Utilisation de WCF Data Services 1.5 avec Silverlight

par Benjamin Roux (Retour aux articles)

Date de publication : 02/04/2010

Dernière mise à jour :

Cet article vous présentera comment utiliser Silverlight et WCF Data Services 1.5.

1 - Introduction

Dans cet article, nous allons voir comment utiliser WCF Data Services 1.5 CTP3 avec Silverlight 3.

Premièrement, pourquoi utiliser Data Services 1.5 ? Tout simplement parce que l'intégration avec Silverlight est grandement améliorée (INotifyPropertyChanged et ObservableCollection, Two-way binding.).

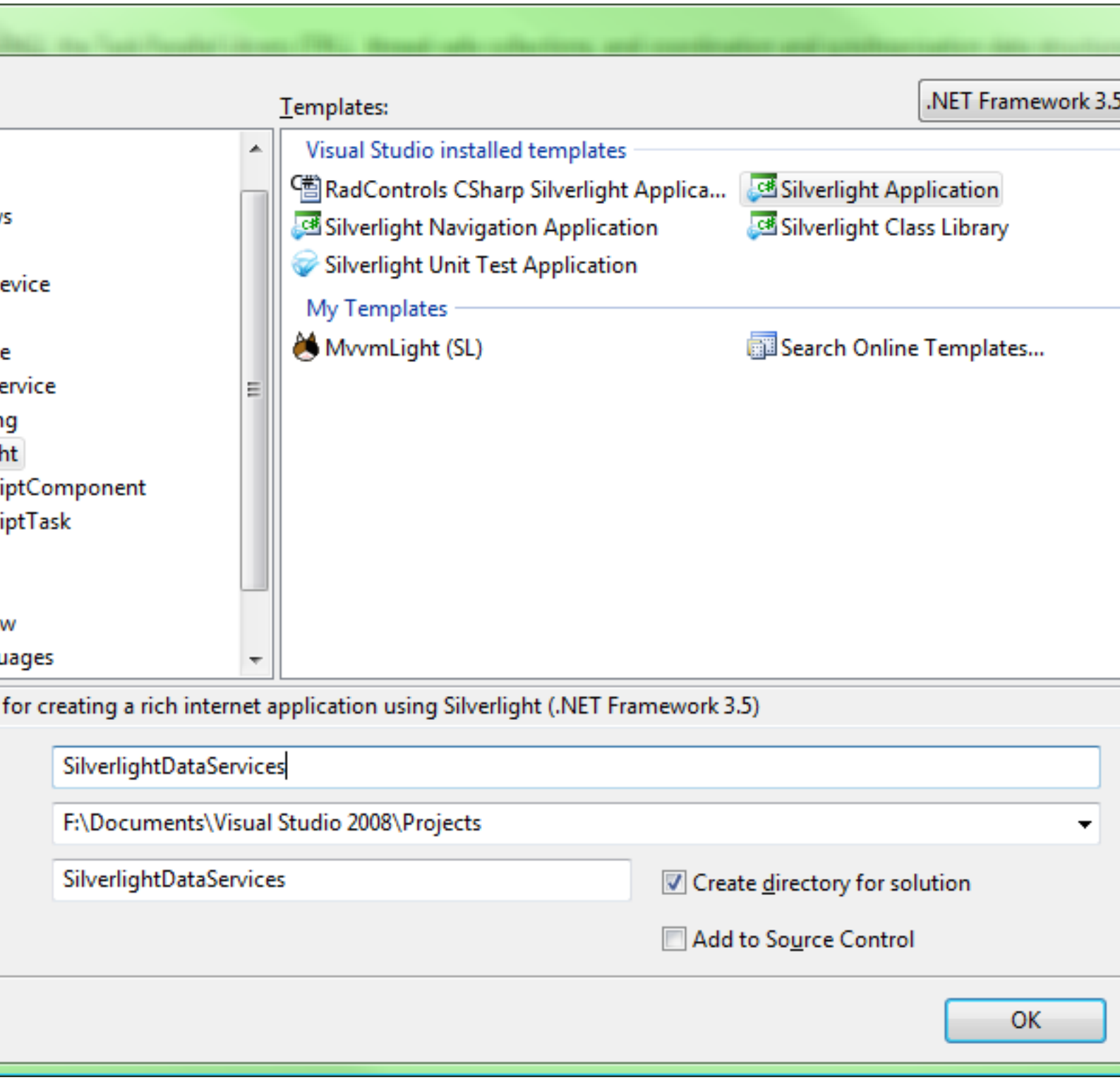
Pré-requis :

- Visual Studio 2008
- **Silverlight 3 Tools**
- **WCF Data Services 1.5 CTP2**

Veillez à bien installer les Silverlight Tools avant Data Services !

