بانك اطلاعاتي در Silverlight

مقدمه

اخيرا كه استفاده از سيلورلايت را شروع كرده ام به قابليت هاي جالبي در آن برخورد كرده ام. تركيب امكانات نمايشي WPF و قدرت #C سيلورلايت را بسيار توان مند كرده است. هدف اصلي در اين مقاله استخراج داده ها از بانك اطلاعاتي و استفاده از آن در يك برنامه سيلورلايت مي باشد. اگر شما يك توسعه دهنده ASP.NET هستيد اين مقاله بهترين نقطه شروع براي آشنايي با سيلورلايت و شباهت هاي آن با ASP.NET مي باشد.

گستره بحث

در اين مقاله ما نخستين برنامه تجاري در سيلورلايت را خواهيم ساخت. از آنجايي كه در هر برنامه تجاري بانك اطلاعاتي به نحوي استفاده مي گردد، ما نيز نمايش داده ها (از بانك اطلاعاتي) را با استفاده از برنامه سيلورلايت بررسي مي كنيم. با استفاده از بانك اطلاعاتي آموزشي NorthWind ما تعدادي كلاس LINQ، يك سرويس WCF براي استخراج داده ها از بانك اطلاعاتي، و در نهايت يك ListBox و يك DataGrid همراه با DataTemplate هايي براي نمايش داده ها. خروجي ما يك خروجي ساده چند بخشي مرتبط (master-detail) شامل مشتريان، سفارش هاي آنها و جزئيات سفارش ها خواهد بود.

تذكر به برنامه نويسان ASP.NET

اگر شما برنامه نويس ASP.NET هستيد، به اين نكته مهم توجه داشته باشيد كدهاي #C در پروژه سيلورلايت برخلاف ASP.NET در سمت سرويس گيرنده (client side) اجرا مي گردد. مانند جاوا

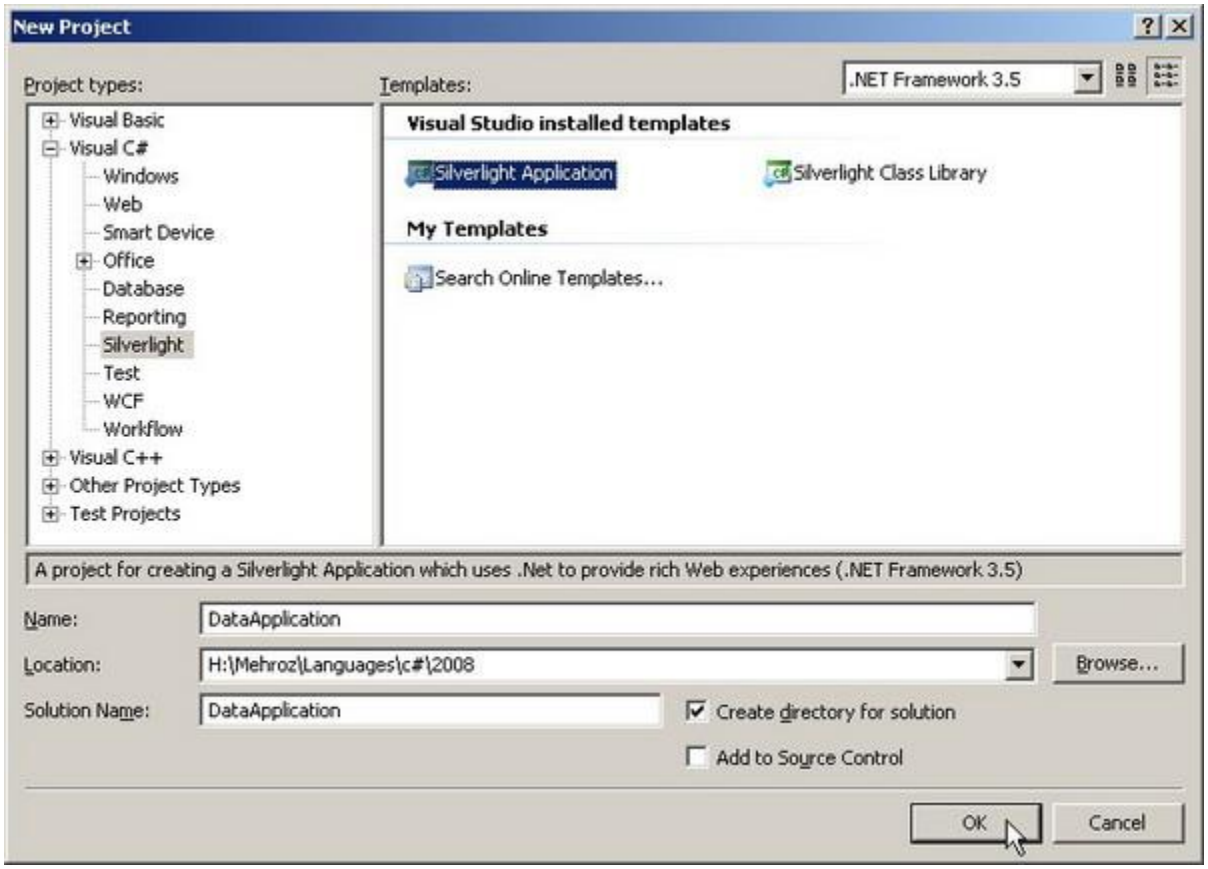
اسكريپت. بنابراين يكي از مزاياي سيلورلايت كنترل كامل شما در سمت سرويس گيرنده است بدون آن كه نيازي به نوشتن و حتي دانستن كد جاوا اسكريپت داشته باشيد.

چرا WCF، مي خواهم به پوشه db دسترسي داشته باشم

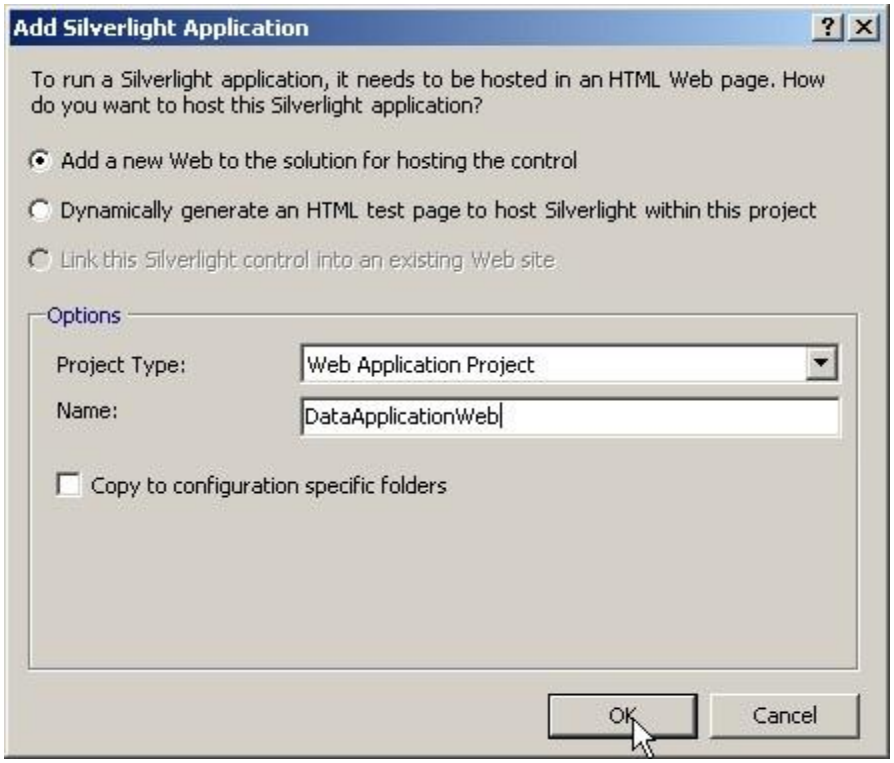
دليل اين امر ساده است، تا زماني كه كد C# در سمت سرويس گيرنده اجرا مي گردد، بانك اطلاعاتي سمت سرور ما به طور مستقيم قابل دسترسي نيست. توجه داشته باشيد كه ما كنترل هايي مانند DataSet يا DataSource در پروژه سيلورلايت نداريم. به علاوه كتابخانه System.Data كه حاوي كلاس هاي مورد استفاده ما است، در دسترس نيست. ولي در عوض جايگزين هاي جالب تري مانند سرويس WCF خواهيم داشت. اين مقاله به شما نشان خواهد داد كه چگونه از سرويس WCF براي دسترسي به داده هاي بانك اطلاعاتي استفاده نماييد.

