Teun Duynstee
Lead developer bij Macaw
www.macaw.nl

ASP.NET en controls

Wat zijn controls?

Iedereen die ervaring heeft met het programmeren van client-applicaties in bijvoorbeeld Visual Basic of Delphi is bekend met het concept: 'controls'. Controls zijn softwarecomponenten die vaak een eigen stukje van de user interface van een applicatie voor hun rekening nemen. Deze softwarecomponenten kunnen in gecompileerde vorm (dus zonder dat je de broncode hoeft te hebben) worden hergebruikt. Controls hebben daarom allerlei instellingen om ze bruikbaar te maken in vele toepassingen. Vaak heeft een control ook een design-time user interface die de ontwikkelaar ondersteunt bij het kiezen van de juiste instellingen.

Een eenvoudig voorbeeld van een control in een client-applicatie is de 'button'. Voor de verwerking van kliks op een bepaald stukje van het scherm en het tekenen van de schaduwen op het scherm (en het veranderen van de schaduwlijnen als de gebruiker op de button klikt) is allerlei logica nodig. Toch hoeft een Visual Basic-programmeur daar nooit over na te denken. Alle functies, inclusief de user interface zitten geëncapsuleerd in de control. Alleen de property's van de control – en de methoden die erop kunnen worden aangeroepen – zijn van belang. De productiviteitswinst van het werken met herbruikbare controls is enorm.

In een webomgeving…

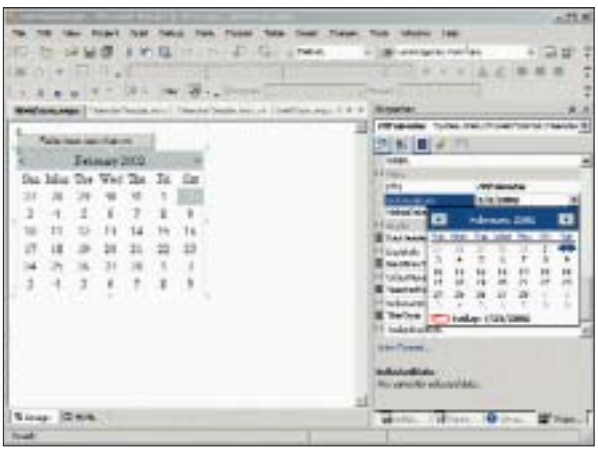
Sinds het web als een serieus platform voor applicatieontwikkeling wordt gezien, proberen de verkopers van ontwikkeltools het paradigma van controls naar het web te vertalen. Tot nu toe met matig succes. Het feit dat een control zowel de logica als de user interface van een stukje van het scherm definieert, bleek lastig over te zetten naar de webomgeving. Daar is namelijk de user interface opgebouwd in HTML, terwijl de logica van de pagina vaak op de server draait, omdat je vanaf de client-machine nu eenmaal geen toegang hebt tot informatie in databases. Verschillende

```
<%@ Page language="c#" Codebehind="WebForm.aspx.cs" AutoEventWireup=
"false" Inherits="NetMagSample.WebForm1" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
  <HEAD>
    <title>.NET Magazine Voorbeeld</title>
    <meta content="Microsoft Visual Studio 7.0" name="GENERATOR">
    <meta content="C#" name="CODE_LANGUAGE">
    <meta content="JavaScript" name="vs_defaultClientScript">
    <meta content="http://schemas.microsoft.com/intellisense/ie5"
    name="vs_targetSchema">
  </HEAD>
  <body>
    <form id="Form1" method="post" runat="server">
      <asp:button id="ctlButton" runat="server" Text=
          "Selecteer een datum"></asp:button>
      <br>
      <asp:calendar id="ctlCalendar" runat="server" SelectedDate=
          "2002-02-02" VisibleDate="2002-02-02"></asp:calendar>
    </form>
  </body>
</HTML>
```

Afbeelding 1.

