Michiel Post

Snel en eenvoudig een data-editing UI

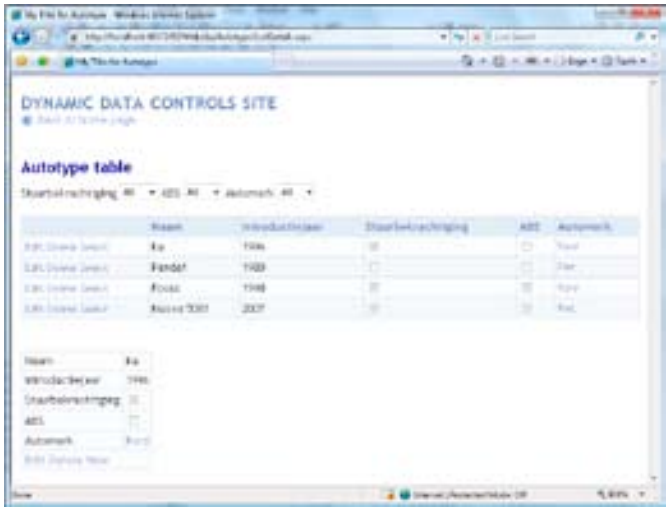
DYNAMIC DATA IN ASP.NET 3.5 EXTENSIONS CTP

Bij het ontwikkelen van webapplicaties gaat nog altijd veel tijd zitten in het maken van lijst- en detailpagina's. Zeker als je voor tientallen tabellen een lijst- en detailpagina moet maken, vraag je je af: kan dit niet eenvoudiger en sneller? In dit artikel beschrijft de auteur hoe dit moet.

Met de Dynamic Data Controls uit de ASP.NET 3.5 Extensions CTP (versie: december 2007) is het nu heel gemakkelijk om een eenvoudige data-editing-applicatie te maken voor de tabellen in jouw database. De Dynamic Data Controls halen het databaseschema runtime op en bepalen daaruit welke kolommen getoond worden en welke relaties er zijn. De controls presenteren de data vervolgens op de wijze die de gebruiker normaal zou verwachten. Bij foreign key-relaties wordt bijvoorbeeld automatisch een dropdownlist getoond. Je bespaart tijd, omdat je niet meer zelf hoeft te definiëren welke velden er op jouw pagina staan en hoe je ze uit de database ophaalt of wegschrijft. Natuurlijk kun je er ook voor kiezen om wel je eigen velden te definiëren en de presentatie ervan volledig aan te passen. Hoe dat in zijn werk gaat, laat ik in dit artikel zien. De Dynamic Data Controls zijn zeer geschikt voor het maken van een beheeromgeving of prototype. In dit artikel gebruik ik als voorbeeld een case waar we een beheeromgeving willen maken voor een website waarop auto's worden verhandeld. Op de beheeromgeving kunnen stamtabellen beheerd worden, in dit geval de tabellen 'Automerk' en 'Autotype'. Deze tabellen hebben een 1-n-relatie. Een uitwerking van deze applicatie is te downloaden; zie daarvoor de referenties aan het einde van dit artikel.

Dynamic Data

De Dynamic Data Controls werken op basis van *LINQ to SQL*. Nadat je een Dynamic Data Project hebt aangemaakt in Visual Studio hoef je alleen maar een LINQ to SQL-bestand aan te maken met daarin de tabellen die je wilt kunnen benaderen. In de web.config verwijs je vervolgens naar deze aangemaakte DataContext. Het resultaat is dat je direct een website hebt, waarop je alle gegevens kunt bekijken en wijzigen. De tabellen ondersteunen daarnaast standaard de mogelijkheid om te sorteren en te pagineren.



