7.2009 | www.dotnet-magazin.de S&S

.NET & More Visual Studio

dot.net magazin

.NET, Visual Studio & More

AUF CD:

- Telerik Sitefinity CMS 3.6 Community Edition
- Mono 2.4
- sharpcms 0.4.0
- db4o-7.9 Beta
- DockPanel Suite 2.3
- Pidgin 2.5.5

BASTA! tv

LOB-Anwendungen mit Silverlight

Entwurf und Realisierung eines MVC-Frameworks ▸ 14

...darstellen mit DataGrid in Silverlight

Funktionalität und Optionen ▸ 33

Die Sache mit dem BLOB

Das Filestream Feature des SQL Servers 2008 im Einsatz ▸ 73

Alles im Fluss

Der SharePoint Workflow Web Service ▸ 102

Ansprechende Effekte für unterwegs

Alpha Blending und Transparenz für mobile Geräte ▸ 96

Agiles Anforderungs-management

User Stories und Use Cases in agilen Projekten ▸ 54

Datenträger enthält Info- und Lehrprogramme gemäß §14 JuSchG

Frischer Wind für Windows Forms

.NET-Komponenten durch eigene UserControls erweitern ▸ 45

Tuning für .NET

Profiler gegen träge Anwendungen ▸ 68

Listenreiches Control

Daten darstellen mit dem Silverlight DataGrid

Auf die Darstellung von Datensätzen in Form einer Liste kann in kaum einer Geschäftsanwendung verzichtet werden. Dafür wird meist das DataGrid verwendet. Das Silverlight SDK enthält ein solches DataGrid, das nützliche Eigenschaften besitzt. Auch wenn die DataGrids von Drittherstellern noch mehr Funktionalität bieten, genügt in vielen Fällen das Silverlight-eigene den Anforderungen. Teil 2 der Silverlight-Serie.

kurz & bündig

Inhalt

Ein Überblick über die Möglichkeiten des DataGrids von Silverlight

Zusammenfassung

Das in Silverlight enthaltene DataGrid bietet reichlich nützliche Funktionen. Der Einsatz von Controls von Drittanbietern ist daher nicht immer erforderlich, wie anhand der bereits eingeführten MagazinStore-Anwendung gezeigt wird

Quellcode

C#, XAML

Thomas Claudius Huber

Dieser Artikel ist der zweite Teil einer Artikelserie zum Thema Silverlight. Während in der letzten Ausgabe das Silverlight Toolkit beleuchtet wurde, widmet sich dieser Artikel dem DataGrid Control. Auch hier wird wieder die MagazineStore-Anwendung genutzt und gezeigt, wie dort das DataGrid zum Einsatz kommt. Bevor es so weit ist, zunächst die Grundlagen des DataGrids.

Der erste Einsatz

Nachdem ein neues Silverlight-Projekt angelegt und die *MainPage-xaml*-Datei geöffnet wurde, kann es losgehen. Das DataGrid Control lässt sich direkt aus der Toolbox von Visual Studio auf den XAML-Editor ziehen. Wurde das DataGrid aus der Toolbox hinzugefügt, wird in XAML der Namespace *System.Windows.Controls* aus der Assembly *System.Windows.Controls.Data* dem XML-Alias *data* zugeordnet. Unter diesem Alias wird mit dem XML-Tag *data:DataGrid* die DataGrid-Instanz erstellt (Listing 1). Neben den Änderungen in XAML werden durch das Hinzufügen des DataGrids aus der Toolbox auch automatisch ein paar Referenzen zum Silverlight-Projekt hinzugefügt. Unter anderem sind dies die Assemblies *System.Windows.Controls.Data* und *System.ComponentModel.*

DataAnnotations. Um zum Setzen der Datenquelle aus der Codebehind-Datei auf die in XAML erstellte DataGrid-Instanz zugreifen zu können, wird mit dem *x:Name*-Attribut ein Name vergeben (Listing 1).

