Silverlight 4

پژوهش و نگارش: وحید نصیری
Vahid_Nasiri@Yahoo.Com
بهار ۱۳۸۹

فهرست مطالب

فصل ۱۸ – بررسی جامع توانایی‌های کنترل DataGrid

مقدمه

کنترل DataGrid یکی از اساسی‌ترین اجزای برنامه‌های کاربردی و تجاری است. عموما از این کنترل جهت ارائه‌ی گزارشات پیشرفته استفاده می‌شود؛ به همراه قابلیت‌های گروه بندی اطلاعات، مرتب سازی، صفحه بندی، قابلیت‌های درج، ویرایش و حذف اطلاعات و غیره. DataGrid از نگارش دوم Silverlight حتی پیش از ایجاد نسخه‌ی سازگار با WPF آن، دردسترس برنامه نویسان بوده است. DataGrid در فضای نام System.Windows.Controls قرار دارد؛ اما جزو اسمبلی‌های هسته‌ی Silverlight به شمار نمی‌رود. این کنترل از ویژگی‌ها UI virtualization بهره مند است به این معنا که اگر لیستی بسیار طولانی از اطلاعات را توسط آن نمایش دهیم، در هر لحظه تنها آیتم‌های نمایان و قابل مشاهده‌ی آن ایجاد شده و این امر از افت کارآیی آن جلوگیری می‌کند. در طی فصول قبل، مروری سریع بر این کنترل در حد انتساب لیست یک سری از اشیاء به خاصیت ItemsSource آن داشتیم. در این فصل قصد داریم تا سایر توانایی‌های ارزنده‌ی این کنترل را بررسی نمائیم.

نمایش لیستی از اطلاعات با ظاهری سفارشی در یک DataGrid

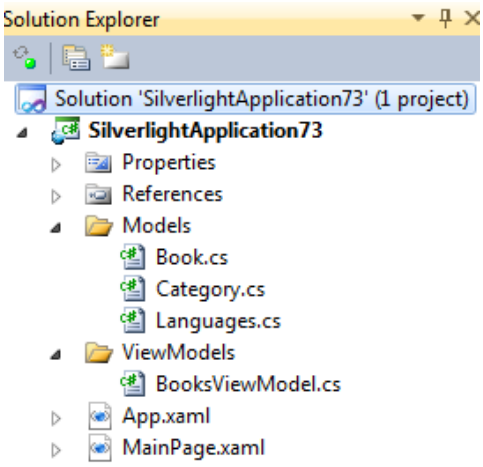
قصد داریم اطلاعات لیستی از کتاب‌ها را به نحو ذیل در یک DataGrid نمایش دهیم:

- کاربران نتوانند تغییری در مکان قرارگیری ستون‌ها ایجاد کنند و حالت نمایشی آن باید ثابت باشد.
- رنگ پس زمینه‌ی ردیف‌هایی که جزو گروه Thriller هستند باید قرمز شود.
- رنگ ردیف‌های گرید باید یک درمیان مطابق رنگی مشخص تغییر کند.
- تنها خطوط افقی جدول نمایش داده شوند.

جهت پیاده سازی این مثال از الگوی MVVM استفاده خواهیم کرد و کلاس‌های آن در مثال‌های بعدی فصل نیز بکارگرفته خواهند شد. از آنجائیکه نیاز است از ویژگی EventToCommand مرتبط با MVVM Light toolkit استفاده شود، ارجاعی را به اسمبلی‌های آن نیز باید اضافه نمائید. این اسمبلی‌ها در مسیر زیر قرار دارند (یا از قالب‌های MVVM آن استفاده نموده و یک پروژه را بر این اساس آغاز نمائید):

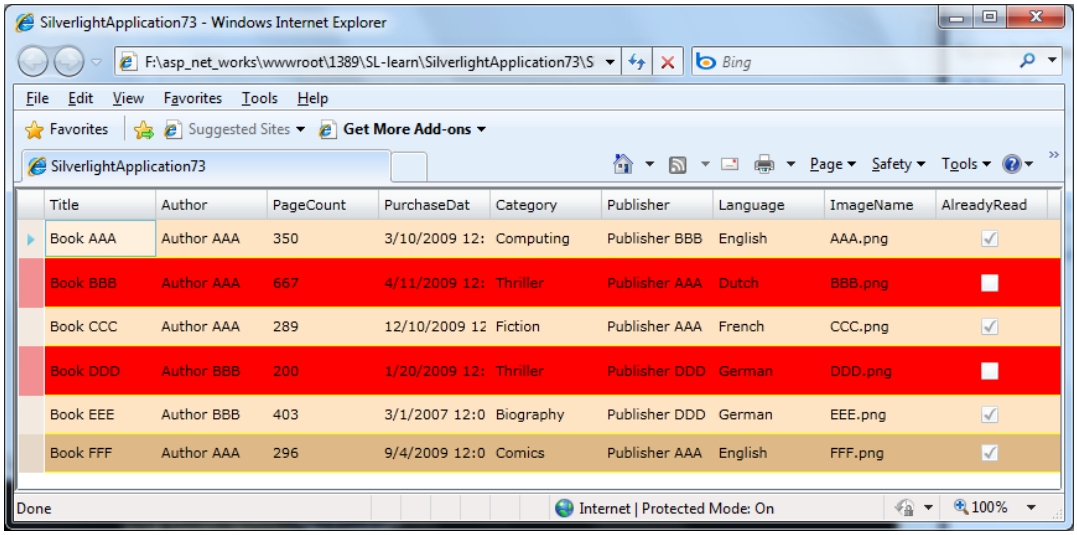
C:\Program Files\Laurent Bugnion (GalaSoft)\Mvvm Light Toolkit\Binaries\Silverlight4

پس از این مقدمات، تصاویری از ساختار فایل‌های این پروژه را به همراه نمونه‌ی در حال اجرای آن، در شکل‌های ۱ و ۲ مشاهده خواهید نمود.



شکل ۱– ساختار فایل‌ها و پوشه‌های اولین پروژه استفاده از DataGrid

سلول‌های کنترل DataGrid در حالت پیش فرض قابل ویرایش هستند و همچنین با استفاده از کشیدن و رها کردن نیز می‌توان جای ستون‌ها را تغییر داد. این موارد در گزارش‌های ساده از اطلاعات عموما غیر ضروری می‌باشند. به همین جهت قصد داریم در این مثال این موارد را کنترل کرده و همچنین بر اساس اطلاعات سلول‌ها، رنگ پس زمینه‌ی ویژه‌ای را به آن‌ها اعمال نمائیم.