Afbeelding 1. Standaard Dynamic Data-pagina voor de Autotype-tabel

Dynamic Data-pagina's zijn verbonden met één tabel aan de hand van overeenkomende namen. Er wordt gekeken of er een tabel is in de Linq DataContext waarvan de naam overeenkomt met de subdirectory waar de pagina in staat. Het lijst-control dat een lijst laat zien uit de tabel 'Autotype' zal dus op een pagina in de subdirectory 'Autotype' staan. De naam van een pagina bepaalt binnen het Dynamic Data Framework de 'viewname'. Een viewname refereert naar een template-pagina. Standaard worden drie templates meegeleverd: een lijstpagina, een detailpagina en een pagina met zowel lijst- als detailgegevens. Welke view een bepaalde template aanspreekt, kun je configureren in de web.config. Verderop in dit artikel zal ik dieper ingaan op de templates. Zoals je in codevoorbeeld 1 ziet, verwijst de view 'List' naar het ListTemplate. Als je dus de url '~/Autotype/List.aspx' zou bezoeken, dan krijg je een pagina volgens de ListTemplate met de gegevens uit de tabel Autotype. Dit is geen fysiek bestaande pagina, maar omdat de opgevraagde url aan het gedefinieerde 'pattern' voldoet, zorgt het Dynamic Data Framework (DDF) ervoor dat toch een pagina getoond wordt. In codevoorbeeld 1 zie je ook de mogelijkheid een 'action' aan te geven. Er zijn drie ingebouwde acties te definiëren. Dit zijn list, details en de combinatie daarvan (list,detail). Het DDF gebruikt dit om in een lijstpagina te kunnen bepalen waar de detail-link naar moet verwijzen. Ook wordt het gebruikt om navigatie mogelijk te maken naar tabellen waar een relatie tussen ligt.

Let op: Het is belangrijk om te weten dat voor elke tabel slechts één list-actie en detail-actie kan worden geconfigureerd.

```
<mappings pattern="~/{table}/{viewName}.aspx">
  <add actions="list" viewName="List"
    templateFile="ListTemplate.aspx" />
  <add actions="details" viewName="Details"
    templateFile="DetailsTemplate.aspx" />
</mappings>
```

Codevoorbeeld 1.

```
<add actions="list" tables="Autotype" viewName="AangepasteLijst"/>
```

Codevoorbeeld 2.

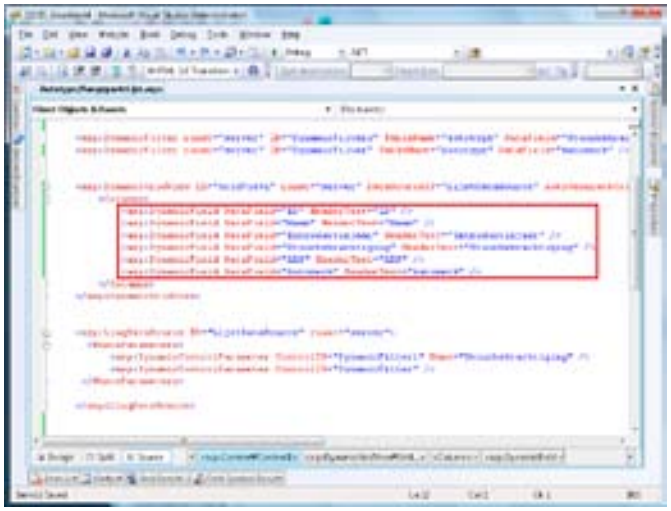
```
void GridView1_RowDataBound(object sender, GridViewRowEventArgs e)
{
  if (e.Row.RowType == DataControlRowType.DataRow)
    {
      Autotype type = (Autotype)e.Row.DataItem;

        if(type.Introductiejaar > 2000)
          e.Row.Cells[2].BackColor = Color.Yellow;
  }
}
```

Codevoorbeeld 3

Lijstpagina

Standaard krijgen we dus al een lijst- en detailpagina voor elke tabel. Voor de beheerpagina van de autoverhandelsite willen we een aangepaste lijst maken voor de tabel Autotype. Een aangepaste pagina die dus niet via een template wordt opgebouwd, maar waar we zelf alle elementen op plaatsen. Om dit te kunnen doen, maken we een subdirectory 'Autotype' met daarin een nieuwe pagina genaamd 'AangepasteLijst.aspx':
- ~/Autotype/ AangepasteLijst.aspx
Nu moeten we het DDF nog duidelijk maken dat we deze aangepaste pagina willen gebruiken in plaats van de standaardlijst. Dit kan door de configuratieregel op te nemen uit codevoorbeeld 2.