2 - Création du projet

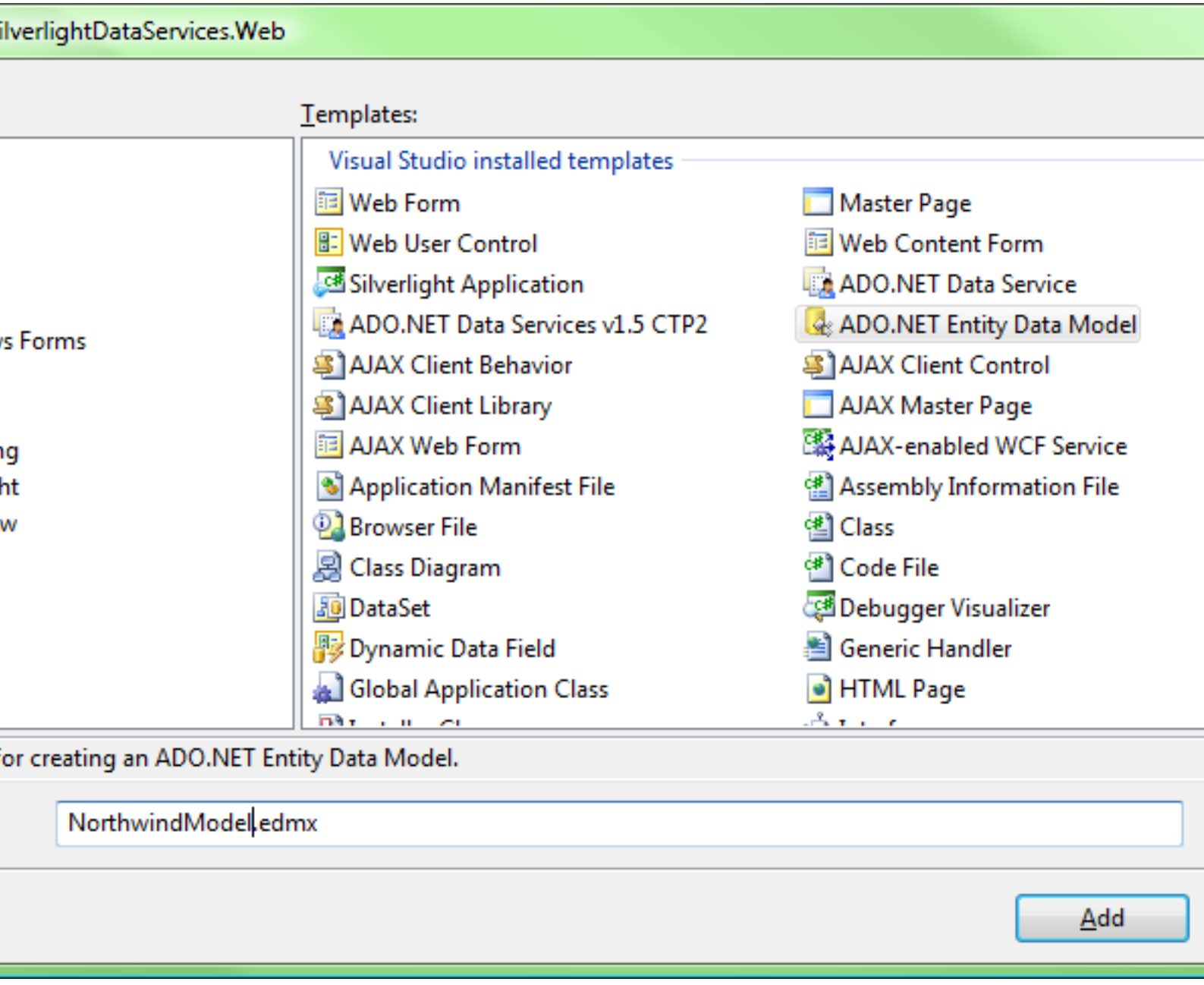
Tout d'abord, créons un projet Silverlight 3 basique.



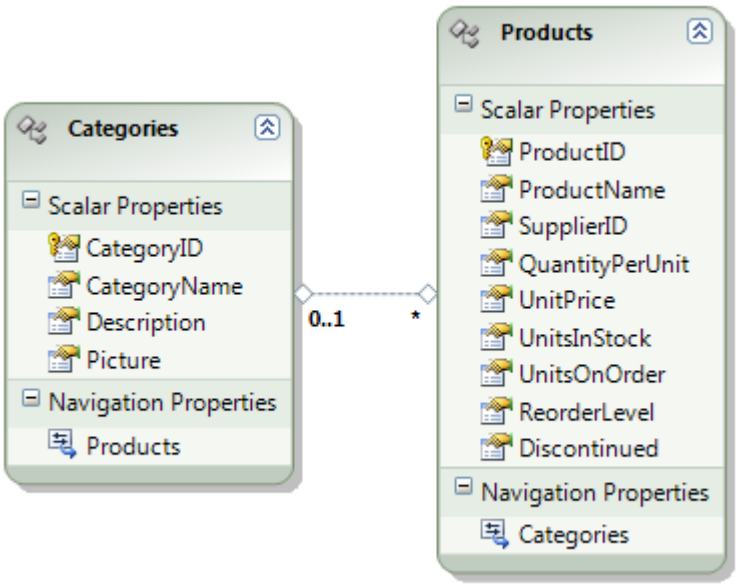
3 - Le projet Web

Occupons-nous maintenant de la création du projet côté Web.