شروع كار

براي ايجاد يك پروژه سيلورلايت در محيط ويژوال استوديو ابتدا بايد ابزار سيلورلايت را در اين محيط نصب كنيم. دانلود سيلورلايت را مي توانيد از اينجا انجام دهيد. بعد از نصب اين ابزار افزودني دو نوع پروژه جديد با عناوين Silverlight Application و Silverlight Class Library به محيط ويژوال استوديو اضافه خواهند شد. ما ابتدا يك پروژه از نوع Silverlight Application به نام DataApplication را ايجاد مي نماييم.



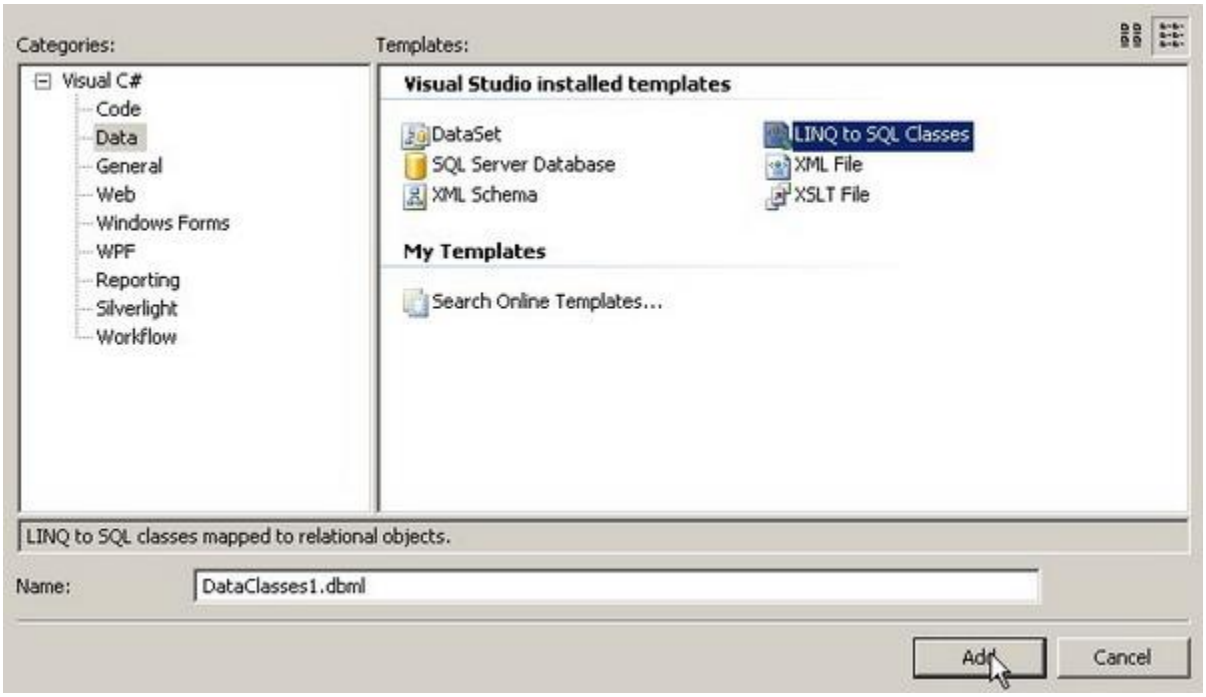
سپس ویژوال استودیو از شما مي پرسد که چگونه مي خواهید پروژه سیلورلایت را ایجاد نمایید؛ ما گزینه ASP.NET Web Application Project انتخاب مي کنیم و کلاس هاي LINQ و سرویس WCF را نیز ایجاد خواهیم کرد.



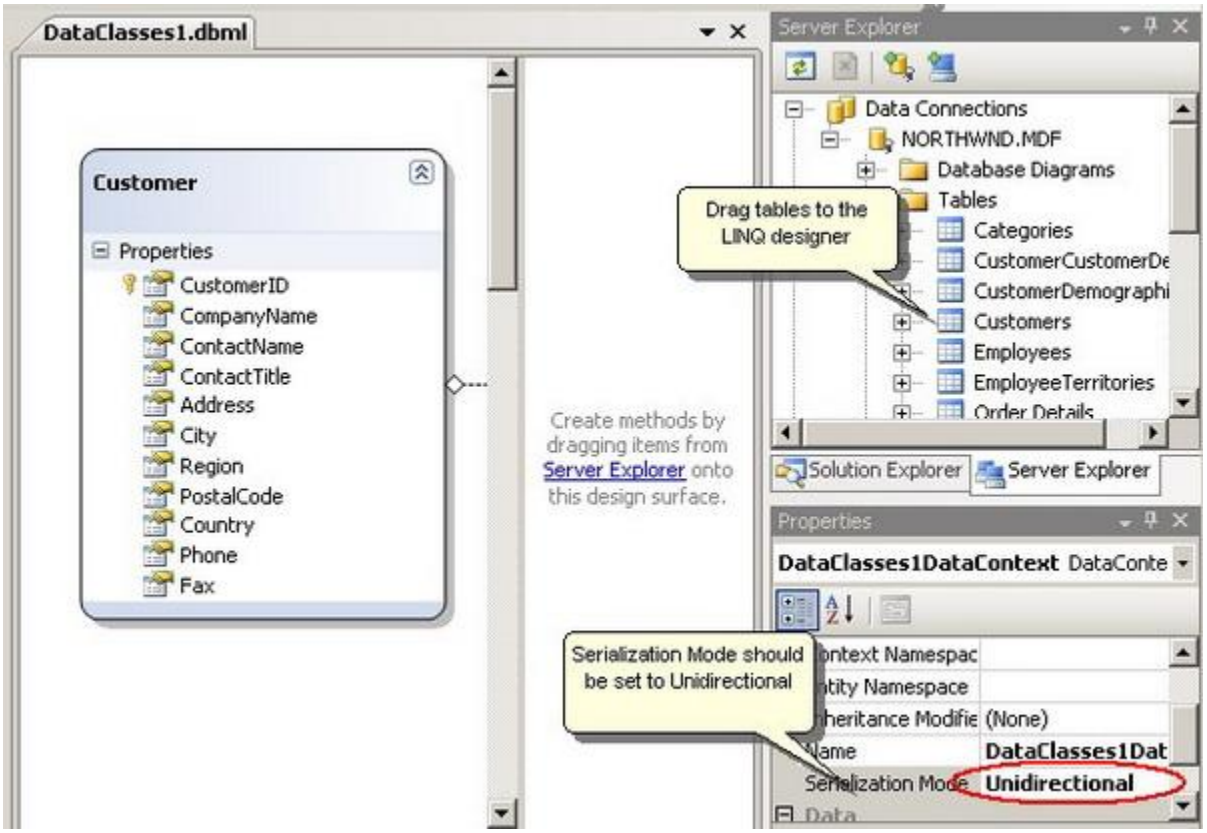
اگر همه چيز درست پيش رفته باشد ما دو پروژه در مجموعه خود خواهيم داشت: DataApplication كه سمت سرويس گيرنده پروژه سيلورلايت و DataApplicationWeb نيز پروژه ASP.NET سمت سرويس دهنده مي باشد.