… mit Daten füttern

Die Datenquelle des DataGrids wird über die *ItemsSource*-Eigenschaft gesetzt. Diese Eigenschaft ist vom Typ *IEnumerable*. Obwohl auch Controls wie *ListBox* oder *ComboBox* zum Setzen der Datenquelle eine *ItemsSource*-Eigenschaft vom Typ *IEnumerable* besitzen, ist der Ursprung dort ein anderer. Sowohl *ListBox* als auch *ComboBox* sind Subklassen von *ItemsControl* und erben über diese Klasse die *ItemsSource*-Eigenschaft. Das DataGrid dagegen definiert die *ItemsSource*-Eigenschaft selbst und erbt direkt von Control. Als Datencontainer dient an dieser Stelle die Klasse *Person* (Listing 2).

In der Codebehind-Datei *MainPage.xaml.cs* wird im Konstruktor die Datenquelle des DataGrids gesetzt. Dazu wird eine Liste mit mehreren *Person*-Objekten erstellt. Diese Liste wird der *ItemsSource*-Eigenschaft des DataGrids zugewiesen. Zu beachten ist, dass zum Zugriff auf das DataGrid der zuvor in XAML definierte Name *dataGrid* verwendet wird (Listing 3).

Einem Start der Anwendung steht jetzt nichts mehr im Wege. Wie zu erwarten war, zeigt das DataGrid die Daten entsprechend an. Die Spalten im DataGrid wurden automatisch generiert. Für die *HasSilverlightKnowledge*-Eigenschaft vom Typ *bool* wird statt der Textdarstellung eine CheckBox angezeigt (Abbildung 1).

Abb. 1: Das DataGrid mit den Standardeinstellungen

FirstName	LastName	Birthday	HasSilverlightKnowledge
Thomas	Huber	10/28/1980 12:00:00 AM	☑
Christoph	Pletz	1/1/1966 12:00:00 AM	☑
Michael	Ballack	9/26/1976 12:00:00 AM	☐
Lukas	Podolski	6/4/1985 12:00:00 AM	☐
Bastian	Schweinsteiger	8/1/1984 12:00:00 AM	☐
Arne	Friedrich	5/29/1979 12:00:00 AM	☐
René	Adler	1/15/1985 12:00:00 AM	☐
Joachim	Löw	2/3/1960 12:00:00 AM	☐

Abb. 2: Das DataGrid mit speziellen Werten für RowBackground und AlternatingRowBackground

FirstName	LastName	Birthday	HasSilverlightKnowledge
Thomas	Huber	10/28/1980 12:00:00 AM	☑
Christoph	Pletz	1/1/1966 12:00:00 AM	☑
Michael	Ballack	9/26/1976 12:00:00 AM	☐
Lukas	Podolski	6/4/1985 12:00:00 AM	☐
Bastian	Schweinsteiger	8/1/1984 12:00:00 AM	☐
Arne	Friedrich	5/29/1979 12:00:00 AM	☐
René	Adler	1/15/1985 12:00:00 AM	☐
Joachim	Löw	2/3/1960 12:00:00 AM	☐

Verhalten und Aussehen anpassen

Zeigt das DataGrid die Daten bereits an, lassen sich spielerisch noch die gewünschten Einstellungen durch Setzen diverser Eigenschaften vornehmen. Grob lassen sich die Eigenschaften in drei Kategorien aufteilen: Eigenschaften, die das Verhalten des DataGrids betreffen, Eigenschaften, die das Aussehen beeinflussen, und jene Eigenschaften, welche die Darstellung der Daten beeinflussen. Zum Steuern des Verhaltens gibt es unter anderem die Eigenschaften *CanUserReorderColumns*, *CanUserResizeColumns* und *CanUserSortColumns*. Die Eigenschaften sind per Default alle *true*. Somit ist der Benutzer in der Lage, die Spalten zu verschieben, die Spaltengröße anzupassen oder durch einen Klick auf einen Spaltenkopf die Daten im DataGrid nach der entsprechenden Spalte zu sortieren. Dabei ist für die Sortierfunktion keine weitere Logik notwendig, das DataGrid beinhaltet die entsprechende Funktionalität.

Die im DataGrid dargestellten Daten lassen sich standardmäßig editieren. Um das Editieren zu unterbinden, wird die *IsReadOnly*-Eigenschaft auf *true* gesetzt. Die letzte Eigenschaft, die für das Verhalten von Bedeutung ist, ist *SelectionMode*. Sie ist vom Typ der Enumeration *DataGridSelectionMode*, welche die Werte *Extended* und *Single* enthält. Wird der Wert *Extended* gesetzt, lassen sich mit gedrückter CTRL-Taste mehrere Zeilen auswählen. Beim Wert *Single* hingegen ist immer nur eine Zeile selektiert.