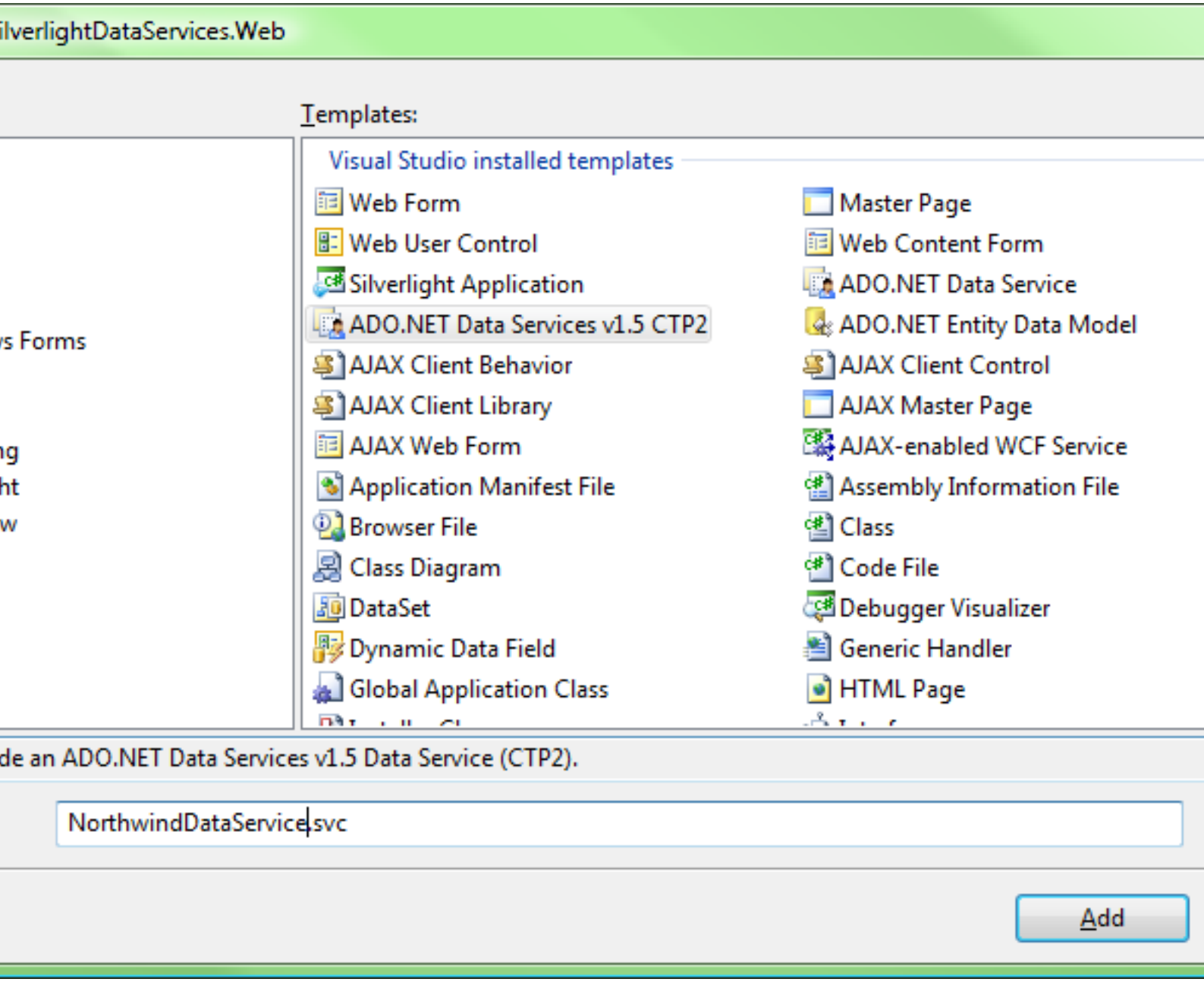
Nous ajoutons un nouvel item de type ADO.NET Entity Data Model, appelons le NorthwindModel.edmx.



Nous le lions à notre base de données Northwind et nous ajoutons les tables que nous voulons utiliser ; Categories et Products dans notre cas.



Il nous faut maintenant ajouter un nouvel item de type WCF Data Services v1.5 CTP2, nous allons l'appeler NorthwindDataService.svc.



Dans le code généré, nous allons gérer les permissions données aux utilisateurs de notre service. Dans notre cas, nous allons autoriser toutes les opérations sur toutes les tables exposées.

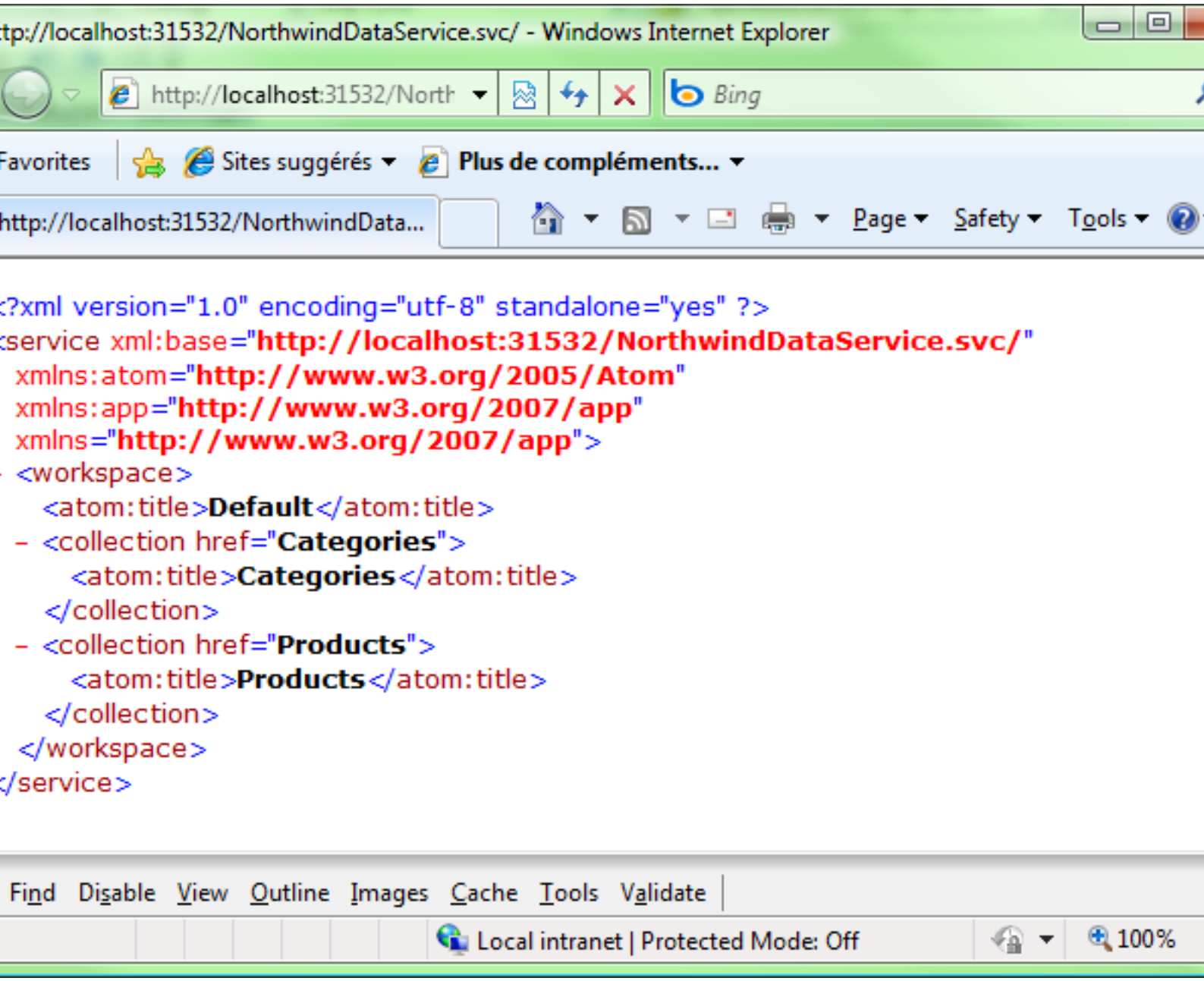
```
config.SetEntitySetAccessRule("*", EntitySetRights.All);
```