ايجاد كلاس هاي LINQ

فايل MDF بانك اطلاعاتي Northwind را مي توانيد همراه با كد دانلود كنيد يا اگر SQLExpress را نيز نصب كرده باشيد به عنوان يكي از بانك هاي نمونه از ابتدا به آن متصل است. براي ايجاد كلاس هاي داده LINQ بايد يك Classes LINQ Data جديد را به پروژه ASP.NET خود اضافه كنيم.

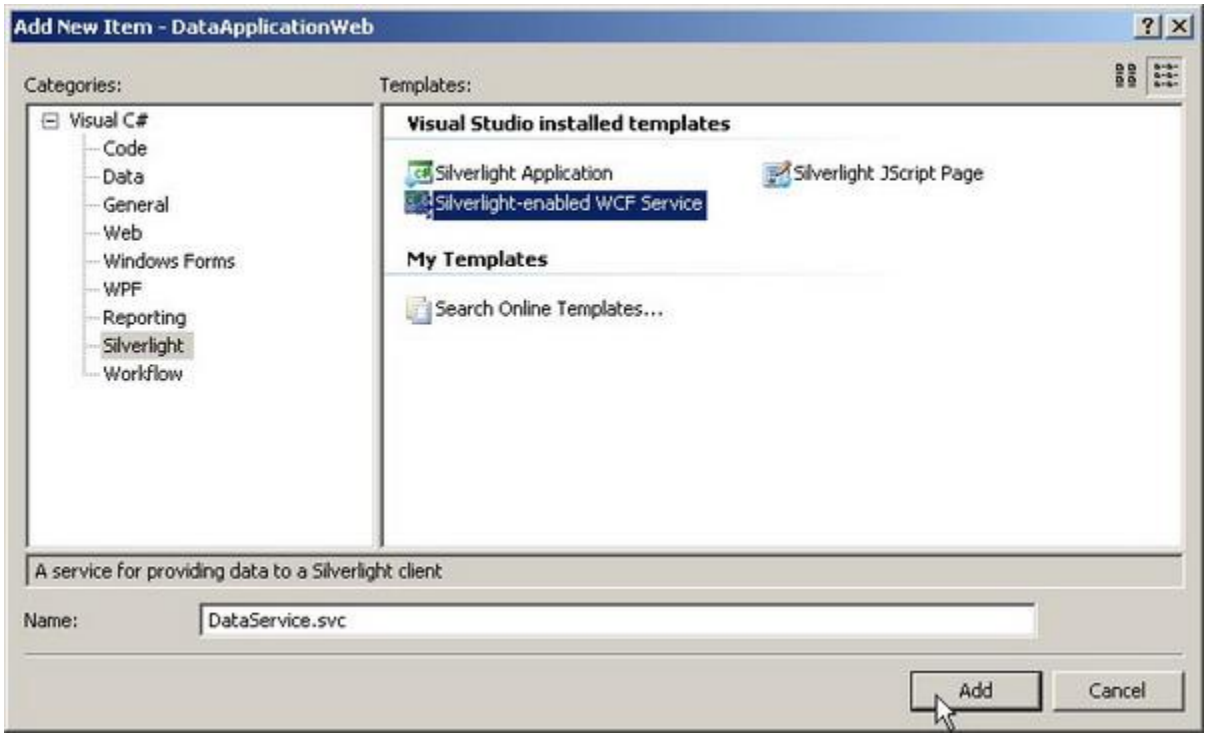


حال با استفاده از Server Explorer يك اتصال جديد به بانك اطلاعاتي Northwind (با استفاده از SQLExpress يا SQL Server) ايجاد مي نماييم و جداول Customer، Orders و OrderDetails را به محيط طراحي LINQ مي كشيم. نكته مهم فعال كردن serialization كلاس هاي توليد شده LINQ به هنگام انتقال اين اشيا به پروژه سيلورلايت است. بدين منظور در يك فضاي خالي در محيط طراحي LINQ كليك كنيد و در پنجره مشخصات، Serialization را به UniDirectional تغيير دهيد.



ایجاد سرویس WCF سیلورلایت

حال در پروژه سمت سرویس دهنده، یك سرویس براي بازیابي داده ها اضافه مي كنيم. در نگارش هاي پيش از بتا۲ ما بايد تغييراتي را در الگوي استاندارد سرويس WCF جهت استفاده در پروژه سيلورلايت اعمال مي كرديم؛ ولي از پس از نگارش مذكور الگوي با نام Silverlight-Enabled WCF service استفاده مي گردد كه تمامي نيازهاي ما را برآورده مي نمايد. ما يك Silverlight-Enabled WCF service را به نام DataService به پروژه ASP.NET اضافه مي نماييم.



ما سه روال (تابع) را براي سرويس WCF بايد بنويسيم؛ يكي كه همه مشتريان را برمي گرداند، يكي سفارش هاي مشتريان را برمي گرداند و در نهايت براي جزئيات يك سفارش. توجه داشته باشيد كه روال هاي مذكور بايد با مشخصه [OperationContract] نشانه گذاري شوند. در اينجا يك پياده سازي سريع از روال هاي مذكور را با استفاده از الفباي LINQ مي بينيد. اين كد ها را در فايل DataService.svc.cs بنويسيد.