Um das Aussehen des DataGrids anzupassen, gibt es verschiedene Möglichkeiten. Für ein komplett neues Design wird ein neues ControlTemplate definiert. Allerdings ist dies eher selten notwendig. Es gibt zahlreiche Eigenschaften, welche visuelle Anpassungen

Listing 1

```
<UserControl xmlns:data="clr-namespace:System.
                Windows.Controls;assembly=System.
                        Windows.Controls.Data"
    ...>
 ...
  <data:DataGrid x:Name="dataGrid"></
data:DataGrid>
 ...
</UserControl>
```

Listing 2

```
public class Person
{
 public string FirstName { get; set; }
 public string LastName { get; set; }
 public DateTime Birthday { get; set; }
 public bool HasSilverlightKnowledge { get; set; }
}
```

Listing 3

```
public MainPage()
{
 InitializeComponent();
 dataGrid.ItemsSource = new List<Person>
 {
  new Person {FirstName = "Thomas",
   LastName = "Huber",
   Birthday = new DateTime(1980, 10, 28),
   HasSilverlightKnowledge = true
  },
  new Person {FirstName = "Christoph",
   LastName = "Pletz",
   Birthday = new DateTime(1966, 1, 1),
   HasSilverlightKnowledge = true
  },
  ...
 };
}
```

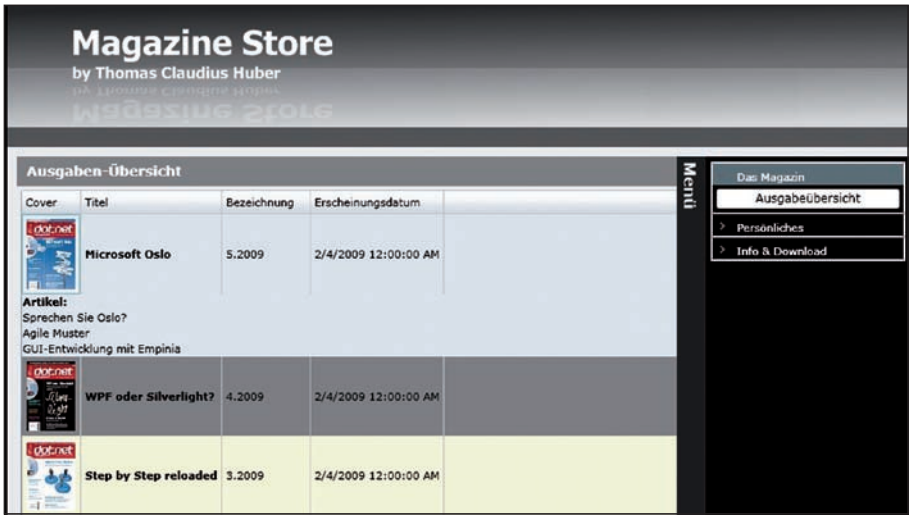

Abb. 5: Bilder im DataGrid bei der MagazinStore-Anwendung

terstützt. Soll die *FirstName*-Eigenschaft einer Person-Instanz beispielsweise nie leer sein, wird auf der Eigenschaft das *Required*-Attribut definiert (Listing 5).

Wird im DataGrid in einer Zeile der Vorname einer Person gelöscht und versucht, auf eine neue Zeile zu springen, greift die Validierung. Die Zeile wird im ungültigen Zustand rot dargestellt und der Fehler am unteren Rande des DataGrids angezeigt. Im Falle der *FirstName*-Eigenschaft der *Person*-Klasse wird dem Benutzer mitgeteilt, dass der Vorname ein Pflichtfeld ist (Abbildung 4). Um die dargestellte „Fehler"-Nachricht anzupassen, besitzt das *Required*-Attribut die *ErrorMessage*-Eigenschaft. So lässt sich beispielsweise eine angepasste oder lokalisierte Fehlermeldung ausgeben.