Afbeelding 2.

leveranciers hebben hun tanden stukgebeten op het integreren van het web met de gedachte aan controls. De ontwikkeling van ActiveX-controls, Java-applets en DHTML maakte het gemakkelijker om met controls op de client te werken. Het nadeel hiervan bleek echter te zijn dat de werking van de applicatie niet erg 'website-achtig' meer was en zeer afhankelijk werd van de gebruikte client-software (de browser). Bovendien is het niet aantrekkelijk om de logica van een businessapplicatie in de browser te laten uitvoeren. In ASP.NET wordt de control-infrastructuur geheel naar de serverkant gehaald. De controls 'leven' op de server en hebben interactie met elkaar op de server. De webpagina is een zuivere HTML-neerslag van deze server-controls. Elke control is verantwoordelijk voor zijn eigen stukje HTML-code. Uiteraard komt daar heel wat bij kijken, want de controls op de server moeten natuurlijk wel kunnen reageren op de acties van de gebruiker. Maar deze acties moeten dan op de één of andere manier worden gecommuniceerd naar de controls op de server. Verder in dit artikel zullen we een tipje oplichten van hoe de architectuur van ASP.NET dit mogelijk maakt. Maar eerst zullen we bekijken hoe je als ontwikkelaar van een website met deze technologie omgaat.



Afbeelding 3.

Een eenvoudig voorbeeld

Een webpagina wordt in ASP.NET een webform genoemd. Deze webforms bestaan normaal gesproken uit een ASPX-document en een codebestand dat de achterliggende code van het webform bevat. Het ASPX-document bevat de opmaak van de webpagina en lijkt sprekend op een klassiek HTML-bestand. Pas op: het lijkt wel op een HTML-bestand, maar er gebeurt een heleboel met dit bestandje voordat er een daadwerkelijke HTML-pagina naar de browser wordt gestuurd! In afbeelding 1 zie je de code van een eenvoudige ASPX. Onmiddellijk vallen een paar verschillen op met een klassieke HTML-pagina. De eerste regel bevat allerlei registratie-informatie en instellingen van de pagina. Verder zien we in de pagina twee tags gebruikt worden die we in HTML niet kennen: `asp:button` en `asp:calendar`. Beide hebben een attribuut `runat=server`, het teken voor ASP.NET dat het hier controls betreft. De HTML-pagina die een gebruiker in zijn browser te zien krijgt zie je in afbeelding 2.

Het resultaat van een ASP.NET-control kan dus nogal uiteenlopen. In het eerste geval heeft de control gewoon een bijbehorende HTML-tag aangemaakt, terwijl de andere control een grote hoeveelheid HTML-code heeft gegenereerd. In de Design View van Visual Studio kan de ontwikkelaar de controls als een vast blok selecteren en verslepen over de pagina. Net als in Visual Basic client-applicaties bepaalt de control zelf hoe het er uit ziet in de Design Mode en kan dus ook extra hulpschermen tonen voor het configureren van de control. Vooral voor controls met veel complexe instellingen is dat een belangrijke eigenschap (zie afbeelding 3).

```
/// <summary>
/// Als op de button geklikt wordt, tonen we de kalender als hij
/// onzichtbaar was en omgekeerd.
/// </summary>
private void ctlButton_Click(object sender, System.EventArgs e)
{
  ctlCalendar.Visible = !ctlCalendar.Visible;
}
```

Afbeelding 4.

Dan nu het spannende stuk: wat moet je als ontwikkelaar doen om te reageren op acties van de gebruiker? We maken de kalender onzichtbaar (door in de ASPX de property `Visible="False"` op de control te zetten) voor de gebruiker. Als deze dan vervolgens op de button klikt, moet de kalender worden getoond. De control die verantwoordelijk is voor het tonen van de kalender moet op de hoogte worden gebracht dat er iets moet veranderen. Net als in client-toepassingen doen we dit door het afvangen van events op de controls. Let wel: dit zijn dus events die plaatsvinden op de server. In de browser bestaan deze controls immers niet, daar is alleen HTML-code. ASP.NET zorgt ervoor dat de acties van de gebruikers worden omgezet in een event op de server, zodat we eenvoudig in de code van het webform kunnen beschrijven wat er na een klik op de button moet gebeuren (zie afbeelding 4).

Visual Studio.NET helpt je bij het koppelen van de events. Selecteer in Design Mode de control waarvan je een event wilt afvangen, in het Properties-window kan je nu met de Events-button (het

kleine bliksemschichtje) de events tonen. Dubbelklikken op een event maakt de event handling code voor je aan (zie afbeelding 5). De code die je hier ziet is geschreven in C#, maar dit had natuurlijk ook in VB.NET of een andere .NET-taal kunnen gebeuren. Omdat deze code op de server draait, stelt deze verder geen eisen aan de browser. Als de gebruiker een datum selecteert, willen we dat de datum getoond wordt op de button en dat de kalender weer verborgen wordt. Ook dit is eenvoudig te programmeren door een event handler te schrijven; ditmaal op het `SelectionChanged` event van de Calendar-control (zie afbeelding 6). Het veranderen van de Visible Property op de kalender en de Text Property op de button is natuurlijk vrij triviaal, maar met deze manier van werken is iedere soort van interactie met een webapplicatie vorm te geven op de manier waarop je dat ook in een VB-applicatie zou doen!