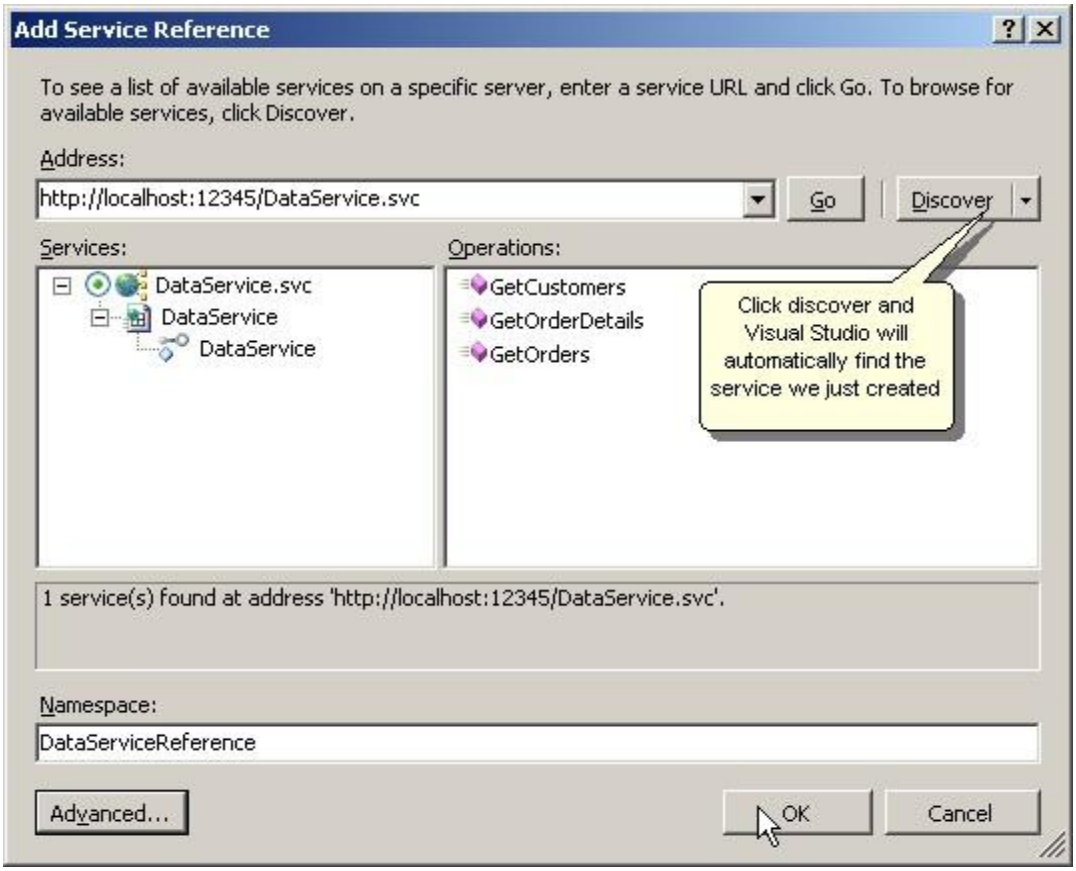
```
[OperationContract]

public List<Customer> GetCustomers()

{

    DataClasses1DataContext datacontext = new DataClasses1DataContext();

    return datacontext.Customers.ToList();
```

```
}


[OperationContract]
public List<Order> GetOrders(string customerID)
{
    DataClasses1DataContext datacontext = new DataClasses1DataContext();
    return (from order in datacontext.Orders
            where order.CustomerID == customerID
            select order).ToList();
}


[OperationContract]
public List<Order_Detail> GetOrderDetails(int orderID)
{
    DataClasses1DataContext datacontext = new DataClasses1DataContext();
    return (from orderdetail in datacontext.Order_Details
            where orderdetail.OrderID == orderID
```

```
        select orderdetail).ToList();

}
```

افزودن يك مرجع سرويس به پروژه سيلورلايت

اين تمام آن كاري بود كه در پروژه سمت سرويس دهنده ASP.NET بايد انجام مي داديم. ما كلاس داده LINQ را براي دريافت داده ها از بانك اطلاعاتي و يك سرويس WCF نيز براي برگرداندن اشيا LINQ ايجاد كرديم. حال نوبت استفاده از اين سرويس در سمت سرويس گيرنده پروژه سيلورلايت است. بدين منظور ما بايد يك مرجع (service reference) در پروژه DataApplication اضافه كنيم. دكمه Discover را در پنجره Add Service Reference مي زنيم تا ويژوال استوديو به طور خودكار سرويس جديد WCF را كه ما ايجاد كرده ايم، پيدا كند.

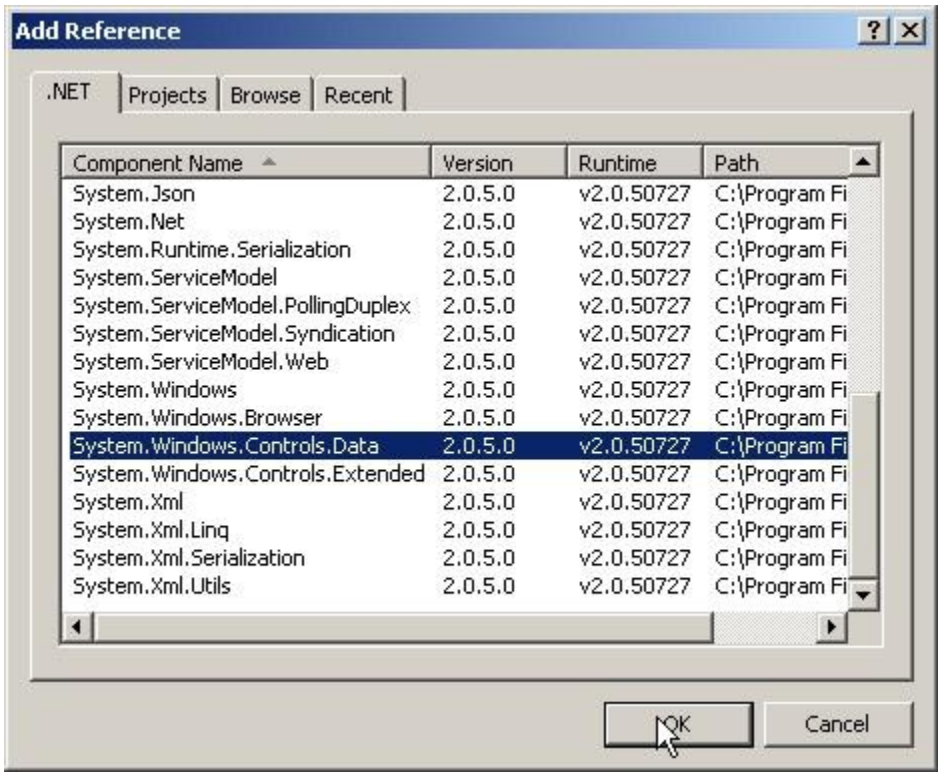


ايجاد واسط كاربري

يك كنترل يا صفحه سيلورلايت از يك فايل XAML براي ذخيره سازي تركيب آن و يك فايل xaml.cs براي كد پس زمينه تشكيل شده است. مشابه صفحه ASP.NET فايل XAML شامل تعريف تركيب صفحه (مانند فايل aspx كه واسط كاربري را در آن تعريف مي كنيم) و فايل xaml.cs كه شامل منطق و مديريت رويدادها (مانند فايل aspx.cs) است. حال با ايجاد چند تركيب ساده براي برنامه مان شروع مي كنيم.

افزودن assembly براي استفاده از كنترل DataGrid

ما داده ها را با استفاده از كنترل DataGrid نشان مي دهيم ولي سيلورلايت به طور پيش فرض مرجع به كنترل DataGrid ندارد و ما بايد آن را اضافه نماييم. اين روش بسيار شبيه به استفاده از كنترل سفارشي (custom control) در ASP.NET است. همان طور كه ما يك مرجع به فايل DLL را به يك پروژه اضافه مي كنيم و يك علامت Register(tag) را در صفحه aspx اضافه مي كنيم، در سيلورلايت هم يك مرجع جديد اضافه مي كنيم (روي گزينه References راست كليك مي كنيم و گزينه new References را انتخاب مي نماييم) و گزينه System.Windows.Controls.Data از فهرست انتخاب مي كنيم. اين همان فايل اسمبلي است كه كنترل DataGrid در آن تعريف شده است.