Spalten manuell definieren

Bisher wurden die Spalten im DataGrid automatisch generiert. Das DataGrid besitzt eine Eigenschaft namens *AutoGenerateColumns* vom Typ *bool*. Sie hat per Default den Wert *true*. Sollen aus der dargestellten Entität nicht alle Eigenschaften in einer Spalte dargestellt werden, oder muss eine Spalte beispielsweise ein Bild enthalten, müssen die Spalten manuell definiert werden. So geschieht es in der MagazineStore-Anwendung in der Ausgabenübersicht, wo das Cover des jeweiligen Magazins in einem DataGrid angezeigt wird (Abbildung 5).

Bevor das Geheimnis zum Anzeigen eines Bildes gelüftet wird, zunächst zu den Daten. Der *ItemsSource*-Eigenschaft des DataGrids der MagazineStore-Anwendung liegt eine Liste mit *Magazine*-Instanzen zu Grunde. Die *Magazine*-Klasse definiert unter anderem die Eigenschaft *Title* und *Description* sowie die Eigenschaft *Cover*, die vom Typ *byte[]* ist (Listing 6) und das Bild enthält.

Um die Spalte *Cover* als Bild im DataGrid darzustellen, müssen die Spalten im DataGrid manuell definiert werden. Dazu besitzt das DataGrid eine *Columns* Property. Es gibt derzeit drei mögliche Spaltentypen. Den *DataGridTextColumn* zum Darstellen von Text, den *DataGridCheckBoxColumn* zum Darstellen von booleschen Werten, und den *DataGridTemplateColumn*. Die Klasse *DataGridTemplateColumn* besitzt eine Eigenschaft namens *CellTemplate*, die ein *DataTemplate* entgegennimmt. Darin lässt sich das Aussehen der Zelle mit den üblichen Silverlight-Elementen frei definieren. Im Falle der MagazineStore-Anwendung wird im DataTemplate für die *Cover*-Spalte lediglich ein *Image*-Objekt hinzugefügt (Listing 7). Dieses wird mittels Data Binding an die *Cover*-Eigenschaft der jeweiligen *Magazine*-Instanz gebunden. Da die *Cover*-Eigenschaft vom Type *byte[]* ist, die *Source*-Eigenschaft der *Image*-Klasse allerdings vom Typ *ImageSource*, wird ein *ValueConverter* benötigt. Dieser wurde in der MagazineStore-Anwendung in

MagazineStore

Während der Artikelserie wird die Magazine Store-Anwendung stetig weiterentwickelt. Die entsprechende Version zu diesem Artikel ist online verfügbar unter der Adresse http://87.230.82.15/MagazineStore/v0.2.0.0

Validieren auf Zellenebene

Die Validierungslogik greift erst, wenn der Benutzer versucht, die bearbeitete Zeile zu verlassen. Um beim Verlassen der Zelle eine Validierung auszulösen, muss im set-Accessor der betroffenen Eigenschaft entweder eine *ValidationException* ausgelöst werden, oder die *ValidateProperty*-Methode der Validator-Klasse aufgerufen werden. Nähere Infos dazu gibt es unter [1].

Listing 7

```
public class Magazine
{
 public string Title { get; set; }
 public string Description { get; set; }
 public DateTime ReleaseDate { get; set; }
 public byte[] Cover { get; set; }
 public IEnumerable<Article> Articles { get; set; }
}
```

Listing 8

```
<data:DataGrid ItemsSource="{Binding Magazines}"
      AutoGenerateColumns="False"
      RowBackground="Beige"
      AlternatingRowBackground="Gray"
      IsReadOnly="True"
      SelectionMode="Single">
 <data:DataGrid.Columns>
  <data:DataGridTemplateColumn Header="Cover">
   <data:DataGridTemplateColumn.CellTemplate>
    <DataTemplate>
     <Image Margin="5"
       Source="{Binding Cover,Converter=
                {StaticResource byteToImageConv}}" />
    </DataTemplate>
   </data:DataGridTemplateColumn.CellTemplate>
  </data:DataGridTemplateColumn>
  <data:DataGridTextColumn Binding="{Binding Title}"
            FontWeight="Bold"
            Header="Titel" />
  <data:DataGridTextColumn Binding=
                "{Binding Description}"
            Header="Bezeichnung" />
  <data:DataGridTextColumn Binding=
                "{Binding ReleaseDate}"
            Header="Erscheinungsdatum" />
 </data:DataGrid.Columns>
 ...
</data:DataGrid>
```

am DataGrid erlauben, ohne gleich das ganze *ControlTemplate* neu definieren zu müssen. Mit den Eigenschaften *RowBackground* und *AlternatingRowBackground* (Listing 3) wird beispielsweise die Hintergrundfarbe für die Zeilen angegeben (Abbildung 2). Mit den Eigenschaften *CellStyle* oder *ColumnHeaderStyle* lassen sich Zellen und Spaltenköpfe anpassen.