In dit codevoorbeeld geven we aan dat we voor de tabel 'Autotype' een view 'AangepasteLijst' hebben en dat dit de standaard lijstpagina is. De pagina's voor de overige tabellen blijven ongewijzigd, daarvoor wordt nog steeds het standaard template gebruikt. Aan de nieuwe lijstpagina voegen we een Dynamic GridView en LinqDataSource toe. Het resultaat is een kant-en-klare tabel die de gegevens uit de database toont. Wat meteen opvalt als we de pagina bekijken, is dat de kolom Automerk niet het ID toont. De daadwerkelijke naam van het merk wordt getoond. De kolomnaam heet zelfs Automerk en niet fkAutomerkID zoals hij eigenlijk in de database heet. Het Dynamic Data Framework ziet dat er een relatie ligt tussen fkAutomerkID in de Autotype-tabel en ID in de Automerk-tabel. De eerste kolom uit de Automerk-tabel die geen key is, wordt vervolgens getoond, in dit geval is dat de 'Naam'-kolom. Dit werkt in veel gevallen erg goed. Je kunt je echter voorstellen dat bijvoorbeeld bij een personentabel de achternaam de eerste kolom is. Er kunnen meerdere mensen zijn met dezelfde achternaam, waardoor je er minder aan hebt als alleen de achternaam wordt getoond. Je kunt er dan voor kiezen om een andere kolom te laten tonen. Dit kun je aangeven via het DisplayColumn-attribuut op de door Linq gegenereerde klasse. Het Dynamic Data Framework probeert de data zo gebruiksvriendelijk mogelijk te tonen. Dit houdt ook in dat de primaire sleutelkolom niet getoond wordt. Deze heeft voor de gebruiker, als het goed is, toch geen enkele betekenis. In ons voorbeeld willen we echter de primaire sleutel



Afbeelding 2. De kolommen van het grid aanpassen

```
object val = DataValue;
if (val != null)
{
  RadioButton1.Checked = (bool)val;
  RadioButton2.Checked = !(bool)val;
}
```

Codevoorbeeld 4.

wel wordt getoond. We kunnen dit doen door zelf de kolommen te definiëren die getoond moeten worden.

Afbeelding 2 laat zien hoe je alle kolommen van het grid handmatig kunt definiëren in plaats van ze automatisch te laten genereren. Het handige van de DynamicGridView is dat deze afgeleid is van de standaard GridView. Alles wat je dus kunt aanpassen aan een standaard GridView, kun je ook aanpassen aan deze DynamicGridView. In de aangepaste lijst willen we alle introductiejaren die hoger zijn dan het jaar 2000 laten opvallen met een gele kleur. We gebruiken hiervoor het standaard GridView-event *RowDataBound*. Hierin kunnen de eigenschappen van een rij aangepast worden, afhankelijk van de data die in de rij staan. In codevoorbeeld 3 maken we elke rij geel waarvan het Introductiejaar hoger ligt dan 2000.

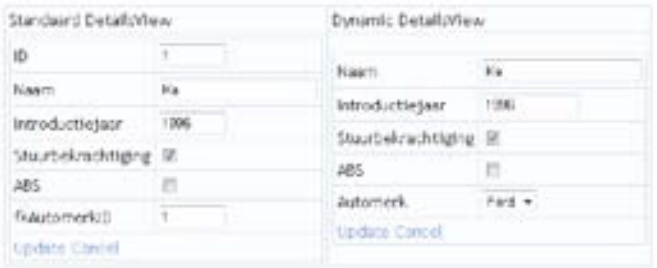
Detail Control

De DynamicDetailsView laat de gegevens van één rij zien. Het is afgeleid van het DetailsView-control dat standaard bij ASP.NET zit. Het control heeft ook de optie om de gegevens te wijzigen. Hiervoor kan de property 'AutoGenerateEditButton' op *true* worden gezet. Op eenzelfde manier ondersteunt dit control ook het verwijderen en het toevoegen van een nieuwe rij.

Op het eerste gezicht lijken de verschillen tussen een standaard DetailsView en een DynamicDetailsView klein. Een standaard DetailsView kan namelijk ook al de gegevens tonen van een rij uit een tabel, een rij wijzigen, toevoegen of verwijderen. Maar het voordeel van de DynamicDetailsView is het automatisch detecteren van de vreemde sleutels waarvoor een dropdownlist wordt getoond. Ook ben je flexibel in de manier waarop je de gegevens kunt presenteren of wijzigen. Dit is mogelijk doordat het Dynamic Data Framework templates gebruikt voor elk afzonderlijk datatype.

