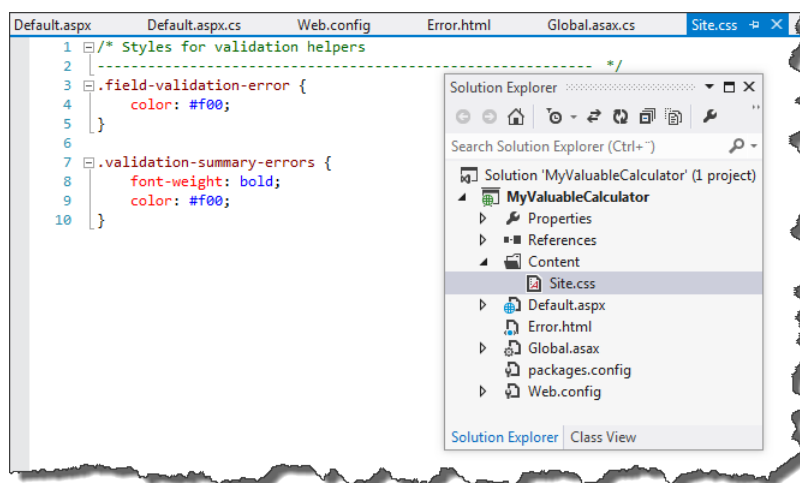
Även om klienten inte tillåter JavaScript kommer valideringen att utföras på servern och listan med fel visas på klienten.



Figur 47.

23. Felmeddelanden framträder inte så tydligt i figur 47 vilket kan åtgärdas genom att knyta CSS-regler, självklart definierade i en extern stilmall, till valideringskontrollerna.

- Skapa en ny katalog, `Content`, under projektroten. I denna katalog placerar du filer som t.ex. externa stilmallar.
- Lägg till en stilmall med namnet `Site.css` i katalogen `Content`. Se till att stilmallen innehåller reglerna enligt figur 48.



Figur 48.

- Webbformuläret `Default.aspx` måste kompletteras med en hänvisning till den externa stilmallen. Lägg märke till den lite speciella URL:en på rad 8 i figur 49. Värdet attributet `href` tilldelas inleds med tecknet `~`, som symboliserar applikationsroten vilket gör det möjligt att använda sökvägar relativt applikationsroten på ett enkelt sätt.

```

1 <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="..." %>
2
3 <!DOCTYPE html>
4
5 <html xmlns="http://www.w3.org/1999/xhtml">
6 <head runat="server">
7 <title>Min värdefulla kalkylator</title>
8 <link href="~/Content/Site.css" rel="stylesheet" type="text/css" />
9 </head>

```

Figur 49.

- d) Genom egenskapen **CssClass** kan du tilldela en kontroll en css-regel. Du kan skriva in css-regeln direkt i webbformuläret...

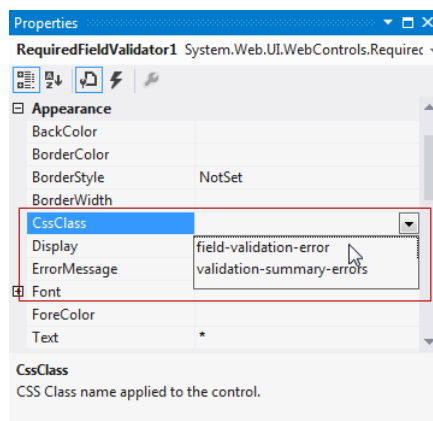
```

<asp:ValidationSummary ID="ValidationSummary1" runat="server" CssClass="validation-summary-errors" />
<asp:TextBox ID="Op1" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server" ErrorMessage="Op1 måste innehålla ett heltal." />
+
<asp:CompareValidator ID="CompareValidator1" runat="server" ErrorMessage="Op2 kan inte vara tom." />
<asp:TextBox ID="Op2" runat="server"></asp:TextBox>

```

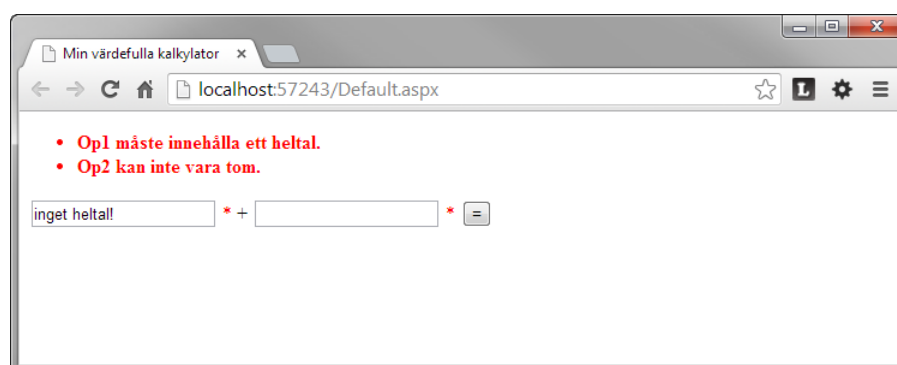
Figur 50.

...eller använda fönstret **Properties**.



Figur 51.

Nu framträder felmeddelanden tydligare.



Figur 52.

24. Provkör och se till att allt fungerar på det sätt du vill. Applikationen är långt från fulländad men förhoppningsvis har du nu fått en liten introduktion till vad ASP.NET Web Forms handlar om. I kurslitteraturen hittar du mycket mer information om de olika saker som tagits upp under denna kortfattade introduktion. Under kommande föreläsningar kommer du självklart också få mer fördjupad information.
25. Har du tid över? Varför då inte komplettera webbapplikationen så användaren kan välja vilket räknesätt som ska användas från en **DropDownList**-kontroll.