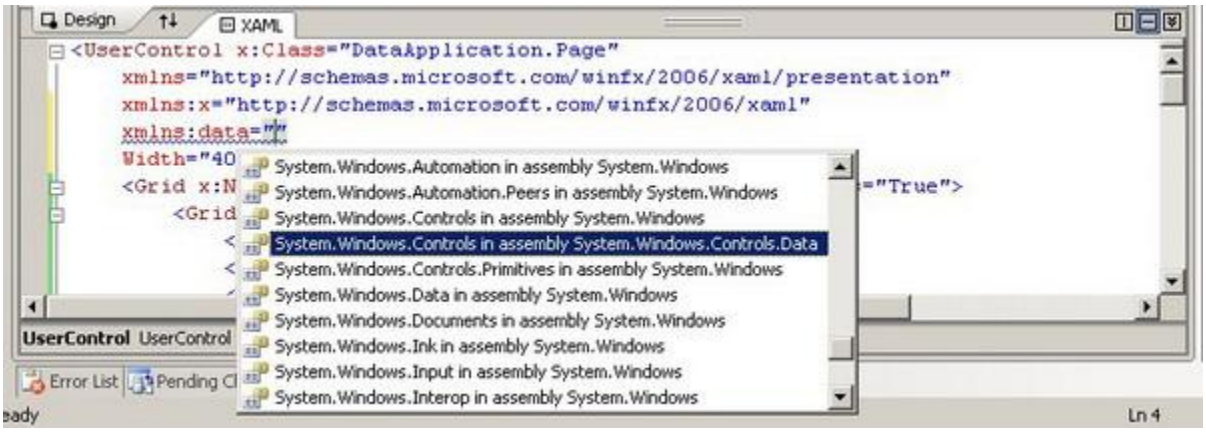


پس از افزدون اين مرجع ما بايد يك كتابخانه (namespace) را به اين اسمبلي در مايل XAML منتسب كنيم. بدين منظور در بخش تعريف كتابخانه فايل Page.Xaml كد زير را اضافه نماييد.

xmlns:data="clr-

namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"

همان طور كه تصوير نيز مي بينيد محيط ويژوال استوديو به ما كمك خواهد كرد:



تركيب واسط كاربر

ابتدا يك طرح كلي را بررسي مي كنيم: ايجاد يك Grid به نام LayoutRoot با سه سطر: اولي براي نمايش عنوان برنامه به عرض ٥٠ (width=50) سومي براي نوار وضعيت (status bar) به عرض ٢٠ (width=20) و وسطي براي نمايش محتواي اصلي با عرض متغير (width=*) كه باقيمانده فضا را در بر مي گيرد. يك عنوان بلوك متني TextBlock را براي سطر اول و يك بلوك متني خالي به نام txtStatus را به سطر آخر LayoutRoot مي افزاييم. در سطر مياني LayoutRoot (كه به عنوان نگهدارنده محتواي اصلي نام برديم) يك كنترل Grid ديگر به نام ContentRoot با دو ستون و دو سطر مي افزاييم؛ ستون سمت چپ به عرض ٢٠٠ و ستون سمت راست بقيه فضا را در بر مي گيرد. سطر ها بايد به نسبت ٦٠% و ٤٠% تقسيم شوند.. در ستون سمت چپ ContentRoot يك جعبه ليست (ListBox) اضافه مي كنيم كه هر دو سطر را پوشش دهد. در ستون سمت راست يك DataGrid براي سفارش هاي مشتري در سطر اول و يك DataGrid ديگر براي جزئيات سفارش در سطر دوم اضافه مي نماييم. اين نگارش XAML براي فايل page.xaml است.

```
<UserControl

  x:Class="DataApplication.Page"

xmlns:my="clr-

namespace:System.Windows.Controls;assembly=System.Windows.Controls.Extended"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

  xmlns:data="clr-

namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"

Width="Auto" Height="Auto">

<Grid x:Name="LayoutRoot" Background="White">

<Grid.RowDefinitions>

<RowDefinition Height="55" x:Name="HeaderRow" />

      <RowDefinition Height="*" x:Name="ContentRow"/>

<RowDefinition Height="20" x:Name="FooterRow"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="*" />

</Grid.ColumnDefinitions>
```

```
        <!-- Heading -->
<TextBlock x:Name="txtHeader" Grid.Row="0"
                FontSize="20" Margin="5,5" Foreground="Blue"
Text="My First Data Application in Silverlight"></TextBlock>
        <!-- A textblock in the footer to be used as an Status bar -->
<TextBlock x:Name="txtStatus" Grid.Row="2"
              FontSize="10" Margin="5,0" Foreground="Red">
</TextBlock>
        <!-- Content Holder -->
<Grid x:Name="ContentGrid" Grid.Row="1" Margin="5">
<Grid.RowDefinitions>
<RowDefinition Height=".6*" />
<RowDefinition Height=".4*" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="200" />
<ColumnDefinition Width="*" />
```

```
</Grid.ColumnDefinitions>

        <!-- Listbox for displaying customers -->

<ListBox x:Name="lstCustomers" Grid.Column="0" Grid.RowSpan="2"

            DisplayMemberPath="ContactName"

Loaded="lstCustomers_Loaded"

SelectionChanged="lstCustomers_SelectionChanged">

</ListBox>

<!-- DataGrid for displaying orders of a customer

          (with autogenerated columns) -->

<data:DataGrid x:Name="dgOrders" Grid.Row="0" Grid.Column="1"

                    AutoGenerateColumns="True"

SelectionChanged="dgOrders_SelectionChanged">

      </data:DataGrid>

<!-- DataGrid for displaying orderdetais for an order -->

<data:DataGrid x:Name="dgOrderDetails" Grid.Row="1" Grid.Column="1"

                    AutoGenerateColumns="True"

   AutoGeneratingColumn="dgOrderDetails_AutoGeneratingColumn">
```

```
</data:DataGrid>

    </Grid>

  </Grid>

</UserControl>
```

توضيح بيشتري در مورد WPF در اين مقاله نخواهم داد چرا كه مقالات متعددي از جمله اين لينك در اين مورد توضيح مي دهند. ما يكي مرور سريع بر روي ListBox و DataGrid خواهيم داشت.

كنترل ListBox

جعبه ليست با نام lstCustomers براي نمايش فهرست مشتريان در بانك اطلاعاتي استفاده خواهد شد. ما اين ListBox را داخل رويداد Loaded (كه ثبت كرده ايم) متصل خواهيم نمود. توجه داشته باشيد كه اگر يك ListBox به يك منبع شيء (object source) متصل گردد مقدار object.ToString() را براي هر كدام از عناصر مجموعه خود نمايش خواهد داد. اگر ما بخواهيم مقادير ديگري (مانند يك عضو داده رشته اي از يك شيء) را نمايش دهيم سه راه خواهيم داشت:

روال object.ToString() را براي هر كدام از عناصر مجموعه اش بازنويسي (Override) كنيم.