شکل ۲– نمایی از اولین مثال استفاده از DataGrid

تعاریف زبان‌ها و گروه‌های مورد استفاده در کلاس کتاب را در کدهای بعد مشاهده می‌نمائید:

Languages.cs

```
namespace SilverlightApplication73.Models
{
    public enum Languages
    {
        English,
        Persian,
        Dutch,
        French,
        German
    }
}
```

Category.cs

```
namespace SilverlightApplication73.Models
{
    public enum Category
    {
        Thriller,
        Fiction,
        Comics,
        Computing,
        Biography
    }
}
```

کدهای کلاس کتاب و خواص آن‌را در ادامه مشاهده می‌نمائید. این کلاس اینترفیس INotifyPropertyChanged را پیاده سازی می‌کند :

Book.cs

```
using System;
using System.ComponentModel;

namespace SilverlightApplication73.Models
{
    public class Book : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        public void OnPropertyChanged(string propertyName)
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new
                    PropertyChangedEventArgs(propertyName));
```

```
            }
        }

        private string _title;
        private string _author;
        private int _pageCount;
        private DateTime _purchaseDate;
        private Category _category;
        private string _publisher;
        private Languages _language;
        private string _imageName;
        private bool _alreadyRead;

        public string Title
        {
            get { return _title; }
            set
            {
                _title = value;
                OnPropertyChanged("Title");
            }
        }
        public string Author
        {
            get { return _author; }
            set
            {
                _author = value;
                OnPropertyChanged("Author");
            }
        }
        public int PageCount
        {
            get { return _pageCount; }
            set
            {
                _pageCount = value;
                OnPropertyChanged("PageCount");
            }
        }
        public DateTime PurchaseDate
        {
            get { return _purchaseDate; }
            set
            {
                _purchaseDate = value;
                OnPropertyChanged("PurchaseDate");
            }
        }
        public Category Category
```

```
        {
            get { return _category; }
            set
            {
                _category = value;
                OnPropertyChanged("Category");
            }
        }
        public string Publisher
        {
            get{ return _publisher; }
            set
            {
                _publisher = value;
                OnPropertyChanged("Publisher");
            }
        }
        public Languages Language
        {
            get { return _language; }
            set
            {
                _language = value;
                OnPropertyChanged("Language");
            }
        }
        public string ImageName
        {
            get { return _imageName; }
            set
            {
                _imageName = value;
                OnPropertyChanged("ImageName");
            }
        }
        public bool AlreadyRead
        {
            get { return _alreadyRead; }
            set
            {
                _alreadyRead = value;
                OnPropertyChanged("AlreadyRead");
            }
        }
    }
}
```

اکنون به مهم‌ترین قسمت برنامه می‌رسیم. در این ViewModel با کمک ویژگی EventToCommand مرتبط با MVVM Light toolkit ، توانسته‌ایم رخداد LoadingRow کنترل DataGrid را مدیریت کنیم. این رخداد در حین بارگذاری هر ردیف، سبب فراخوانی روال رویدادگردان loadingRowCommand می‌شود. از طریق آرگومان ورودی آن می‌توان به شیء جاری هر ردیف در حین Binding دسترسی داشت.

BooksViewModel.cs

```
using System;
using System.Collections.ObjectModel;
using System.Windows.Controls;
using System.Windows.Media;
using GalaSoft.MvvmLight.Command;
using SilverlightApplication73.Models;

namespace SilverlightApplication73.ViewModels
{
    public class BooksViewModel
    {
        public ObservableCollection<Book> BookCollection { set; get; }
        public RelayCommand<DataGridRowEventArgs>
                LoadingRowCommand { get; private set; }

        public BooksViewModel()
        {
            LoadingRowCommand = new
                RelayCommand<DataGridRowEventArgs>(loadingRowCommand);

            BookCollection =
                new ObservableCollection<Book>
                {
                  new Book
                  {
                     Title = "Book AAA",
                     Author = "Author AAA",
                     Language = Languages.English,
                     PageCount = 350,
                     Publisher = "Publisher BBB",
                     PurchaseDate = new DateTime(2009, 3, 10),
                     ImageName = "AAA.png",
                     AlreadyRead = true,
                     Category = Category.Computing
                  },
                  new Book
                  {
                     Title = "Book BBB",
                     Author = "Author AAA",
                     Language = Languages.Dutch,
                     PageCount = 667,
                     Publisher = "Publisher AAA",
```

```
            PurchaseDate = new DateTime(2009, 4, 11),
            ImageName = "BBB.png",
            AlreadyRead = false,
            Category = Category.Thriller
        },
        new Book
        {
            Title = "Book CCC",
            Author = "Author AAA",
            Language = Languages.French,
            PageCount = 289,
            Publisher = "Publisher AAA",
            PurchaseDate = new DateTime(2009, 12, 10),
            ImageName = "CCC.png",
            AlreadyRead = true,
            Category = Category.Fiction
        },
        new Book
        {
            Title = "Book DDD",
            Author = "Author BBB",
            Language = Languages.German,
            PageCount = 200,
            Publisher = "Publisher DDD",
            PurchaseDate = new DateTime(2009, 1, 20),
            ImageName = "DDD.png",
            AlreadyRead = false,
            Category = Category.Thriller
        },
        new Book
        {
            Title = "Book EEE",
            Author = "Author BBB",
            Language = Languages.German,
            PageCount = 403,
            Publisher = "Publisher DDD",
            PurchaseDate = new DateTime(2007, 3, 1),
            ImageName = "EEE.png",
            AlreadyRead = true,
            Category = Category.Biography
        },
        new Book
        {
            Title = "Book FFF",
            Author = "Author AAA",
            Language = Languages.English,
            PageCount = 296,
            Publisher = "Publisher AAA",
```

```
                    PurchaseDate = new DateTime(2009, 9, 4),
                ImageName = "FFF.png",
                AlreadyRead = true,
                Category = Category.Comics
            }
        };
    }

    private static void loadingRowCommand(DataGridRowEventArgs e)
    {
        var loadedBook = e.Row.DataContext as Book;
        if (loadedBook == null) return;
        if (loadedBook.Category == Category.Thriller)
        {
            e.Row.Background = new SolidColorBrush(Colors.Red);
            //It's a thriller!
            e.Row.Height = 40;
        }
        else
        {
            e.Row.Background = null;
        }
    }
}
}
```

در این ViewModel مشخصات یک سری کتاب به لیستی از نوع ObservableCollection اضافه شده‌است تا در اختیار View برنامه قرار گیرد. همچنین یک RelayCommand نیز از نوع DataGridRowEventArgs برای مدیریت روال رخداد گردان loadingRow تعریف شده است. نحوه‌ی دریافت شیء جاری کتاب در حال انقیاد را در متد loadingRowCommand می‌توان مشاهده نمود. پس از دریافت شیء جاری، گروه آن بررسی شده و اگر در گروه Thriller قرار داشت، رنگ زمینه‌ی آن ردیف قرمز می‌گردد. در عمل شاهد مثال‌های زیادی از این دست خواهید بود برای مثال رنگی کردن محصولاتی که تاریخ مصرفشان گذشته است یا تغییر رنگ آیتم‌هایی که موجودی آن‌ها به اتمام رسیده است و امثال آن.