Avant cela, nous allons définir le type des données à exposer :

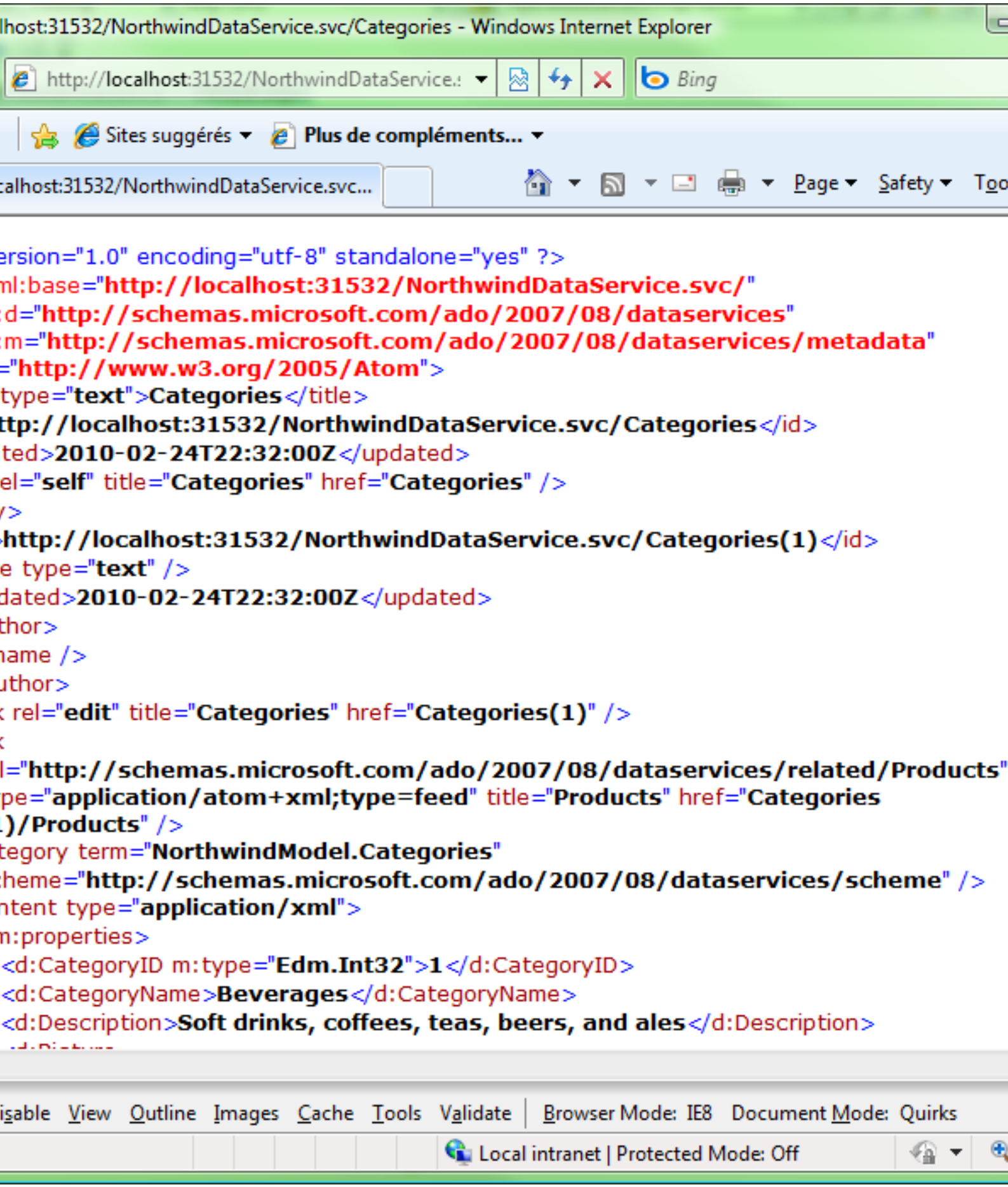
```
public class NorthwindDataService : DataService<NorthwindEntities>
```

Nous pouvons d'ores et déjà tester notre service. Clic droit -> Voir dans le navigateur.