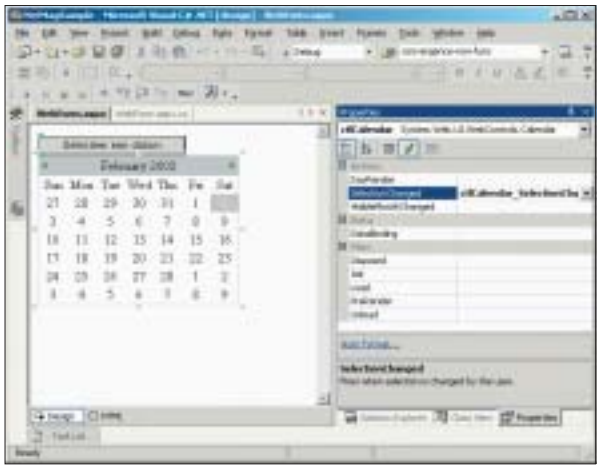
```
/// <summary>
/// Als de gebruiker met behulp van de kalender een nieuwe datum
/// selecteert, dan zetten we de waarde van deze datum op de
/// button en verbergen we de kalender
///
/// </summary>
private void ctlCalendar_SelectionChanged(object sender,
    System.EventArgs e)
{
    ctlButton.Text = ctlCalendar.SelectedDate.
       ToLongDateString();
  ctlCalendar.Visible = false;
}
```

Afbeelding 6.

Wat doen controls in ASP.NET

Hoe controls precies werken en wat er in een ASPX-pagina allemaal onder de motorkap gebeurt, voert een beetje te ver voor dit artikel (zie de links onderaan dit artikel voor sites met meer dan voldoende informatie). We zullen hieronder aan de hand van een aantal vragen proberen te schetsen wat controls voor je doen en hoe je ze dient te gebruiken.

Vraag: *Blijven de controls op de server voortdurend in het geheugen, terwijl ze wachten tot de gebruiker iets doet waardoor een event op de server wordt getriggerd?*



Afbeelding 5.

Nee, gelukkig niet, want dat zou een weinig schaalbaar systeem opleveren. Het is eerder zo dat op het moment dat een gebruiker in de browser iets doet waardoor een event getriggerd wordt, dat dan op de server snel opnieuw de controls voor deze pagina geïnstantieerd worden.

Vraag: *Ik zie in de documentatie nogal veel events. Zo zie ik op de control dat een textbox laat zien (asp:TextBox) een TextChanged event. Betekent dit dat bij ieder stukje tekst dat een gebruiker invoert, een event op de webserver afgaat en de hele pagina opnieuw moet worden opgehaald?*

Nee, het feit dat de tekst in de inputbox is veranderd, zal wel een event op de server veroorzaken, maar zal niet onmiddellijk een roundtrip naar de server tot gevolg hebben. Er zijn events die een zogenaamde "post-back" veroorzaken. Hiervan is het klik-event op een button een voorbeeld. De meeste events op controls worden echter opgespaard. Bijvoorbeeld: de veranderde tekst in een inputbox veroorzaakt wel een event, maar pas als een andere control heeft gezorgd voor een post-back. Op dat moment gaan alle opgespaarde events op de server af. Pas als allerlaatste wordt het event verwerkt dat voor de post-back zorgde. Op deze manier wordt gezorgd dat het aantal roundtrips naar de server beperkt is. Overigens kan je van sommige events aangeven of ze wel of niet een post-back moeten veroorzaken. Bijvoorbeeld, de controls asp: CheckBox en asp:ListBox kennen een AutoPostback property. Als deze op True wordt gezet, veroorzaakt iedere verandering in deze controls (bijvoorbeeld het aanvinken van een checkbox) een post-back, zodat je direct op de server kunt reageren op dit event.