کدهای XAML برنامه را در ادامه مشاهده می‌نمائید:

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication73.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc=
      "http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400"
```

```
    xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk"
    xmlns:vm="clr-namespace:SilverlightApplication73.ViewModels"
    xmlns:i=
       "clr-namespace:System.Windows.Interactivity;assembly=System.Windows.Interactivity"
    xmlns:cmd=
       "clr-namespace:GalaSoft.MvvmLight.Command;assembly=GalaSoft.MvvmLight.Extras.SL4">
    <UserControl.Resources>
        <vm:BooksViewModel x:Key="vmBooksViewModel" />
    </UserControl.Resources>
    <Grid x:Name="LayoutRoot" Background="White"
          DataContext=
           "{Binding Source={StaticResource vmBooksViewModel}}">
        <sdk:DataGrid x:Name="BookDataGrid"
                ItemsSource="{Binding BookCollection}"
                AutoGenerateColumns="True"
                CanUserReorderColumns="False"
                CanUserResizeColumns="False"
                RowBackground="Bisque"
                AlternatingRowBackground="BurlyWood"
                ColumnWidth="90"
                RowHeight="30"
                GridLinesVisibility="Horizontal"
                HeadersVisibility="All"
                HorizontalGridLinesBrush="Yellow">
            <i:Interaction.Triggers>
                <i:EventTrigger EventName="LoadingRow">
                    <cmd:EventToCommand
                        Command="{Binding LoadingRowCommand,
                        Mode=OneWay}"
                     PassEventArgsToCommand="True"
                     MustToggleIsEnabledValue="True" />
                </i:EventTrigger>
            </i:Interaction.Triggers>
        </sdk:DataGrid>
    </Grid>
</UserControl>
```

توضیحات:

برای تعریف کنترل DataGrid بهتر است تا آن‌را از جعبه‌ابزار VS.NET کشیده و بر روی فرم رها کنید تا به صورت خودکار ارجاعات لازم به اسمبلی‌های مورد نیاز و همچنین تعریف فضای نام مرتبط در XAML جاری، ایجاد شوند.

سپس نحوه‌ی تعریف ViewModel را به همراه انتساب آن به DataContext گرید صفحه ملاحظه می‌کنید. DataGrid قرار گرفته بر روی صفحه اطلاعات خود را از طریق Binding و خاصیت ItemsSource دریافت خواهد کرد.

با تنظیم خاصیت AutoGenerateColumns به true ، کنترل DataGrid ستون‌های متناظر با خاصیت‌های کلاس کتاب را به صورت خودکار تولید می‌نماید.

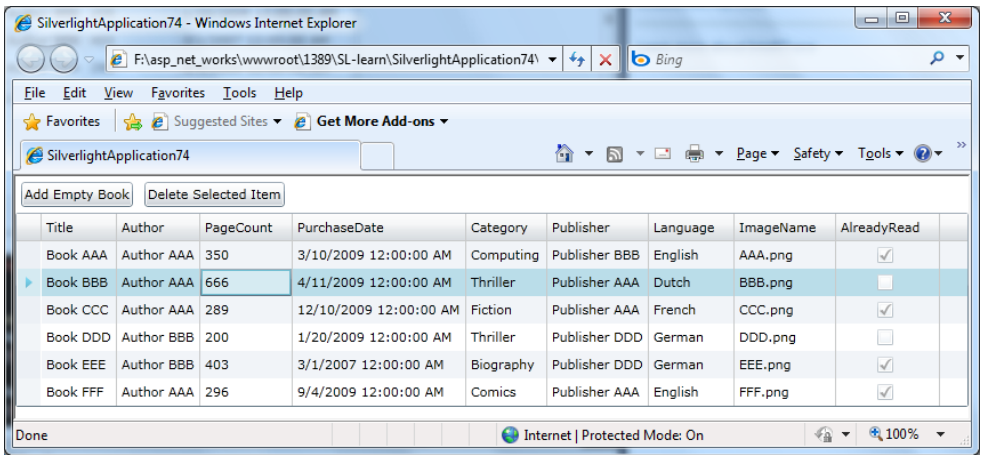
در ادامه نحوه‌ی ثابت کردن ستون‌های این DataGrid با کمک خواص CanUserReorderColumns و CanUserResizeColumns ذکر شده‌اند. رنگ‌های متفاوت سطرهای این کنترل از طریق مقدار دهی خواص RowBackground و AlternatingRowBackground تعریف گردیده‌اند.

مهم‌ترین قسمت این مثال، تعاریف مرتبط با EventToCommand جهت معرفی رخداد LoadingRow یک کنترل DataGrid می‌باشد. به این ترتیب از گره خوردگی کدهای برنامه با View آن جلوگیری خواهیم نمود. در مورد جزئیات بیشتر پیاده سازی EventToCommand در طی فصل آشنایی با MVVM Light toolkit توضیح لازم ارائه شده است.

افزودن، حذف و ویرایش اطلاعات یک DataGrid

کنترل DataGrid امکاناتی را شبیه به نرم افزار Excel جهت افزودن، ویرایش و حذف اطلاعات در اختیار کاربران قرار می‌دهد. در ادامه با توجه به کلاس‌های تعریف شده در مثال قبل و اطلاعات آزمایشی آن، قصد داریم این اعمال را پیاده سازی نمائیم.

برای این منظور همان کلاس‌های Models و ViewModel مثال قبل را به پروژه‌ای جدید اضافه نمائید. بدیهی است ارجاعاتی به اسمبلی‌های کتابخانه MVVM Light toolkit نیز باید افزوده شوند. سپس یک DataGrid را از جعبه ابزار VS.NET کشیده و بر روی View برنامه (همان صفحه‌ای اصلی پروژه) رها کنید تا ارجاعات لازم مورد نیاز آن نیز به صورت خودکار افزوده شوند.



شکل ۳– نمایی از مثال افزودن، ویرایش و حذف اطلاعات یک DataGrid

در این مثال (شکل ۳) قصد داریم به کمک دکمه‌ی Add Empty Book یک سطر جدید را به DataGrid اضافه نمائیم. همچنین به کمک دکمه‌ی Delete Selected Item ، سطر انتخاب شده باید حذف گردد. علاوه بر آن این دو مورد به کمک دکمه‌های Insert و Delete صفحه کلید نیز باید عمل کنند. امکان ویرایش سلول‌ها به صورت خودکار توسط DataGrid فراهم است و تنها باید کدهای مرتبط با آن را به برنامه افزود.

با توجه به اینکه لیست کتاب‌های مقید شده به DataGrid از نوع ObservableCollection است و شیء Book نیز اینترفیس INotifyPropertyChanged را پیاده سازی می‌کند و حالت پیش فرض Binding در DataGrid ، انقیاد دو طرفه است، هرگونه تغییری در این مجموعه بلافاصله در رابط کاربری نمایان شده و برعکس اگر آیتمی ویرایش گردد، بلافاصله اطلاعات آن به ObservableCollection برنامه منعکس خواهد شد.

همانطور که پیشتر نیز عنوان شد کدهای Models و قسمت تعاریف اطلاعات مرتبط با لیست کتاب‌ها در ViewModel ، همانند مثال قبل است و از ذکر مجدد آن‌ها صرفنظر می‌شود. کدهای XAML مرتبط با View برنامه را در ادامه مشاهده خواهید نمود:

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication74.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc=
      "http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400"
    xmlns:sdk=
    "http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk"
    xmlns:vm="clr-namespace:SilverlightApplication74.ViewModels"
    xmlns:i=
       "clr-namespace:System.Windows.Interactivity;assembly=System.Windows.Interactivity"
    xmlns:cmd=
      "clr-namespace:GalaSoft.MvvmLight.Command;assembly=GalaSoft.MvvmLight.Extras.SL4">
    <UserControl.Resources>
        <vm:BooksViewModel x:Key="vmBooksViewModel" />
    </UserControl.Resources>
    <Grid x:Name="LayoutRoot" Background="White"
              DataContext=
                "{Binding Source={StaticResource vmBooksViewModel}}">
        <StackPanel>
            <StackPanel Orientation="Horizontal">
                <Button Content="Add Empty Book"  Margin="5"
                        Command="{Binding AddCommand}"
                        />
                <Button Content="Delete Selected Item"  Margin="5"
                        Command="{Binding DeleteCommand}"
                        CommandParameter="{Binding
                               ElementName=BooksDataGrid,
                               Path=SelectedItem}"
```

```
                        />
            </StackPanel>
          <sdk:DataGrid
            ItemsSource="{Binding BookCollection}"
            HeadersVisibility="All"
            Name="BooksDataGrid"
            AutoGenerateColumns="True">
                <i:Interaction.Triggers>
                    <i:EventTrigger EventName="BeginningEdit">
                        <cmd:EventToCommand
                        Command="{Binding BeginningEditCommand,
                        Mode=OneWay}"
                     PassEventArgsToCommand="True"/>
                    </i:EventTrigger>

                    <i:EventTrigger EventName="CellEditEnded">
                        <cmd:EventToCommand
                        Command="{Binding CellEditEndedCommand,
                        Mode=OneWay}"
                     PassEventArgsToCommand="True"/>
                    </i:EventTrigger>

                    <i:EventTrigger EventName="KeyDown">
                        <cmd:EventToCommand
                        Command="{Binding KeyDownCommand,
                        Mode=OneWay}"
                     PassEventArgsToCommand="True"/>
                    </i:EventTrigger>
                </i:Interaction.Triggers>
            </sdk:DataGrid>
        </StackPanel>
    </Grid>
</UserControl>
```