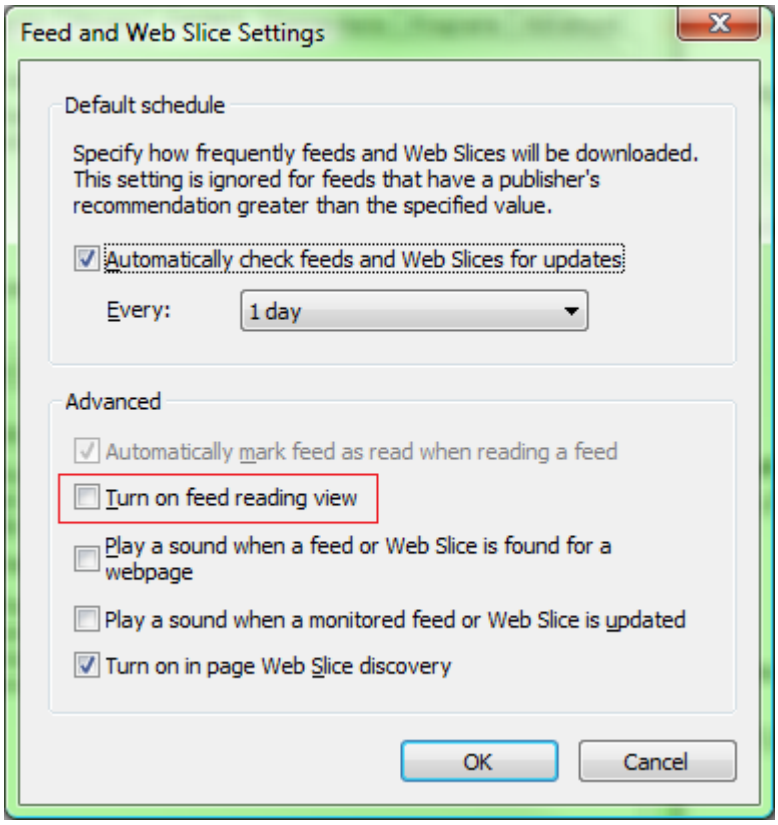
Vous devriez voir quelque chose dans ce genre.



Dans l'URL, vous pouvez rajouter Categories et vous obtiendrez ceci :



Si ce n'est pas le cas, il faut que vous désactiviez la lecture de flux dans les options de IE.



4 - Activer le Data Binding sur le client

Par défaut, les types générés côté client par WCF Data Services n'implémentent pas les interfaces de DataBinding (*INotifyPropertyChanged*). Pour activer cette fonctionnalité, il faut rajouter deux variables d'environnements sur notre machine :

- dscodegen_usedsc 1
- dscodegen_version 2.0

Il est également possible de générer le code du proxy à la main :

DataSvcUtil.exe /uri:**http://localhost...** /out:service.cs /dataservicecollection /version:2.0