الگوي داده (Data Template) تعريف كنيم. (اين آسان ترين روش است و ما به هنگام تعريف ستون هاي DataGrid به طور خلاصه الگوي داده را بررسي خواهيم كرد ولي در حال حاضر از اين مورد صرف نظر مي كنيم.

يك مشخصه (property) براي استفاده در مشخصه ي DisplayMemberPath جعبه ليست تعريف كنيم. (كه ما الآن از اين روش استفاده مي كنيم)

زماني كه بخواهيم كه جعبه ليست، مشخصه ي ContactName از شيء Customer را كه به آن متصل شده است- نمايش دهد، ما دستور "DispalyMemberPath="ContactName را استفاده مي كنيم. در عين حال ما رويداد SelectionChanged را ثبت كرده ايم كه در فايل كد پس زمينه (-code

behind) از آن براي به روز رساني DataGrid با محتواي سفارش مشتري انتخاب شده استفاده مي نماييم.

کنترل هاي DataGrid

فعلا ما کار خاصي با کنترل هاي DataGrid انجام نمي دهيم. صرفا آن ها را طوري تنظيم کرده ايم که به صورت خودکار ستون هاي خود را به هنگام اتصال به داده ها ايجاد نمايند. توجه داشته باشيد که ما رويداد AutoGeneratingColumns را در dgOrderDetails ثبت کرده ايم. اين تمرين ساده براي ترکيب ستون هاي ايجاد شده خودکار براي زماني که ما مي خواهيم ستون هاي عير ضروري را حذف کنيم، مفيد مي باشد. در همين مقاله ما نحوه تعريف دستي ستون ها را بررسي خواهيم کرد ولي فعلا از همين حالت ساده استفاده مي کنيم.

نوشتن کد

پر کردن ListBox

راي نمايش اطلاعات تمام مشتريان در جعبه ليست lstCustomers ما از رويداد Loaded از کنترل ListBox استفاده مي نماييم. توجه داشته باشيد که تمامي فراخواني سرويس ها در سيلورلايت بايد ناهمزمان باشند از اين رو ما تابع بازگشت به عقب را براي زماني که داده هاي ورودي را به ListBox متصل مي کنيم، ثبت کرده ايم. توجه داشته باشيد که ما از جعبه متن txtStatus استفاده مي کنيم. (به خاطر بياوريد که در سطر آخر LayoutGrid براي اطلاع دادن عمليات به روز رساني به کاربر، اين کنترل را اضافه کرديم)

```
private void lstCustomers_Loaded(object sender, RoutedEventArgs e)

{

    DataServiceClient svc = new DataServiceClient();

    this.txtStatus.Text = "Loading customers...";
```

```
    svc.GetCustomersCompleted += new
    EventHandler<getcustomerscompletedeventargs>(svc_GetCustomersCompleted);
    svc.GetCustomersAsync();
}

void svc_GetCustomersCompleted(object sender, GetCustomersCompletedEventArgs e)
{
     if (e.Error == null)
     {
        this.lstCustomers.ItemsSource = e.Result;
        this.txtStatus.Text = string.Empty;
     }
     else
     {
         this.txtStatus.Text =
             "Error occured while loading customers from database";
     }
```

```
}
```

نمايش سفارش هاي يك مشتري

حال كد مربوط به نمايش سفارش هاي يك مشتري را به هنگام انتخاب آن مشتري در جعبه ليست، مي نويسيم. در رويداد SelectionChanged ما سرويس WCF را فراخواني مي كنيم و پس از بازيابي داده ها، آن ها را به dgOrders متصل مي نماييم.

```
private void lstCustomers_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

    Customer selectedCustomer = this.lstCustomers.SelectedItem as Customer;

    if (selectedCustomer != null)

    {

        DataServiceClient svc = new DataServiceClient();

        this.txtStatus.Text = "Loading orders...";

        svc.GetOrdersCompleted +=

            new EventHandler<GetOrdersCompletedEventArgs>(svc_GetOrdersCompleted);

       svc.GetOrdersAsync(selectedCustomer.CustomerID);

    }

}
```

```
void svc_GetOrdersCompleted(object sender, GetOrdersCompletedEventArgs e)

{

    if (e.Error == null)

    {

       this.dgOrders.ItemsSource = e.Result;

       this.txtStatus.Text = string.Empty;

     }

    else

    {

       this.txtStatus.Text =

       "Error occured while loading orders from database";

     }

}
```

نمايش جزئيات سفارش زماني كه يك سفارش انتخاب مي شود

مشابه رويداد SelectionChanged در جعبه ليست، كد زير را به رويداد SelectionChanged از كنترل dgOrders مي افزاييم. براي فشردگي كار ما از روال هاي بدون نام استفاده مي نماييم.

```
private void dgOrders_SelectionChanged(object sender, EventArgs e)
```

```
{

    Order selectedOrder = this.dgOrders.SelectedItem as Order;

    if (selectedOrder != null)

    {

        DataServiceClient svc = new DataServiceClient();

        this.txtStatus.Text = "Loading order details...";

        svc.GetOrderDetailsCompleted +=

          delegate(object eventSender, GetOrderDetailsCompletedEventArgs eventArgs)

         {

            if (eventArgs.Error == null)

            {

                this.dgOrderDetails.ItemsSource = eventArgs.Result;

                this.txtStatus.Text = string.Empty;

            }

           else

           {

                this.txtStatus.Text =
```

```
"Error occured while loading order details from database";
            }
        };
        svc.GetOrderDetailsAsync(selectedOrder.OrderID);
    }
}
```