در این View ابتدا محل دریافت اطلاعات یا همان ViewModel برنامه تعریف شده است. سپس کلیه رخدادهای مورد نظر از طریق Commnads و یا ویژگی EventToCommand کتابخانه‌ی MVVM Light toolkit به ViewModel برنامه منتقل خواهند شد.

از رخداد KeyDown برای تشخیص فشرده شدن دکمه‌های Insert و Delete استفاده خواهیم کرد. از رخدادهای BeginningEdit و CellEditEnded جهت مشخص شدن زمان شروع به ویرایش و خاتمه‌ی آن در یک سلول کمک می‌گیریم.

نحوه‌ی پیاده سازی روال‌های رخدادگردان متناظر با این رویدادها را در کدهای ViewModel برنامه در ادامه ملاحظه خواهید نمود:

BooksViewModel.cs

```
using System;
using System.Collections.ObjectModel;
using System.Windows.Controls;
using System.Windows.Input;
using GalaSoft.MvvmLight.Command;
using SilverlightApplication74.Models;

namespace SilverlightApplication74.ViewModels
{
    public class BooksViewModel
    {
              #region Fields (1)

        bool _cellEditing;

              #endregion Fields

              #region Constructors (1)

        public BooksViewModel()
        {

            AddCommand = new RelayCommand(addCommand);
            DeleteCommand = new RelayCommand<Book>(deleteCommand);
            BeginningEditCommand = new
                 RelayCommand<DataGridBeginningEditEventArgs>(beginningEditCommand);
            CellEditEndedCommand = new
                 RelayCommand<DataGridCellEditEndedEventArgs>(cellEditEndedCommand);
            KeyDownCommand =
             new RelayCommand<KeyEventArgs>(keyDownCommand);

            BookCollection =
                new ObservableCollection<Book>
                {
                  new Book
                  {
                    //همانند تعاریف مثال قبل عمل نمائید
                  }
                };
        }

              #endregion Constructors

              #region Properties (6)
```

```
public RelayCommand AddCommand { get; private set; }

public RelayCommand<DataGridBeginningEditEventArgs>
           BeginningEditCommand { get; private set; }

public ObservableCollection<Book> BookCollection { set; get; }

public RelayCommand<DataGridCellEditEndedEventArgs>
           CellEditEndedCommand { get; private set; }

public RelayCommand<Book> DeleteCommand { get; private set; }

public RelayCommand<KeyEventArgs> KeyDownCommand
{ get; private set; }

       #endregion Properties

       #region Methods (7)

       // Private Methods (7)

private void addCommand()
{
    addEmptyBook();
}

private void addEmptyBook()
{
    var b = new Book();
    BookCollection.Add(b);
}

private void beginningEditCommand(
          DataGridBeginningEditEventArgs obj)
{
    _cellEditing = true;
}

private void cellEditEndedCommand(
          DataGridCellEditEndedEventArgs obj)
{
    _cellEditing = false;
}

private void deleteCommand(Book obj)
{
    removeBook(obj);
}
```

```
        //نحوه‌ی عکس العمل نشان دادن به فشرده شدن کلیدها و یافتن شیء جاری
        private void keyDownCommand(KeyEventArgs e)
        {
            if (e.Key == Key.Delete && !_cellEditing)
            {
                var dataGrid = e.OriginalSource as DataGrid;
                if (dataGrid == null) return;
                var item = dataGrid.SelectedItem as Book;
                if (item == null) return;
                removeBook(item);
            }
            else if (e.Key == Key.Insert && !_cellEditing)
            {
                addEmptyBook();
            }
        }

        private void removeBook(Book obj)
        {
            BookCollection.Remove(obj);
        }

              #endregion Methods
    }
}
```

توضیحات:

همانطور که پیشتر نیز عنوان شد، اساس این مثال استفاده از شیء ObservableCollection می‌باشد که جهت انعکاس آنی تغییرات بکارگرفته شده است. هر آیتمی که به آن اضافه شود بلافاصله در رابط کاربری برنامه منعکس شده و هر سلولی که در DataGrid ویرایش شود با توجه به Binding دو طرفه پیش فرض در DataGrid ، سبب به روز رسانی خودکار منبع داده‌ای نیز می‌گردد.