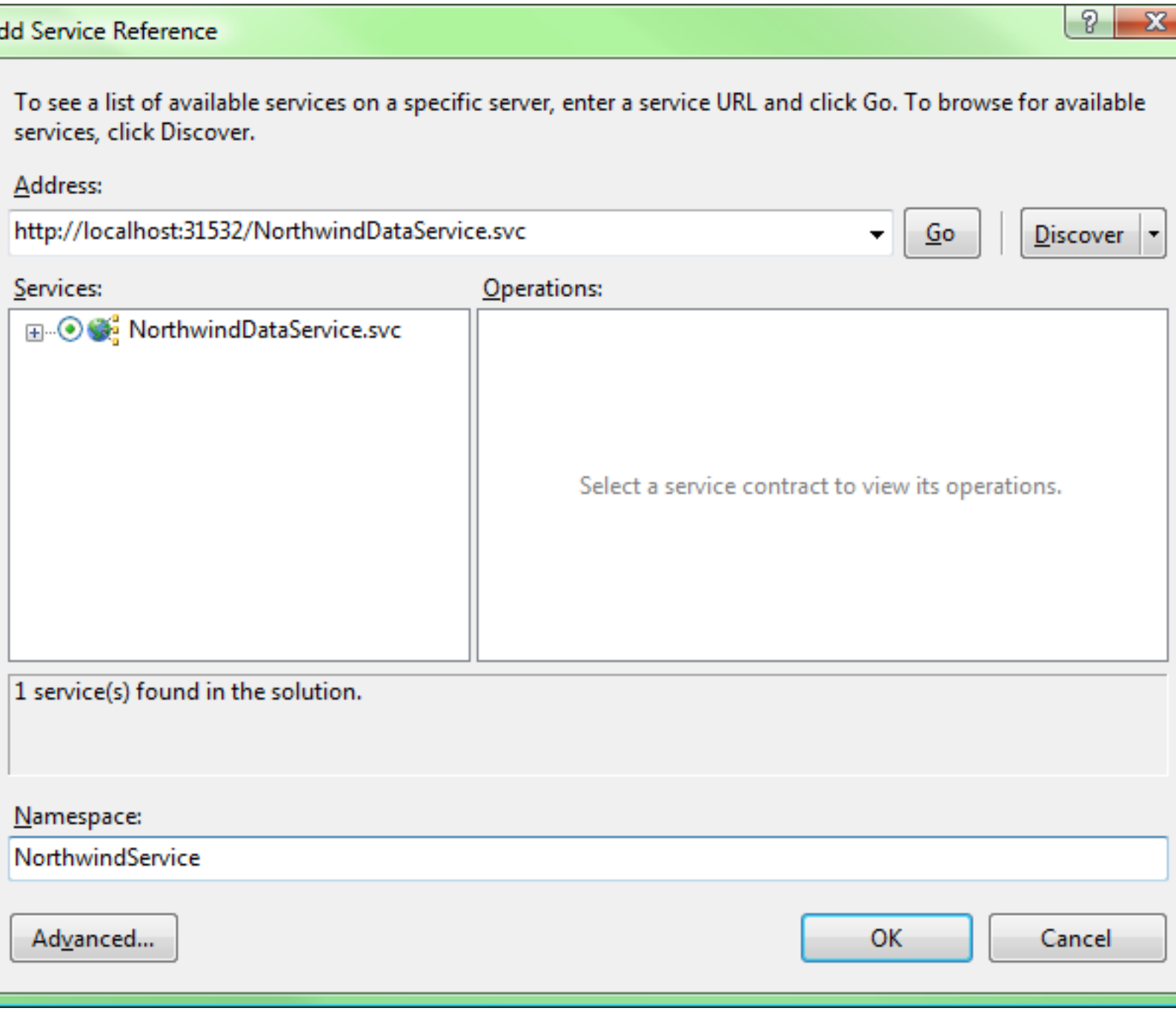
5 - Création du proxy côté client

Retour dans notre projet Silverlight. Nous allons maintenant rajouter une référence sur notre service créé plus haut.

Clic droit -> Add Service Reference.

Nous cliquons sur Discover puis nous choisissons NorthwindDataService.svc.

Dans Namespace on met NorthwindService.



Dans le cas d'une génération manuelle via DataSvcUtil.exe, cette étape est remplacée par l'ajout du fichier généré (service.cs dans l'exemple) à notre solution.

6 - Création et utilisation de notre Service côté client

Créons d'abord notre interface graphique :

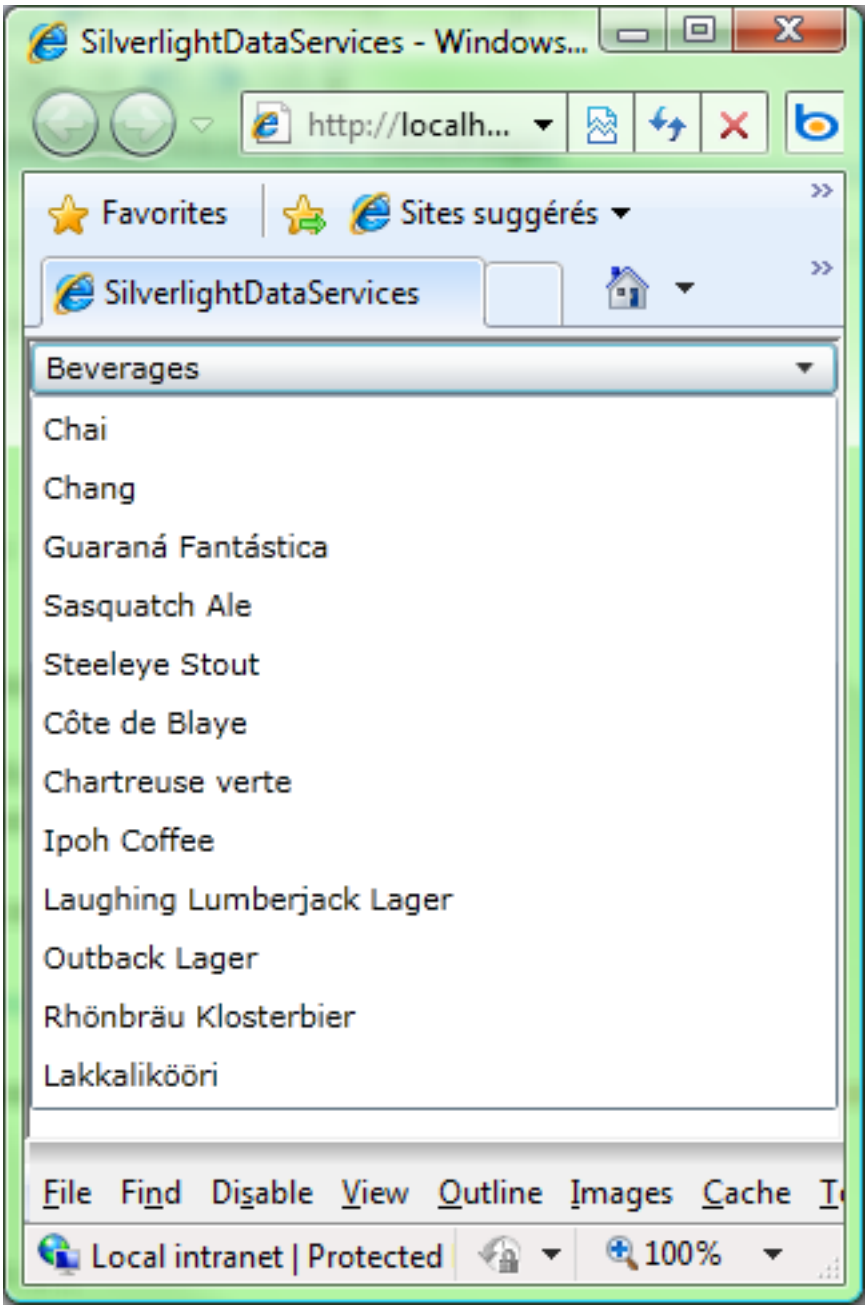
```
<StackPanel x:Name="LayoutRoot" Orientation="Vertical" Width="Auto">
    <ComboBox x:Name="cbCategories" DisplayMemberPath="CategoryName" />

  <ListBox ItemsSource="{Binding Path=SelectedItem.Products, ElementName=cbCategories}" DisplayMemberPath="Produc
</StackPanel>
```

Nous avons donc une ComboBox qui affichera toutes les catégories, et une ListBox qui affichera tous les produits de la catégorie séléctionnée.