حذف بعضي ستون هاي ايجاد شده خودكار از dgOrderDetails

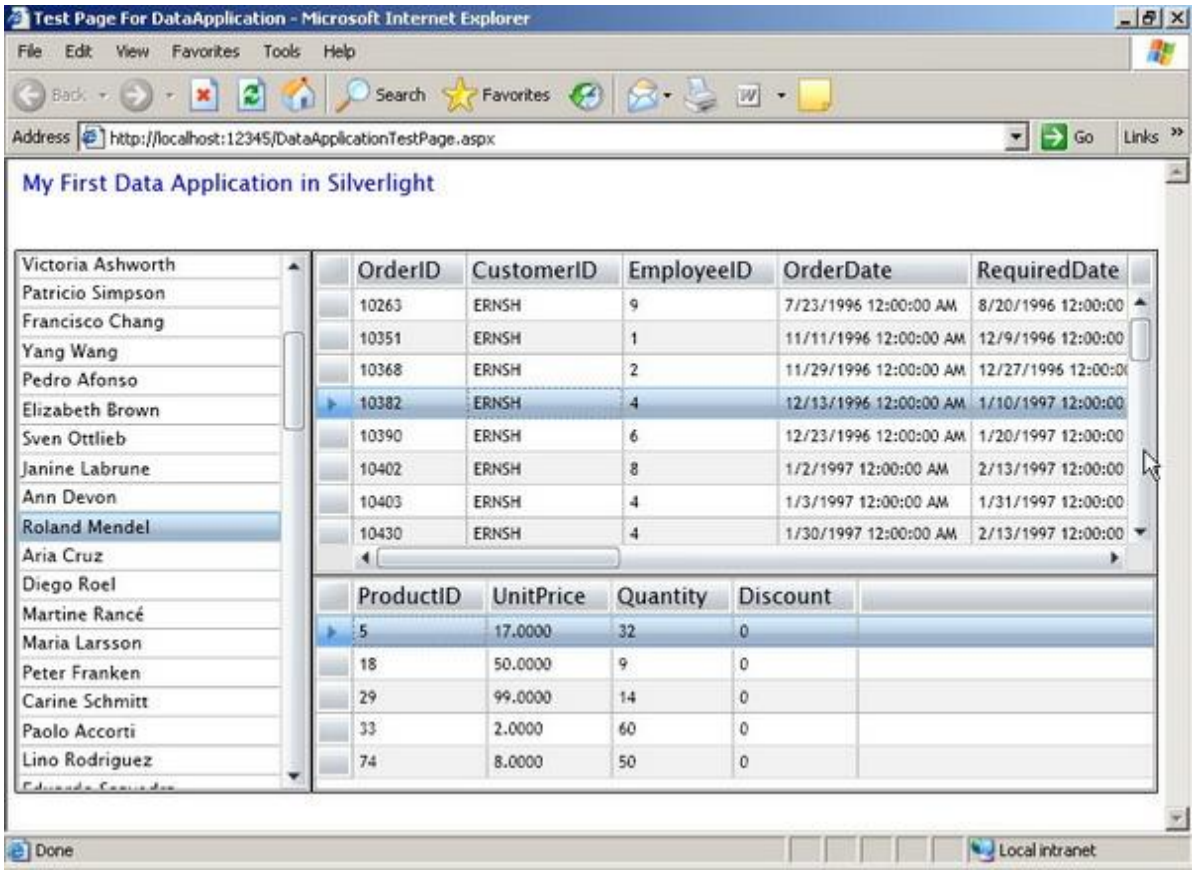
توجه كنيد كه در XAML ما مشخصه AutoGenerateColumns را براي كنترل هاي DataGrid فعال كرديم. ليكن براي نمايش ندادن ستون OrderID در dgOrderDetails كد زير را در رويداد AutoGeneratingColumns مي نويسيم.

```
private void dgOrderDetails_AutoGeneratingColumn (object sender,
        DataGridAutoGeneratingColumnEventArgs e)
{
        if (e.Column.Header.ToString() == "OrderID")
            e.Column.Visibility = Visibility.Collapsed;
}}
```

اتمام مرحله، اجراي پروژه

هم اكنون برنامه ما آماده اجرا است. آن را اجرا كنيد و از فهرست مشتريان يكي به دلخواه انتخاب

کرده، سفارش هاي آن را ببينيد، با كليك كردن در بالاي ستون هاي DataGrid جدول جدول را مرتب نماييد، اندازه ستون ها را تغيير دهيد، اندازه پنجره مرورگر را تغيير دهيد و ... . حالا نوبت آن است كه ستون ها و الگوهاي كنترل DataGrid را مرور كنيم.



تعريف ستون ها

تعريف ستون هاي در كنترل DataGrid سيلورلايت مشابه ASP.NET صورت مي گيرد. در كنترل DataGrid تعريف ستون را به سه روش مي توانيم انجام دهيم:

DataGridTextBoxColumn : اين نوع ستون داده ها را در بلوك متني (TextBlock) نمايش مي دهد و يك جعبه متن (TextBox) را براي ويرايش داده ها استفاده مي كند. شما بايد معين كنيد كه كدام مشخصه از شيء data-bound بايد نمايش داده شود. بدين منظور از DisplayMemberPath استفاده مي كنيد.

DataGridCheckBoxColumn : اين نوع ستون يك CheckBox فقط خواندني را براي نمايش يك مقدار

منطقي (درست/نادرست) يا مقدار منطقي با امكان خالي بودن مقدار، استفاده مي نمايد. و يك CheckBox عادي نيز براي ويرايش مقدار استفاده مي گردد.

DataGridTemplateColumn : اين نوع ستون بسيار قدرتمند بوده، به شما امكان مي دهد تا الگوي داده تعريف كنيد و كنترل هاي مورد نظر خودتان را مشابه TemplateColumn در ASP.NET استفاده نماييد. براي كسب اطلاعات بيشتر در مورد الگوي داده اينجا را كليك كنيد. در ضمن در ويلاگ اسكات موريس هم اطلاعات جالبي در مورد انواع ستون بندي خواهيد ديد.

حال ما از اين امكانات در برنامه خودمان استفاده مي كنيم. براي سادگي كار ما صرفا چهار ستون تعريف مي كنيم: ما يك DataGridTextBoxColumn براي OrderID و EmployeeID تعريف مي كنيم. يك DataGridTemplateColumn براي OrderDate همراه با TextBlock در CellTemplate آن و يك كنترل DatePicker در بخش CellEditingTemplate . در نهايت ما يك الگوي ستون بندي (TemplateColumn) ديگر براي Frieght تعريف خواهيم كرد. ليكن در حال حاضر ما دو كنترل در CellEditingTemplate آن تعريف مي نماييم: يك كنترل Slider براي افزايش/كاهش مقدار فيلد Frieght و يك كنترل TextBlock براي نشان دادن مقدار فعلي Slider . هر دو اين كنترل ها در StackPanel افقي قرار مي گيرند كه ما فقط يك مورد را مي توانيم در DataTemplate تعريف كنيم. اين كد در بخش dgOrders جايگزين مي گردد:

```
<!-- DataGrid for displaying orders of a customer -->

<data:DataGrid x:Name="dgOrders" Grid.Row="0" Grid.Column="1"

        AutoGenerateColumns="False"

SelectionChanged="dgOrders_SelectionChanged">

<data:DataGrid.Columns>

<!-- OrderID text column -->
```

```
<data:DataGridTextColumn Header="Order ID"

                         DisplayMemberBinding="{Binding OrderID}">

</data:DataGridTextColumn>



 <!-- EmployeeID text column -->

<data:DataGridTextColumn Header="Employee ID"

                         DisplayMemberBinding="{Binding EmployeeID}">

</data:DataGridTextColumn>



<!-- OrderDate template column -->

<data:DataGridTemplateColumn Header="Order Date" Width="150">

<data:DataGridTemplateColumn.CellTemplate>

<DataTemplate>

<TextBlock Text="{Binding OrderDate}"></TextBlock>

</DataTemplate>

</data:DataGridTemplateColumn.CellTemplate>

<data:DataGridTemplateColumn.CellEditingTemplate>
```

```
<DataTemplate>

<my:DatePicker SelectedDate="{Binding OrderDate, Mode=TwoWay}">

</my:DatePicker>

</DataTemplate>

</data:DataGridTemplateColumn.CellEditingTemplate>

</data:DataGridTemplateColumn>



<!-- Freight template column -->

<data:DataGridTemplateColumn Header="Freight" Width="150">

<data:DataGridTemplateColumn.CellTemplate>

<DataTemplate>

<TextBlock Text="{Binding Freight}"></TextBlock>

</DataTemplate>

</data:DataGridTemplateColumn.CellTemplate>

<data:DataGridTemplateColumn.CellEditingTemplate>

<DataTemplate>

<StackPanel Orientation="Horizontal">
```

```
<TextBlock Text="{Binding Freight}" Width="50"></TextBlock>

<Slider Value="{Binding Freight, Mode=TwoWay}" Width="100"

Minimum="0" Maximum="500" ></Slider>

</StackPanel>

</DataTemplate>

</data:DataGridTemplateColumn.CellEditingTemplate>

</data:DataGridTemplateColumn>

</data:DataGrid.Columns>

</data:DataGrid>
```