روال‌های رویدادگردان متناظر با رویدادهای برنامه نیز به کمک توانایی‌های کتابخانه‌ی MVVM Light toolkit ایجاد شده‌اند. به این صورت دیگر نیازی به تعریف این روال‌ها در فایل Code behind مرتبط با View برنامه نیست.

مرتب سازی و گروه بندی اطلاعات DataGrid

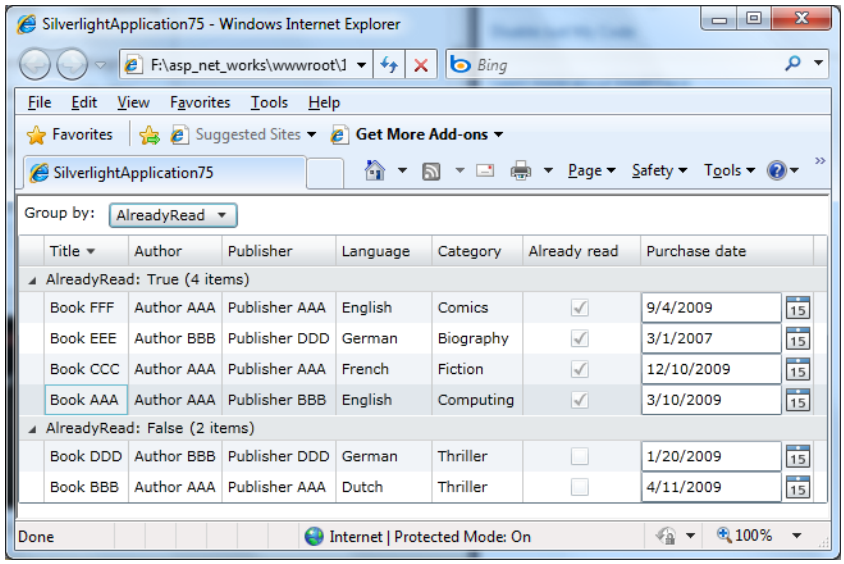
مرتب سازی اطلاعات در یک DataGrid به صورت پیش فرض مهیا است و نیازی به کد نویسی ندارد. برای این منظور کاربر تنها کافی است بر روی سرستونی دلخواه یکبار کلیک نماید تا اطلاعات DataGrid بر اساس آن فیلد انتخابی مرتب شوند.

برای گروه بندی اطلاعات باید بجای استفاده مستقیم از شیء ObservableCollection از شیء PagedCollectionView که در فصل آشنایی با کنترل‌های Silverlight معرفی گردید استفاده نمائیم.

در ادامه قصد داریم اطلاعات لیست کتاب‌های مثال‌های قبلی را با کمک الگوی MVVM و همچنین عدم استفاده از ویژگی AutoGenerateColumns ، به شکلی گروه بندی شده نمایش دهیم.

برای این منظور ابتدا یک پروژه‌ی جدید را گشوده، فایل‌های Models و ViewModel مثال‌های قبل را به آن اضافه نمائید (در حد دریافت لیست کتاب‌ها کفایت می‌کند). سپس دو کنترل DataGrid و DatePicker را از جعبه ابزار VS.NET کشیده و بر روی فرم رها کنید تا ارجاعات لازم به اسمبلی‌های آن‌ها و همچنین فضای نام متناظر آن‌ها افزوده گردد. سپس این دو کنترل را حذف کرده و کدهای View را مطابق کدهای XAML بعد تغییر دهید. علاوه بر آن افزودن ارجاعاتی به اسمبلی‌های کتابخانه‌ی MVVM Light toolkit را نیز فراموش نکنید.

در کنترل DataGrid در صورت عدم استفاده از ویژگی AutoGenerateColumns ، باید نسبت به تعریف دستی کلیه ستون‌های مورد نظر اقدام کرد که در این حالت امکان تعریف کنترل‌های سفارشی مانند یک DatePicker بجای صرفا نمایش یک برچسب ساده یا استفاده از ComboBox برای نمایش لیستی از آیتم‌ها در هر یک از سلول‌های یک ستون، مهیا خواهد بود (شکل ۴).



شکل ۴– نمایی از کنترل‌های سفارشی به همراه گروه بندی اطلاعات در یک DataGrid

در ادامه کدهای ViewModel برنامه را مشاهده می‌نمائید (کلاس‌های مدل برنامه همانند کدهای مثال‌های قبل فصل جاری هستند):

BooksViewModel.cs

```
using System;
using System.Collections.ObjectModel;
```

```
using System.ComponentModel;
using System.Windows.Controls;
using System.Windows.Data;
using GalaSoft.MvvmLight.Command;
using SilverlightApplication75.Models;

namespace SilverlightApplication75.ViewModels
{
    public class BooksViewModel
    {
        private ObservableCollection<Book> bookCollection { set; get; }
        public PagedCollectionView BooksView { set; get; }

        public RelayCommand<SelectionChangedEventArgs>
           SelectionChangedCommand { set; get; }

        public BooksViewModel()
        {
            SelectionChangedCommand =
                new RelayCommand<SelectionChangedEventArgs>(
                    selectionChangedCommand);

            bookCollection =
                new ObservableCollection<Book>
                {
                    //تعاریف لیست کتاب‌های این قسمت نیز مانند قسمت‌های قبل است
                };


            BooksView = new PagedCollectionView(bookCollection);
            BooksView.SortDescriptions.Add(
                new SortDescription("Title",
                ListSortDirection.Descending));
            BooksView.GroupDescriptions.Add(new
                PropertyGroupDescription("Language"));
        }

        private void selectionChangedCommand(
                      SelectionChangedEventArgs e)
        {
            if (e.AddedItems.Count == 0) return;
            var selectedGroup = e.AddedItems[0] as ComboBoxItem;
            if (selectedGroup == null) return;
            BooksView.GroupDescriptions.Clear();
            BooksView.GroupDescriptions.Add(
                new PropertyGroupDescription(
                    selectedGroup.Content.ToString()));
        }
```

```
        }
}
```

توضیحات:

این‌بار بجای معرفی مستقیم شیء ObservableCollection ، یک شیء از نوع PagedCollectionView را بر اساس آن ساخته و در اختیار View برنامه قرار می‌دهیم. در ابتدای ایجاد شیء BooksView، به کمک متد BooksView.SortDescriptions.Add ، نحوه‌ی مرتب سازی پیش فرض را تغییر داده‌ایم و سپس به کمک متد BooksView.GroupDescriptions.Add ، گروه‌بندی پیش فرضی را نیز معرفی نموده‌ایم.