Une fois notre référence ajoutée, nous pouvons créer et utiliser notre service.

```
public partial class MainPage : UserControl
{
    NorthwindEntities context = new NorthwindEntities(new Uri("/NorthwindDataService.svc",
 UriKind.Relative));

    public MainPage()
    {
        InitializeComponent();

        this.Loaded += new RoutedEventHandler(MainPage_Loaded);
    }

    void MainPage_Loaded(object sender, RoutedEventArgs e)
    {
        var queryCategories = context.Categories.Expand("Products");
        queryCategories.BeginExecute(result =>
        {
            cbCategories.ItemsSource = queryCategories.EndExecute(result);
            cbCategories.SelectedIndex = 0;
        }, null);
    }
}
```

Ce code charge toutes les catégories ainsi que les produits rattachés (*Expand("Products")*) et les affiche dans notre ComboBox.



Nous pouvons également ne charger qu'à la demande les produits reliés à une catégorie, ceci avec quelques lignes de code.

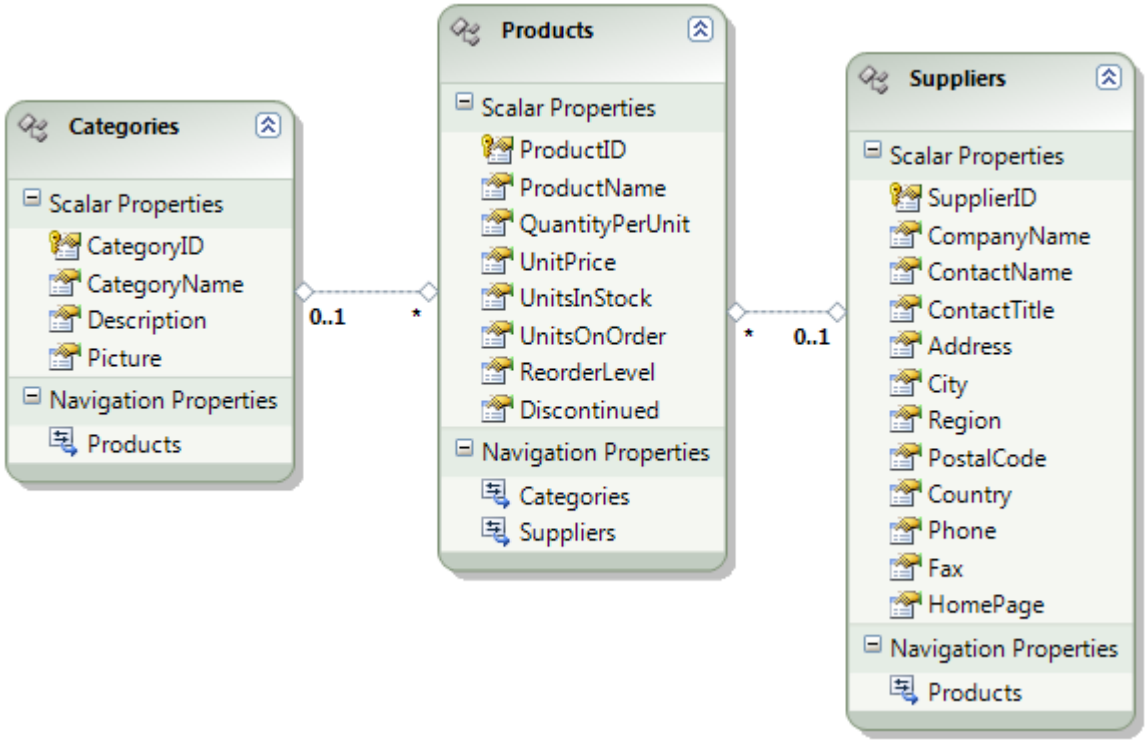
```
<Button Content="Load products" Click="Button_Click" />
```

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    Categories category = cbCategories.SelectedItem as Categories;

    context.BeginLoadProperty(category, "Products", (a) => context.EndLoadProperty(a), null);
}
```

7 - Ajouter/modifier/supprimer un produit

Dans cette partie, nous allons modifier notre modèle Entity Framework pour rajouter la table Suppliers.

Le designer de VS2008 ayant quelques limitations, le mieux est de supprimer le fichier edmx et d'en recréer un nouveau avec nos trois tables.



Nous mettons ensuite à jour la référence vers le service dans le client.

Créons tout d'abord notre interface en Xaml.