Templates

De kracht van de Dynamic Data Controls ligt in het gebruik van templates. Natuurlijk kun je gewoon een masterpage gebruiken voor jouw Dynamic Data-pagina's, maar tegelijkertijd kan de opmaak voor een Dynamic Data-pagina ook in een template gedefinieerd worden. Hierdoor ben je flexibel in de opmaak van de pagina's. Ook hoef je niet voor elk afzonderlijke tabel een aparte pagina aan te maken, omdat elke tabel standaard beschikbaar is via een template. Voor het tonen en wijzigen van gegevens uit één veld worden ook templates gebruikt. Er zijn templates voor elk afzonderlijk datatype. De templates zijn UserControls die vrij aan te passen zijn. Er is steeds één template dat gebruikt wordt om de gegevens van een bepaald datatype te tonen en één template voor de wijzigfunctionaliteit. Een veld van het type Boolean wordt standaard als checkbox getoond. We kunnen dit nu heel eenvoudig



Afbeelding 3. Verschil tussen standaard DetailsView en DynamicDetailsView

```
<asp:DynamicFilter runat="server" ID="DynamicFilter"
TableName="Autotype" DataField="Automerk" />
<asp:LinqDataSource ID="LijstDataSource" runat="server">
  <WhereParameters>
    <asp:DynamicControlParameter ControlID="DynamicFilter" />
  </WhereParameters>
</asp:LinqDataSource>
```

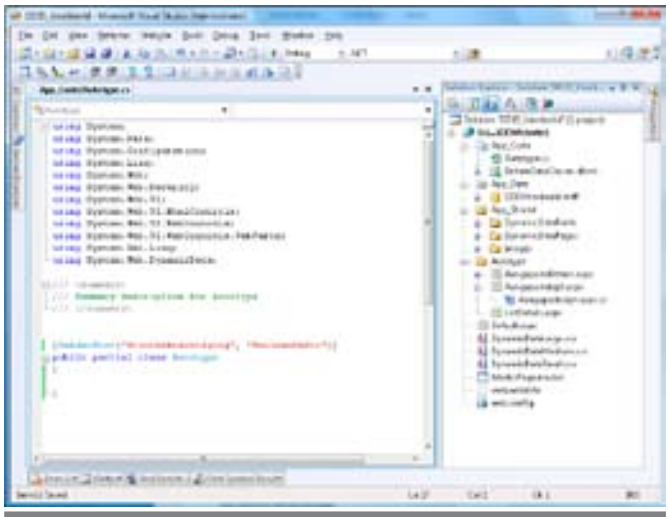
Codevoorbeeld 5.

wijzigen in radiobuttons. Hiervoor voegen we aan het 'Boolean.ascx' template-bestand twee RadioButtons toe en verwijderen we de CheckBox. De code die de waarde van de checkbox zet, wijzigen we zoals in codevoorbeeld 4.

Dit heeft als resultaat dat Boolean-waarden nu als RadioButtons worden gepresenteerd. Het wijzigen van een waarde is onveranderd gebleven. Dat gaat dus nog steeds via een CheckBox. In het voorbeeld van de tabel Autotype hebben we twee Boolean-kolommen. We willen eigenlijk alleen de kolom 'Stuurbekrachtiging' tonen met radiobuttons en de kolom 'ABS' op de oude manier met een checkbox. Dit is mogelijk door je eigen templates te maken. We creëren een 'BooleanRadio.ascx'- en een 'BooleanRadio_edit.ascx'-template met daarin de radiobuttons. De originele Boolean-templates laten we ongewijzigd. Via een RenderHint kunnen we nu aangeven dat voor de kolom 'Stuurbekrachtiging' altijd het BooleanRadio-template gebruikt moet worden. We maken hiervoor een partial class met de naam van de tabel waar de kolom inzit. In dit geval dus Autotype. Boven de partial class geven we het attribuut RenderHint mee, zoals in afbeelding 4. In afbeelding 5 zien we het uiteindelijke resultaat. De twee Boolean-kolommen worden nu elk door een ander template afgehandeld. Door het gebruik van templates heb je dus een krachtig mechanisme om de Dynamic Data Controls aan te passen aan je eigen wensen. Je zou de templates bijvoorbeeld kunnen gebruiken om een datetime-kolom via een kalender-interface aan te passen of een integer-kolom via een slider te laten wijzigen.