علاوه بر آن رخداد selectionChanged یک ComboBox را نیز در ViewModel برنامه به کمک ویژگی EventToCommand کتابخانه‌ی MVVM Light toolkit مدیریت نموده‌ایم. در این روال رخداد گردان ابتدا متن آیتم انتخابی دریافت شده و سپس کلیه گروه‌های موجود حذف و گروه جدیدی بر اساس انتخاب جدید کاربر ایجاد خواهد شد. اعمال تغییرات PagedCollectionView به UI برنامه از طریق امکانات Binding، آنی است و نیازی به کد نویسی خاصی ندارد.

و در ادامه کدهای XAML متناظر با View برنامه به شرح زیر هستند:

BooksViewModel.cs

```
<UserControl x:Class="SilverlightApplication75.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc=
     "http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400"
    xmlns:sdk=
     "http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk"
    xmlns:vm="clr-namespace:SilverlightApplication75.ViewModels"
    xmlns:i=
     "clr-namespace:System.Windows.Interactivity;assembly=System.Windows.Interactivity"
    xmlns:cmd=
     "clr-namespace:GalaSoft.MvvmLight.Command;assembly=GalaSoft.MvvmLight.Extras.SL4">
    <UserControl.Resources>
        <vm:BooksViewModel x:Key="vmBooksViewModel" />
    </UserControl.Resources>
    <Grid x:Name="LayoutRoot" Background="White"
          DataContext
              ="{Binding Source={StaticResource vmBooksViewModel}}"
          >
        <StackPanel>
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="Group by:" Margin="5" />
                <ComboBox  Margin="5" >
                    <ComboBoxItem Content="Language" />
```

```
                <ComboBoxItem Content="Category" />
                <ComboBoxItem Content="AlreadyRead" />
                <i:Interaction.Triggers>
                    <i:EventTrigger EventName="SelectionChanged">
                        <cmd:EventToCommand
                           Command=
                               "{Binding SelectionChangedCommand,
                                 Mode=OneWay}"
                           PassEventArgsToCommand="True" />
                    </i:EventTrigger>
                </i:Interaction.Triggers>
            </ComboBox>
        </StackPanel>
    <sdk:DataGrid x:Name="CopyBookDataGrid"
            ItemsSource="{Binding BooksView}"
            AutoGenerateColumns="False">
        <sdk:DataGrid.Columns>
            <sdk:DataGridTextColumn x:Name="CopyTitleColumn"
                Binding="{Binding Title}"
                Header="Title">
            </sdk:DataGridTextColumn>
            <sdk:DataGridTextColumn x:Name="CopyAuthorColumn"
                    Binding="{Binding Author}"
                    Header="Author">
            </sdk:DataGridTextColumn>
            <sdk:DataGridTextColumn x:Name="CopyPublisherColumn"
                    Binding="{Binding Publisher}"
                    Header="Publisher">
            </sdk:DataGridTextColumn>
            <sdk:DataGridTextColumn x:Name="CopyLanguageColumn"
                    Binding="{Binding Language}"
                    Header="Language">
            </sdk:DataGridTextColumn>
            <sdk:DataGridTextColumn x:Name="CopyCategoryColumn"
                    Binding="{Binding Category}"
                    Header="Category">
            </sdk:DataGridTextColumn>
            <sdk:DataGridCheckBoxColumn
                    x:Name="CopyAlreadyReadColumn"
                    Binding="{Binding AlreadyRead,
                    Mode=TwoWay}"
                    Header="Already read">
            </sdk:DataGridCheckBoxColumn>
            <sdk:DataGridTemplateColumn Header="Purchase date"
                    SortMemberPath="PurchaseDate"
                    x:Name="CopyPurchaseDateColumn">
                <sdk:DataGridTemplateColumn.CellTemplate>
                    <DataTemplate>
```

```
                                <sdk:DatePicker
                                SelectedDate=
                                "{Binding PurchaseDate}">
                            </sdk:DatePicker>
                        </DataTemplate>
                    </sdk:DataGridTemplateColumn.CellTemplate>
                </sdk:DataGridTemplateColumn>
            </sdk:DataGrid.Columns>
        </sdk:DataGrid>
        </StackPanel>
    </Grid>
</UserControl>
```

توضیحات:

در این مثال با نحوه‌ی تعاریف ستون‌های سفارشی آشنا می‌شویم. خاصیت AutoGenerateColumns به False تنظیم شده است بنابراین نیاز می‌باشد تا تمامی ستون‌های مورد نظر را به صورت دستی تعریف نمائیم. هرچند اینکار اندکی وقت گیر است اما امکان تعریف کنترل‌های سفارشی همانند تعاریف DataGridTemplateColumn فوق مهیا خواهد بود. در این حالت حتما باید SortMemberPath مقدار دهی گردد، در غیر اینصورت DataGrid امکانات مرتب سازی خودکار را برای این ستون که می‌تواند ترکیبی از چند کنترل را در یک سلول خود داشته باشد، درنظر نخواهد گرفت.

در این حالت ابتدا باید گروه ستون‌ها توسط DataGrid.Columns تعریف شوند:

XAML

```
<sdk:DataGrid x:Name="BookDataGrid"
       AutoGenerateColumns="False">
   <sdk:DataGrid.Columns>
   </sdk:DataGrid.Columns>
</sdk:DataGrid>
```

سپس برای نمایش اطلاعات فقط متنی می‌توان از DataGridTextColumn استفاده کرد و جهت ستونی با اطلاعاتی از نوع bool ، از DataGridCheckBoxColumn می‌توان کمک گرفت:

XAML

```
<sdk:DataGridTextColumn x:Name="TitleColumn"
       Binding="{Binding Title}"
       Header="Title">
</sdk:DataGridTextColumn>
```

انعطاف پذیرترین حالت تعریف ستون‌ها به کمک DataGridTemplateColumn است. در اینجا می‌توان با استفاده از یک DataTemplate ، قالب دلخواهی را برای اطلاعات قابل نمایش در یک سلول ستون مورد نظر خود، طراحی کرد:

XAML

```
<sdk:DataGridTemplateColumn x:Name="ImageColumn">
   <sdk:DataGridTemplateColumn.CellTemplate>
      <DataTemplate>
```

```
      ...
     </DataTemplate>
   </sdk:DataGridTemplateColumn.CellTemplate>
</sdk:DataGridTemplateColumn>
```