Vraag: *Is het nog mogelijk om client-scripting te gebruiken?*

Ja. De server controls kunnen alle HTML genereren die je maar wilt. De ontwikkelaar van de control kan daarbij ook gebruik maken van client-scripting code. Een aantal van de controls van Microsoft doet dat ook, omdat het gebruik van client-scripts natuurlijk een grote performancewinst voor een website kan betekenen. Overigens is het schrijven van controls die client-scripting gebruiken wel lastiger dan het lijkt. Een control dient bijvoorbeeld niet zomaar een functie in een scriptblok te genereren. Stel dat de control tweemaal op dezelfde pagina voorkomt: dan zijn er twee functies met dezelfde naam (bij de links onderaan dit artikel vind je een zeer uitgebreid artikel over dit onderwerp).

Vraag: *Is een control eigenlijk niet een soort server-side include?*

Deze vraag verdient een subtieler antwoord dan ja of nee. In ASP.NET bestaat er geen server-side include meer. Als je iets dergelijks wilt, zou je dat in ASP.NET

```
<%@ Control Language="c#" AutoEventWireup="false" Codebehind="CalendarSample.ascx.cs"
Inherits="NetMagSample.CalendarSample" TargetSchema="http://schemas.microsoft.com/intellisense/ie5"%>
<asp:button id="ctlButton" runat="server" Text="Selecteer een datum"></asp:button>
<br>
<asp:calendar id="ctlCalendar" runat="server" SelectedDate="2002-02-02" VisibleDate="2002-02-02"
Visible="False"></asp:calendar>
```

Afbeelding 7.

oplossen door een control te definiëren. Toch is het niet hetzelfde. Een server-side include is een stukje code dat in een HTML-pagina geplakt wordt. De rest van de pagina wordt dan ongewijzigd naar de browser gestuurd. Bij ASP.NET-pagina's is het eigenlijk juist andersom: alle content in de pagina die geen control is, wordt vertaald naar een zeer lichtgewicht control dat als output altijd hetzelfde stukje HTML laat zien. Alle HTML in een pagina is dus het resultaat van een control! Zelfs de pagina zelf (de Page class) is een control.

Vraag: *Heb je met al die server-side events meer roundtrips naar de server?*
Ja. Als je op deze manier programmeert, zul je normaal gesproken vaker een nieuwe pagina ophalen van de server. Dat is vaak niet zo belangrijk, maar in bepaalde gevallen (erg grote pagina's, trage verbindingen) kan dat vervelend zijn. Het is dus van belang dat je je als ontwikkelaar bewust bent van wat er onder water bij het afhandelen van events gebeurt.

Zelf controls maken

Het is erg eenvoudig om in ASP.NET zelf controls te maken. Je doet precies hetzelfde als bij het schrijven van een pagina, maar in plaats van "Add Web Form" kies je "Add Web User Control". Net als bij ASPX-pagina's bestaat een user-control uit een pagina met HTML-opmaak (met de extensie ASCX in plaats van ASPX) en een achterliggend codebestand voor de programmacode (de "codebehind"). De class die je schrijft in de codebehind is een subclass van System.Web.UI.UserControl, die op zijn beurt een subclass is van System.Web.UI.Control. In afbeelding 7 zie je de code in de ASCX die we hebben gecreëerd op basis van de ASPX-pagina uit afbeelding 1. Zoals je ziet is met name de eerste regel verschillend. Deze geeft aan dat het hier niet om een pagina gaat, maar om een control. Als je deze control nu in een pagina wilt gebruiken, sleep je hem gewoon vanuit de Project Explorer op de pagina (in Design View). In afbeelding 8 zie je een voorbeeld van een pagina met meerdere instanties van de control. De bijbehorende code staat in afbeelding 9. Let op de tweede regel: deze is automatisch aangemaakt toen de control op het webform werd gesleept. In deze regel wordt de ASCX-file geregistreerd als control voor gebruik op de pagina. Dat is het! Met een paar regels code hebben we een nieuwe soort kalender-control gemaakt, die minder ruimte inneemt op de pagina, totdat je hem openklikt. Gemakkelijk te herbruiken, omdat het zowel de logica als een deel van de user interface encapsuleert.

Meerwaarde van controls

Alle onderdelen van de pagina, inclusief de pagina zelf, zijn dus Control-objecten. Het is interessant om te zien welke functionaliteit deze Control-class allemaal herbergt en waarop we verder kunnen bouwen.