Filters

Het Dynamic Filter-control stelt je in staat de gegevens in een lijst gemakkelijk te filteren. Er kunnen meerdere filters aan een pagina toegevoegd worden. Elk filter-control filtert op één kolom, deze kolom hoeft niet per se zichtbaar te zijn in de lijst op de pagina. Op dit moment is het alleen mogelijk een filter te maken op een foreign key-kolom en een Boolean-veld. Door deze beperking is het met filters dus niet mogelijk een lijst gemakkelijk doorzoekbaar te maken op een vrij tekstveld. In het voorbeeld van de beheerpagina waar we mee bezig zijn, voegen we aan de pagina met autotype's een filter-control toe om te filteren op merk. Het filter wordt altijd als een dropdownlist getoond. In de configuratie van het filter verwijzen we niet naar de foreign key-kolom, maar naar



Afbeelding 4. RenderHint voor stuurbekrachtiging

Afbeelding 5. Verschil in presentatie van Boolean-velden

de tabelnaam van de gekoppelde tabel. Zoals we al eerder hebben gezien, wordt ook hier niet het ID getoond, maar de echte naam van het merk. Om de filtering te laten werken, moeten we dit filter koppelen aan de LinqDataSource. Dit kan door een DynamicControlParameter als WhereParameter bij de LinqDataSource op te nemen. Zie daarvoor codevoorbeeld 5.
Het gevolg is dat de query van de LinqDataSource wordt aangepast. Linq is vervolgens verantwoordelijk voor het ophalen van de gegevens uit de database. De query die Linq op de database zal uitvoeren, haalt alleen de gegevens op die voldoen aan het filter. Er worden niet onnodig veel gegevens uit de database gehaald, waardoor de performance goed blijft.

Dynamic Data en Linq

De Dynamic Data Controls en Linq werken erg goed samen. De controls, zoals de DynamicGridView en de DynamicDetailsView, hebben als input enkel een LinqDataSource nodig en doen dan hun werk. Ook de filters die je kunt aanmaken, werken direct op de LinqDataSource. Deze passen de where-claus aan van de Linq-query. Het Dynamic Data Framework zorgt er voor dat wanneer je een pagina opvraagt, de koppeling met de juiste tabel wordt gelegd. Het enige dat er dan eigenlijk gebeurt, is dat de TableName-property bij de LinqDataSource gezet wordt. De Dynamic Data Controls zijn daardoor ook erg bruikbaar op al bestaande pagina's. Je kunt dan zelf het TableName-property bij de LinqDataSource invullen en de Dynamic Data Controls doen de rest.

Iets moois

Zoals je hebt kunnen lezen, is het met het Dynamic Data Framework erg gemakkelijk om snel een data-editing user-interface te krijgen. De Dynamic GridView en DynamicDetailsView die gebruikt worden, zijn afgeleid van bestaande ASP.NET-controls. Hierdoor kun je gebruik blijven maken van alle opties die de standaard controls al bieden en is de drempel daartoe laag. Door het gebruik van templates heb je voldoende mogelijkheden de presentatie van de gegevens aan te passen aan jouw eigen wensen. De werking van de controls is goed geïntegreerd met Linq. Er kan een willekeurige Linq-datasource worden opgegeven en de controls werken meteen. Een nadeel is dat de controls die update- en insert-mogelijkheden hebben, afhankelijk zijn van het gebruik van viewstate. Als je echt maatwerk aan het maken bent, kost het waarschijnlijk te veel tijd om de controls helemaal naar jouw wens te laten werken. Wanneer de eisen echter iets lager zijn, zoals bij een prototype of beheeromgeving, kan er met de Dynamic Data Controls in weinig tijd iets moois in elkaar gezet worden.

Michiel Post is Software Engineer bij Qurius Advanced Solutions (http://as.qurius.nl).
michiel.post@qurius.nl

Referenties

Download voorbeeldapplicatie: http://www.softwarestraat.NET/magazine/
ASP.NET 3.5 Extensions: http://www.asp.NET/downloads/3.5-extensions/
Quickstart: http://quickstarts.asp.NET/3-5-extensions/dyndata/
David Ebbo's Blog: http://blogs.msdn.com/davidebb/