Erwähnenswert ist auch die Eigenschaft *GridLinesVisibility*. Sie nimmt einen Wert der Enumeration *DataGridGridLinesVisibility* entgegen und legt darüber fest, ob zwischen den Zeilen und Spalten Linien angezeigt werden. Mögliche Werte sind *All*, *None*, *Horizontal* und *Vertical*. Über die Eigenschaften *HorizontalGridLinesBrush* und *VerticalGridLinesBrush* lässt sich der Pinsel bestimmen, mit dem die Linien gezeichnet werden. Neben vielen weiteren Eigenschaften, wie *ColumnWidth* oder *RowHeight*, ist die hier zuletzt betrachtete die *HeadersVisibility*-Eigenschaft. Sie ist vom Typ der Enum *DataGridHeadersVisibility*. Beispielsweise lassen sich mit dem Wert None die Spaltenköpfe ausblenden. Zu beachten ist, dass dadurch der Benutzer natürlich nicht mehr einfach sortieren kann, da ein Klicken auf den Spaltenkopf nicht mehr möglich ist.

Daten gruppieren

In der dritten Version von Silverlight besitzt das DataGrid auch die Möglichkeit, Daten zu gruppieren. Die Vorgehensweise ist relativ simpel. Es werden lediglich zur *GroupDescriptions*-Eigenschaft des DataGrids *PropertyGroupDescriptor*-Objekte aus dem Namespace *System.Windows.Data* hinzugefügt. Die *PropertyGroupDescriptor*-Objekte definieren über ihre *PropertyName*-Eigenschaft den Namen der Eigenschaft, nach der gruppiert werden soll. Um in XAML die Klasse *PropertyGroupDescriptor* zu verwenden, muss der *System.Windows.Data*-Namespace aus der Assembly *System.ComponentModel* bekannt gemacht werden. In Listing 4 wird dazu der Alias *comp* verwendet.

Anschließend steht einer Gruppierung nichts mehr im Wege. Hier wird die *HasSilverlightKnowledge*-Eigenschaft der *Person*-Klasse zum Gruppieren genutzt (Listing 4). Das DataGrid zeigt anschließend über jeder Gruppe die Summe der darin enthaltenen Datensätze an (Abbildung 3). Gruppen lassen sich per Mausklick auf- und zuklappen.

Abb. 3: Gruppierte Daten

FirstName	LastName	Birthday	HasSilverlightKnowledge
HasSilverlightKnowledge: True (2 items)			
Thomas	Huber	10/28/1980 12:00:00 AM	☑
Christoph	Pletz	1/1/1966 12:00:00 AM	☑
HasSilverlightKnowledge: False (6 items)			
Michael	Ballack	9/26/1976 12:00:00 AM	☐
Lukas	Podolski	6/4/1985 12:00:00 AM	☐
Bastian	Schweinsteiger	8/1/1984 12:00:00 AM	☐
Arne	Friedrich	5/29/1979 12:00:00 AM	☐
René	Adler	1/15/1985 12:00:00 AM	☐
Joachim	Löw	2/3/1960 12:00:00 AM	☐

Abb. 4: Validierung mit Required-Attribut

FirstName	LastName	Birthday	HasSilverlightKnowledge
	Huber	10/28/1980 12:00:00 AM	☑
Christoph	Pletz	1/1/1966 12:00:00 AM	☑
Michael	Ballack	9/26/1976 12:00:00 AM	☐
Lukas	Podolski	6/4/1985 12:00:00 AM	☐
Bastian	Schweinsteiger	8/1/1984 12:00:00 AM	☐
Arne	Friedrich	5/29/1979 12:00:00 AM	☐
René	Adler	1/15/1985 12:00:00 AM	☐
Joachim	Löw	2/3/1960 12:00:00 AM	☐
The FirstName field is required.			