به روش مشابه ما مي توانيم الگو هاي داده را براي جعبه ليست مان هم استفاده كنيم. فرض كنيد در مدل خودمان يك PictureProperty داريم كه تصوير (BitmapImage) مشتري (Customer) را برمي گرداند. براي نمايش تصوير در جعبه ليست بايد از كد زير استفاده نماييم:

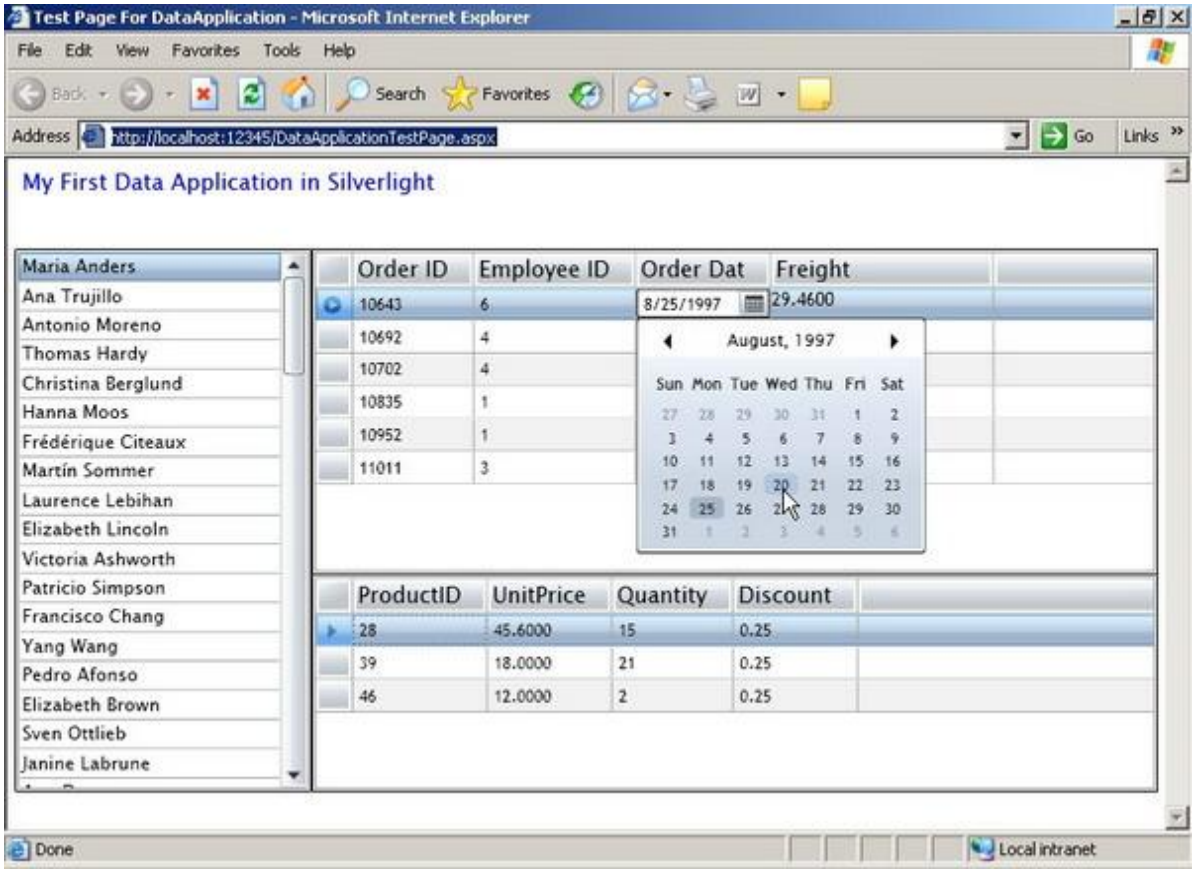
```
<ListBox x:Name="lstCustomer">

    <ListBox.ItemTemplate>

        <DataTemplate>

            <StackPanel Orientation="Vertical">

                <TextBlock Text="{Binding NameProperty}"></TextBlock>

                <Image Source="{Binding PictureProperty}"></Image>
```

```
            </StackPanel>

        </DataTemplate>

    </ListBox.ItemTemplate>

</ListBox>
```

توجه كنيد كه كد بالا صرفا يك مثال است. تا زماني كه ما در اشياي خود تصوير نداشته باشيم نمي توانيم از اين كد استفاده كنيم.

اجراي بعدي

دوباره برنامه را اجرا نماييد. روي يك OrderDate (تاريخ سفارش) دابل كليك كنيد و خواهيد ديد كه كنترل DatePicker باز مي شود. كنترل slider را براي تغيير مقدار Freight (كرايه حمل) استفاده كنيد. اين بار ما فايل cs را تغيير نداديم بلكه همه تغييراتي كه اعمال كرديم در بخش تركيب (layout) صورت مي گيرد.



ذخيره داده ها در بانك اطلاعاتي

اگر شما در مورد چگونگي ارتباط بين سرويس گيرنده و سرويس دهنده از طريق سرويس WCF مطالعه كنيد به راحتي مي توانيد داده هاي خود را در بانك اطلاعاتي بنويسيد. كافي است توابع را داخل سرويس ايجاد كنيد و در برنامه سيلورلايت فراخواني نماييد. توجه كنيد كه اتصالات داده اي (data binding) در اين مقاله به دو روش تعريف شده اند. تغيير مقدار در داخل DataGrid مقدار متناظر در عنصر DataContext را تغيير مي دهد. كافي است كه ما از طريق سرويس WCF اين تغييرات را به پروژه ASP.NET منتقل كنيم و در آنجا براي به روز رساني بانك اطلاعاتي، آن ها را دريافت كنيم. راني ساؤرنمن اين كار را به صورت ويدئو نمايش داده است كه از اينجا مي توانيد دريافت نماييد. او با استفاده از روشي شبيه به DataSet داده هاي اصلي و تغيير يافته را نگه مي دارد و فقط ركوردهاي لازم را با استفاده از يك سري كلاس هاي كمكي به سرور ارسال مي نمايد.

نتيجه

اين مقاله نشان داد كه چطور به سادگي مي توانيم به طور پايه برنامه بانك اطلاعاتي ايجاد نماييم. اكنون آن چه را كه آموختيم جمع بندي مي كنيم: ما لايه دسترسي به داده (DAL=Data Access Layer) را با استفاده از LINQ ايجاد و آن را همراه با يك سرويس WCF در سمت سرويس گيرنده پروژه ASP.NET استفاده كرديم. داده ها را با استفاده از برنامه سيلورلايت سمت سرويس گيرنده بازيابي كرديم و در نهايت چند الگوي داده را براي كنترل بيشتر در نمايش داده ها استفاده نموديم. اميدوارم اين مقاله براي شما انگيزه ي نوشتن برنامه با استفاده از سيلورلايت را ايجاد نمايد.