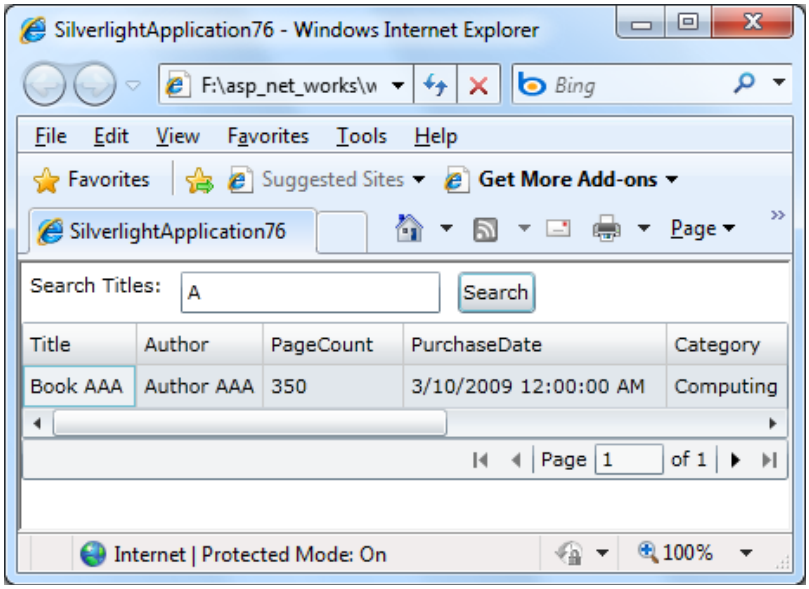
برای مثال می‌توان بجای نام تصویر، یک کنترل Image را در اینجا تعریف نمود و با استفاده از تبدیل کننده‌ها که در طی فصول قبلی کتاب در مورد آن‌ها به تفصیل بحث شد، اطلاعات ImageSource مناسبی را دریافت و نمایش داد؛ برای نمونه :

XAML

```
<sdk:DataGridTemplateColumn x:Name="ImageColumn">
  <sdk:DataGridTemplateColumn.CellTemplate>
    <DataTemplate>
       <Image Source="{Binding ImageName,
          Converter={StaticResource localImageConverter}}"
          Margin="2">
       </Image>
     </DataTemplate>
  </sdk:DataGridTemplateColumn.CellTemplate>
</sdk:DataGridTemplateColumn>
```

جستجو و صفحه بندی اطلاعات در یک DataGrid

قابلیت‌های PagedCollectionView تنها به ارائه‌ی اطلاعاتی به شکل گروه بندی شده خلاصه نمی‌شود. با استفاده از آن می‌توان در اطلاعات یک DataGrid جستجو نمود و همچنین با کمک کنترل DataPager ، نسبت به صفحه بندی اطلاعات نمایش داده شده، اقدام کرد. در ادامه قصد داریم توسط یک مثال کاربردی این موارد را پیاده سازی نمائیم (شکل ۵). مدل برنامه از همان کلاس‌های قبلی تعاریف شیء کتاب تشکیل شده و قسمت تهیه‌ی لیست کتاب‌ها در ViewModel برنامه نیز همانند مثال‌های قبل است. ارجاعاتی نیز به اسمبلی‌های کتابخانه‌ی MVVM Light toolkit نیاز خواهد بود.



شکل ۵– نمایی از اجرای پروژه‌ی جستجو و صفحه بندی اطلاعات یک DataGrid

کدهای متناظر با View این مثال را در ادامه ملاحظه خواهید نمود:

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication76.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc=
      "http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400"
    xmlns:sdk=
      "http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk"
    xmlns:vm="clr-namespace:SilverlightApplication76.ViewModels">
    <UserControl.Resources>
        <vm:BooksViewModel x:Key="vmBooksViewModel" />
    </UserControl.Resources>
    <Grid x:Name="LayoutRoot" Background="White"
          DataContext=
              "{Binding Source={StaticResource vmBooksViewModel}}"
          >
        <StackPanel>
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="Search Titles:" Margin="5" />
                <TextBox Margin="5" Width="150" Name="txtSearch" />
                <Button Content="Search" Margin="5"
                        Command="{Binding SearchCommand}"
                        CommandParameter=
                            "{Binding ElementName=txtSearch,
```

```
                                Path=Text}"
                        />
            </StackPanel>
            <sdk:DataGrid
                ItemsSource="{Binding BooksView}"
                AutoGenerateColumns="True"  />
            <sdk:DataPager
                Name="dataPager1"
                DisplayMode="FirstLastPreviousNext"
                Source="{Binding BooksView}"
                PageSize="3" />
        </StackPanel>
    </Grid>
</UserControl>
```

تنها نکته‌ی جدید View این مثال استفاده از کنترل DataPager است که محدود به DataGrid نیز نمی‌باشد. برای مثال اگر از یک ListBox هم استفاده نمائید می‌توان کنترل DataPager را به آن اعمال نمود؛ البته با این شرط که اطلاعات آیتم‌های آن نیز از طریق یک شیء PagedCollectionView تامین شود. در کنترل DataPager، مقدار خاصیت Source دقیقا مساوی همان خاصیت ItemsSource کنترل DataGrid قرار داده می‌شود. توسط خاصیت PageSize آن مشخص خواهیم نمود که در هر صفحه چند ردیف نمایش داده شود و به کمک خاصیت DisplayMode، چندین حالت نمایشی مختلف را می‌توان برای این کنترل در نظر گرفت (شکل ۶).

PagerDisplayMode value	Visualization
FirstLastNumeric	1 2 3 4
FirstLastPreviousNext	Page 1 of 4
FirstLastPreviousNextNumeric	1 2 3 4
Numeric	1 2 3 4
PreviousNext	Page 1 of 4
PreviousNextNumeric	1 2 3 4

شکل ۶– مقادیر مختلف خاصیت DisplayMode در کنترل DataPager .

در ادامه کدهای ViewModel برنامه ذکر شده‌اند:

BooksViewModel.cs

```
using System;
using System.Collections.ObjectModel;
using System.Windows.Data;
using GalaSoft.MvvmLight.Command;
using SilverlightApplication76.Models;

namespace SilverlightApplication76.ViewModels
{
```

```
    public class BooksViewModel
{
    private ObservableCollection<Book> bookCollection { set; get; }
    public PagedCollectionView BooksView { set; get; }
    public RelayCommand<string> SearchCommand { set; get; }

    public BooksViewModel()
    {
        SearchCommand = new RelayCommand<string>(searchCommand);
        bookCollection =
            new ObservableCollection<Book>
            {
                    //اطلاعات این قسمت نیز همانند مثال‌های قبل است
           };