Daten validieren

Wurde die *IsReadOnly*-Eigenschaft des DataGrids nicht auf *true* gesetzt, lassen sich die Daten direkt im DataGrid bearbeiten. Die eingegebenen Daten sollten entsprechend validiert werden. Mit Silverlight 3 wird eine Validierung mit den Attributen aus dem Namespace *System.ComponentModel.DataAnnotations* un-

Listing 4

```
<data:DataGrid x:Name="dataGrid"
      RowBackground="LightBlue"
      AlternatingRowBackground="SkyBlue" />
```

Listing 5

```
<UserControl xmlns:data="clr-namespace:System.
                    Windows.Controls;assembly=System.
                                Windows.Controls.Data"
    xmlns:comp="clr-namespace:System.Windows.
            Data;assembly=System.ComponentModel"
...>
 ...
  <data:DataGrid x:Name="dataGrid"
        RowBackground="LightBlue"
        AlternatingRowBackground="SkyBlue">
   <data:DataGrid.GroupDescriptions>
    <comp:PropertyGroupDescription PropertyName=
                        "HasSilverlightKnowledge" />
   </data:DataGrid.GroupDescriptions>
  </data:DataGrid>
 ...
</UserControl>
```

ItemsSource in XAML setzen

Die *ItemsSource*-Eigenschaft des DataGrids lässt sich anstatt in der Codebehind-Datei natürlich auch in XAML setzen. Allerdings ist die Variante mit der Codebehind-Datei die übliche und praktischere, da die Daten noch dynamisch geladen werden können.
Eine andere Variante zum Setzen der *ItemsSource*-Eigenschaft ist über Databinding, wie dies im Model-View-ViewModel-Pattern praktiziert wird. Das ViewModel würde in diesem Fall eine Eigenschaft vom Typ *IEnumerable* besitzen. Die *ItemsSource*-Eigenschaft des DataGrids kann infolgedessen an die entsprechende Eigenschaft des im DataContext befindlichen ViewModels gebunden werden.

Listing 6

```
public class Person
{
 [Required]
 public string FirstName { get; set; }
 public string LastName { get; set; }
 public DateTime Birthday { get; set; }
 public bool HasSilverlightKnowledge { get; set; }
}
```

den Ressourcen des UserControls mit dem Namen *byteToImageConv* instanziiert. Er lässt sich somit im Data Binding mit der *StaticResource*-Erweiterung referenzieren. Die Klasse zur Konvertierung ist sehr simpel und in Listing 8 zu sehen.

Damit ist die Anzeige des Bildes erledigt. Zu beachten ist, dass die *AutoGenerateColumns*-Eigenschaft auf dem DataGrid auf *false* gesetzt wird (Listing 7).

Details anzeigen

Neben der Anzeige des Bildes nutzt die MagazineStore-Anwendung eine weitere Besonderheit des DataGrids. Das DataGrid ermöglicht über die Eigenschaft *RowDetailsTemplate* die Definition einer Detailanzeige für einen Datensatz. Die *Magazine*-Klasse (Listing 6) enthält eine *Articles*-Eigenschaft, die die Artikel des entsprechenden Heftes beinhaltet. Die *Article*-Klasse ist einfach gehalten und besitzt lediglich eine *Title*-Eigenschaft vom Typ String:

```
public class Article
{
 public string Title { get; set; }
}
```

Um im DataGrid die Artikel unterhalb eines Heftes in einer Master-Detail-Form anzuzeigen, wird im DataGrid die *RowDetailsTemplate*-Eigenschaft gesetzt (Listing 9).