- *Gezamenlijke context*: de Page creëert een aantal objecten dat door alle controls in de pagina gemakkelijk benaderd kan worden. Hierin zitten bijvoorbeeld het Session-object en het Application-object die voor ASP-programmeurs oude bekenden zijn, maar ook bijvoorbeeld het Trace-object, dat kan worden gebruikt om debugging-informatie weg te schrijven. Alle controls gebruiken deze gezamenlijke context en zorgen dat mogelijke "child controls" de context ook doorgegeven krijgen.
- *Naamgeving*: als meerdere controls van hetzelfde type op een pagina voorkomen, wordt het belangrijk om te zorgen dat de naamgeving van de elementen binnen de controls geen conflicten creëert. Als bijvoorbeeld een User Control een button bevat met de naam "button", zorgt ASP.NET er voor dat iedere instance van dit control in de uiteindelijke HTML-pagina unieke namen gebruikt voor de button.
- *ViewState*: aangezien een webform soms door acties van de gebruiker een aantal malen opnieuw moet worden opgebouwd, is er behoefte aan een manier om "state" vast te houden. Als we in reactie op een klik op een knop de titel van de pagina hebben veranderd, dan verwachten we dat die titel veranderd blijft zo lang we niets doen. In ASP.NET heeft men hiervoor een apart object in het leven geroepen, ViewState genaamd, dat net zo werkt als Session of Application, maar alleen binnen één Page werkt. Alle controls op een Page kunnen hierin informatie over hun state bewaren; in de Control-class zit code die zorgt dat het wordt bewaard tussen de post-backs en dat de verschillende controls niet per ongeluk elkaars informatie kunnen gaan overschrijven.
- *Caching*: ASP.NET bevat een aantal

Afbeelding 8.

```
<%@ Page language="c#" Codebehind="WebFormUsingCtrl.aspx.cs" AutoEventWireup="false"
Inherits="NetMagSample.WebFormsingCtrl" %>
<%@ Register TagPrefix="uc1" TagName="CalendarSample" Src="CalendarSample.ascx" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
  <HEAD>
    <title>WebFormsUingCtrl</title>
    <meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
    <meta name="CODE_LANGUAGE" Content="C#">
    <meta name="vs_defaultClientScript" content="JavaScript">
    <meta name="vs_targetSchema" content="http://schemas.microsoft.com/intellisense/ie5">
  </HEAD>
  <body>
    <form id="WebFormsingCtrl" method="post" runat="server">
      <table>
        <tr>
          <td valign=top>Startdatum:
          </td>
          <td valign=top>
            <uc1:CalendarSample id="CalendarSample1" runat="server"></uc1:CalendarSample>
          </td>
        </tr>
        <tr>
          <td valign=top>Einddatum:
          </td>
          <td valign=top>
            <uc1:CalendarSample id="CalendarSample2" runat="server"></uc1:CalendarSample>
          </td>
        </tr>
      </table>
    </form>
  </body>
</HTML>
```

Afbeelding 9.

zeer geavanceerde caching-technieken. Deze kunnen de performance van pagina's enorm verbeteren. Doordat ieder onderdeel van een pagina een control is, is het mogelijk om pagina's ook per onderdeel te cachen. Als je een pagina hebt waarvan een klein onderdeeltje telkens verandert, hoeft alleen dat stukje telkens opnieuw gegenereerd te worden.

Deze functionaliteit krijg je allemaal gratis zonder zelf code te schrijven. Om je zelfgemaakte control in een pagina op te nemen, hoef je ze er alleen maar op te slepen.

Even wennen

De ASP.NET -infrastructuur biedt webontwikkelaars de mogelijkheid om met webpagina's te werken op de manier die client-ontwikkelaars gewend zijn: controls met property's, methods en events. Hierbij gebeurt veel van de afhandeling op de server. Dat heeft vele voordelen, maar ook (performance) nadelen. Het is van belang om deze aspecten goed in je op te nemen. Ontwikkelaars met geruime ervaring in webdevelopment zullen in het begin moeten wennen aan het nieuwe programmeermodel. Veel van de zaken die een "klassieke" webdeveloper met client-side script zou oplossen zal je in ASP.NET oplossen via een extra round-trip. Vaak maakt dat qua performance niet zo heel veel uit. Voor de situaties waar dat echt onwenselijk is, dien je controls te schrijven die de correcte client-side code genereren.

Nuttige internetadressen

- http://www.asp.net
- http://msdn.microsoft.com/library/en-us/cpguide/html/cpcondevelopingwebformscontrols.asp (lijst van MSDN artikelen over het ontwikkelen van server controls)
- http://msdn.microsoft.com/library/en-us/cpguide/html/cpconclient-sidefunctionalityinservercontrol.asp (artikel over het ontwikkelen van clientside-functionaliteit in server controls)