        BooksView = new PagedCollectionView(bookCollection);
    }

    private string _srchData = string.Empty;
    private void searchCommand(string data)
    {
        _srchData = data;
        BooksView.Filter = null;
        BooksView.Filter = new Predicate<object>(search);
    }

    private bool search(object obj)
    {
        var book = obj as Book;
        bool foundSearchHit = false;
        if (book != null)
        {
            if (book.Title.Contains(_srchData))
                foundSearchHit = true;
        }
        return foundSearchHit;
    }
}
}
```

توضیحات:

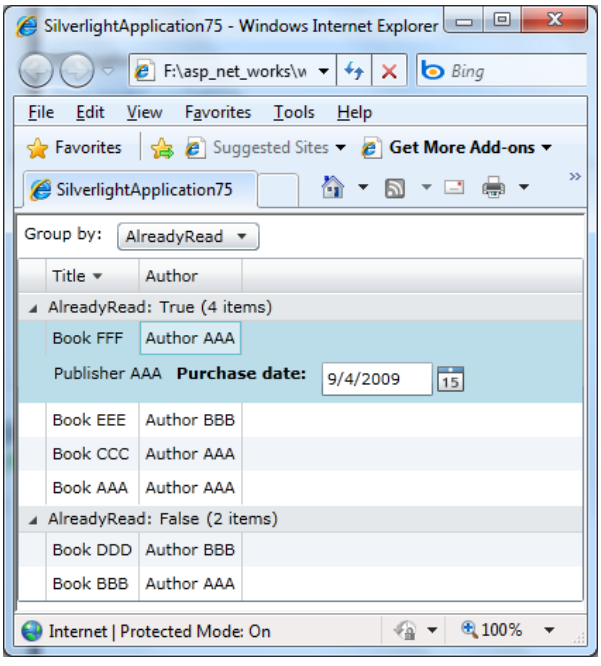
در این ViewModel ، توسط خاصیت BooksView ، اطلاعات لیست کتاب‌ها در اختیار View برنامه قرار می‌گیرد. این لیست از نوع PagedCollectionView می‌باشد.

همچنین روال رخدادگردان جستجوی برنامه نیز در این کلاس مدیریت می‌شود. برای جستجو در یک PagedCollectionView تنها کافی است از امکانات خاصیت Filter آن استفاده نمائیم. به این خاصیت یک شیء

از نوع Predicate نسبت داده شده است که پارامتر اصلی آن منطق جستجوی اطلاعات را مشخص می‌سازد. توسط کلاس Predicate به صورت خودکار یک حلقه از تمامی عناصر PagedCollectionView تشکیل شده و تک تک آیتم‌های آن بررسی خواهند شد. در حین این بررسی هر آیتم ، به متد ذکر شده در آرگومان آن ارسال می‌گردد. اگر این متد true برگرداند به معنای یافت شدن آیتمی مطابق منطق جستجوی برنامه است و برعکس. در نهایت آیتم‌های یافت شده در DataGrid ظاهر خواهند شد.

پیاده سازی سناریوهای Master-Detail در یک DataGrid

اگر تعداد ستون‌های یک گزارش زیاد باشند بهتر است تعدادی از مهم‌ترین ستون‌ها را نمایش داده و مابقی را مخفی کرد. برای این منظور امکان پیاده سازی سناریوهای Master-Detail نیز در کنترل DataGrid پیش بینی شده است. در این حالت، Master هر کدام از ردیف‌های DataGrid خواهند بود و جزئیات بیشتر هر ردیف با کلیک و یا انتخاب آن ردیف نمایش داده خواهند شد (شکل ۷). برای پیاده سازی این قابلیت باید از RowDetailsTemplate استفاده کرد.



شکل ۷– نمایش جزئیات یک ردیف با کلیک بر روی آن به صورت Master-Detail .

همان مثال گروه بندی اطلاعات را که پیشتر به همراه ستون‌های سفارشی مطرح شد، در نظر بگیرید. تنها قصد داریم DataGrid آن‌را اندکی تغییر دهیم:

MainPage.xaml

```
...
            <sdk:DataGrid x:Name="CopyBookDataGrid"
                ItemsSource="{Binding BooksView}"
                AutoGenerateColumns="False">
                <sdk:DataGrid.Columns>
                    <sdk:DataGridTextColumn x:Name="CopyTitleColumn"
                    Binding="{Binding Title}"
                    Header="Title">
                    </sdk:DataGridTextColumn>
                    <sdk:DataGridTextColumn x:Name="CopyAuthorColumn"
                        Binding="{Binding Author}"
                        Header="Author">
                    </sdk:DataGridTextColumn>
                </sdk:DataGrid.Columns>
                <sdk:DataGrid.RowDetailsTemplate>
                    <DataTemplate>
                        <StackPanel Orientation="Horizontal">
                            <TextBlock x:Name="CopyPublisherColumn"
                               Margin="5"
                               Text="{Binding Publisher}" />
                            <TextBlock Text="Purchase date:"
                                FontWeight="Bold"
                                Margin="5"
                                HorizontalAlignment="Left"/>
                            <sdk:DatePicker
                                Margin="5"
                                SelectedDate=
                                "{Binding PurchaseDate}">
                            </sdk:DatePicker>
                        </StackPanel>
                    </DataTemplate>
                </sdk:DataGrid.RowDetailsTemplate>
            </sdk:DataGrid>
...
```

دو ستونی که همواره باید نمایش داده شوند توسط تگ‌های sdk:DataGrid.Columns مشخص شده‌اند. قسمت جزئیات هر ردیف توسط قسمت sdk:DataGrid.RowDetailsTemplate معرفی گردیده و همانطور که مشخص است در آن باید از کنترل‌های معمولی بجای DataGridTextColumn و امثال آن استفاده گردد.

تعیین اعتبار اطلاعات در یک DataGrid

پیشتر در مورد روش‌های مختلف تعیین اعتبار اطلاعات در Silverlight در طی یک فصل، توضیحات لازم ارائه شدند. خبر خوش آن است که کنترل DataGrid اطلاعات تعریف شده توسط روش data annotations (یا همان ویژگی‌های اعتبار سنجی؛ Validation attributes) را به صورت خودکار از تعاریف مدل(های) برنامه خوانده و اعمال خواهد کرد (بدون نیاز به کد نویسی اضافی).

معرفی یک کنترل DataGrid دیگر

اگر کنترل DataGrid مایکروسافت که در طی این فصل مورد بررسی قرار گرفت نیازهای کاری شما را برآورده نمی‌سازد، یک کنترل DataGrid دیگر از شرکت DevExpress به صورت رایگان در اختیار برنامه نویس‌های Silverlight است که از آدرس ذیل قابل دریافت می‌باشد:

http://www.devexpress.com/products/net/controls/silverlight/grid/