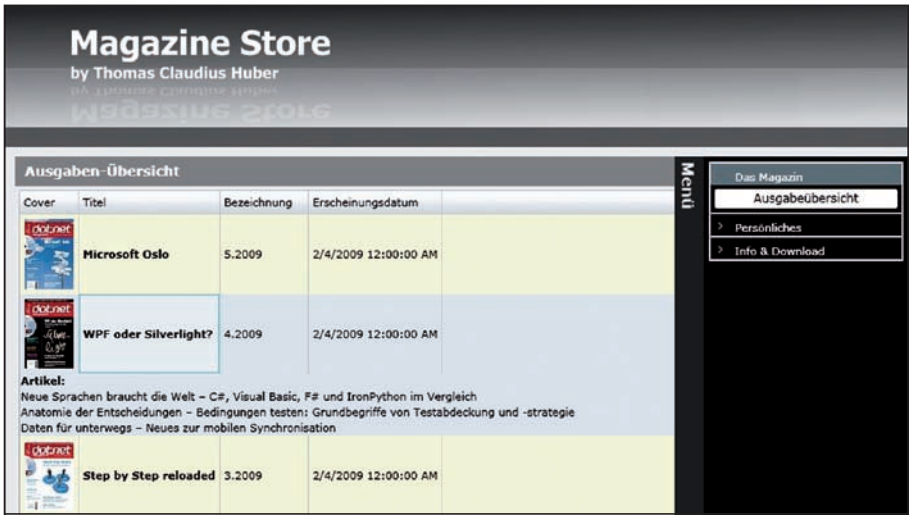

Abb. 6: Das mittlere Magazin wurde ausgewählt, die Artikel werden angezeigt

Darin werden im MagazineStore ein TextBlock sowie ein ItemsControl verwendet. Das ItemsControl enthält dabei als Quelle über Data Binding die *Articles*-Eigenschaft des entsprechend angezeigten Magazins. Wird auf ein Magazin geklickt, werden automatisch unterhalb des Magazins die entsprechenden Artikel der Ausgabe in der ListBox angezeigt (Abbildung 6).

Mit der Eigenschaft *RowDetailsVisibilityMode* lässt sich einstellen, wann die Details angezeigt werden. Per Default werden sie erst bei der Auswahl des Datensatzes angezeigt. Sie lassen sich auch dauerhaft anzeigen (*Visible*) oder dauerhaft ausblenden (*Collapsed*).

Fazit

Das DataGrid hat mit der Gruppiermöglichkeit und mit den Validierungsmechanismen in Silverlight 3 deutlich zugelegt. Richtig eingesetzt ist und bleibt es eine gute Alternative zu den DataGrids von Drittanbietern. Sehr nützlich sind die Möglichkeiten, Details anzeigen zu können, zu sortieren usw. Und mit dem *DataGridTemplateColumn* besitzt der Entwickler alle Freiheiten. Neben den in diesem Artikel dargestellten Eigenschaften besitzt das DataGrid natürlich zahlreiche nützliche Events. In der Version 3 von Silverlight hat es auch dort ein paar neue Ereignisse hinzugekommen. Die Entwicklung der Komponente scheint voranzugehen.

Im nächsten Teil dieser Artikelserie soll der Konsum von ADO.NET Data Services aus Silverlight näher beleuchtet werden. Die MagazineStore-Anwendung wird dazu über ADO.NET Data Services mit einer SQL-Server-Datenbank verbunden und lädt die Magazin-Daten von dort. ●

Listing 9

```
public class ByteArrayToImageSourceConverter:
                                    IValueConverter
{
 public object Convert(...)
 {
 if(value is byte[])
 {
  BitmapImage image = new BitmapImage();
  MemoryStream ms = new MemoryStream((byte[])
value);
  ms.Seek(0, SeekOrigin.Begin);
  image.SetSource(ms);
  return image;
 }
 return value;
 }
 ...
}
```

Listing 10

```
<data:DataGrid ItemsSource="{Binding Magazines}"
      ...>
 <data:DataGrid.Columns>
  ...
 </data:DataGrid.Columns>
 <data:DataGrid.RowDetailsTemplate>
  <DataTemplate>
   <Grid>
    <Grid.RowDefinitions>
     <RowDefinition Height="Auto" />
     <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <TextBlock Text="Artikel:"
        FontWeight="Bold" />
    <ItemsControl Grid.Row="1"
         ItemsSource="{Binding Articles}"
         DisplayMemberPath="Title" />
   </Grid>
  </DataTemplate>
 </data:DataGrid.RowDetailsTemplate>
</data:DataGrid>
```

Thomas Claudius Huber ist Senior Consultant bei der Trivadis AG in Basel. Neben spannenden WPF- und Silverlight-Projekten genießt er derzeit frische Sommerluft auf einer Night Train. Über Anregungen freut er sich: thomas.huber@trivadis.com.

Links & Literatur

ⓘ Validierung im DataGrid: http://blogs.msdn.com/nagasatish/archive/2009/03/22/datagrid-validation.aspx