```
<StackPanel x:Name="LayoutRoot" Orientation="Vertical" Width="Auto">
    <ComboBox x:Name="cbCategories" DisplayMemberPath="CategoryName" Margin="10,5,10,5" />
    <ListBox x:Name="lbProducts" Margin="10,5,10,5" Height="150"
             ItemsSource="{Binding Path=SelectedItem.Products, ElementName=cbCategories}"
             DisplayMemberPath="ProductName"
             SelectionChanged="lbProducts_SelectionChanged" />
    <HyperlinkButton Content="New" Click="New_Click" Margin="10,5,10,5" />
    <Border CornerRadius="5" Background="LightBlue" Margin="10,5,10,5">
        <StackPanel x:Name="spProduct">
            <TextBox x:Name="txtName"
                     Margin="10,5,10,5" Width="150" HorizontalAlignment="Left"
                     Text="{Binding ProductName, Mode=TwoWay,
                            NotifyOnValidationError=true, ValidatesOnExceptions=true,
                            UpdateSourceTrigger=Explicit}" />
            <TextBox x:Name="txtPrice"
                     Margin="10,5,10,5" Width="150" HorizontalAlignment="Left"
                     Text="{Binding UnitPrice, Mode=TwoWay,
                            NotifyOnValidationError=true, ValidatesOnExceptions=true,
                            UpdateSourceTrigger=Explicit}" />
            <ComboBox x:Name="cbSuppliers"
                      DisplayMemberPath="CompanyName" Margin="10,5,10,5"
                      Width="250" HorizontalAlignment="Left"
                      SelectedItem="{Binding Suppliers, Mode=TwoWay,
                                     NotifyOnValidationError=true, ValidatesOnExceptions=true,
                                     UpdateSourceTrigger=Explicit}" />
            <StackPanel Orientation="Horizontal" Margin="10,5,10,5" >
```

```
                <HyperlinkButton Content="Update" Click="Update_Click" />
                <TextBlock Text=" | " />
                <HyperlinkButton Content="Delete" Click="Delete_Click" />
            </StackPanel>
        </StackPanel>
    </Border>
</StackPanel>
```

Nous gardons le Xaml du début et nous rajoutons simplement un cadre qui va servir à l'ajout et l'édition des produits, ainsi que trois boutons pour ajouter, éditer et supprimer.

Les Binding sont créés de façon à supporter la validation. Nous ajoutons également la propriété *UpdateSourceTrigger* de façon à ce que la mise à jour de la source soit faite explicitement en appelant la méthode *UpdateSource* d'un *BindingExpression*.

En parlant de validation, nous allons rajouter quelques règles sur notre produit.

Il faut tout d'abord créer un nouveau fichier *Products.cs* avec le code suivant :

```
namespace SilverlightDataServices.NorthwindService
{
    public partial class Products
    {
        private bool nameValid;
        partial void OnProductNameChanging(string value)
        {
            nameValid = false;

            if (string.IsNullOrEmpty(value)) throw new
 ArgumentException("Product's name cannot be empty");
            else nameValid = true;
        }

        private bool unitPriceValid;
        partial void OnUnitPriceChanging(decimal? value)
        {
            if (value < 0) throw new ArgumentException("Unit's price cannot be negative");
            else unitPriceValid = true;
        }

        public bool IsValid
        {
            get { return nameValid && unitPriceValid; }
        }
    }
}
```

Il faut bien veiller à ce que le namespace soit le même que celui où est définie la classe *Products* produite par l'ajout de la référence au service.

Nous définissons donc deux règles de validation :

- le nom ne peut pas être vide ;
- le prix unitaire ne peut être inférieur à 0.

Nous ajoutons également une propriété *IsValid* afin de savoir si l'objet est valide.

Le code pour l'ajout est relativement simple :

```
private void New_Click(object sender, RoutedEventArgs e)
{
    Products product = new Products();
```

```
    spProduct.DataContext = product;
}
```

Nous créons un nouveau produit que nous bindons sur le panel contenant nos contrôles d'édition.

Pour la modification :

```
private void lbProducts_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    Products product = lbProducts.SelectedItem as Products;

    if (product == null) return;

    // we load the Suppliers property is its not loaded
    if (product.Suppliers == null) context.BeginLoadProperty(product, "Suppliers", (a) =>
 context.EndLoadProperty(a), null);

    spProduct.DataContext = product;
}
```

Lorsque la sélection de la ListBox change, nous prenons le nouvel item et nous le bindons sur notre panel. Juste avant cela, nous chargeons la propriété *Suppliers* si elle ne l'est pas déjà.

Le code de la suppression est un peu plus compliqué.

```
private void Delete_Click(object sender, RoutedEventArgs e)
{
    Products product = (sender as FrameworkElement).DataContext as Products;
    Categories category = cbCategories.SelectedItem as Categories;

    // we dont delete a new product not saved
    if (product.ProductID == 0) return;

    // delete the product in the db
    context.DeleteObject(product);

    // we save the changes to the DB through the service
    context.BeginSaveChanges(SaveChangesOptions.Batch, (a) =>
    {
        try
        {
            // save
            context.EndSaveChanges(a);

            // delete the product from the category
            category.Products.Remove(product);
        }
        catch (Exception)
        {
            HtmlPage.Window.Alert("Cannot delete the product");
            UnDeleteProduct(product);
        }
    }, null);
}

private void UnDeleteProduct(Products product)
{
    context.Detach(product);
    context.AttachTo("Products", product);
    context.UpdateObject(product);
}
```

Nous récupérons le produit et la catégorie concernée. Si le produit vient juste d'être créé mais qu'il n'a pas encore été sauvé (id = 0), alors on ne fait rien. Sinon nous appelons la méthode *DeleteObject* de notre contexte qui met l'état de notre objet à *Deleted*. Ensuite nous appelons la méthode *SaveChanges* de manière asynchrone. Si la suppression

fonctionne, nous enlevons le produit de la liste des produits de la catégorie, si elle ne fonctionne pas nous affichons un message et changeons l'état de l'objet de façon à ce qu'il ne soit plus en *Deleted*.

Pour cela il faut suivre ce petit tableau explicatif :

Etat	Après DeleteObject()	Pour annuler DeleteObject()
Added	Detached	context.AddObject()
Unchanged	Deleted	context.Detach(),context.AttachTo()
Modified	Deleted	context.Detach(),context.AttachTo(),context.Update

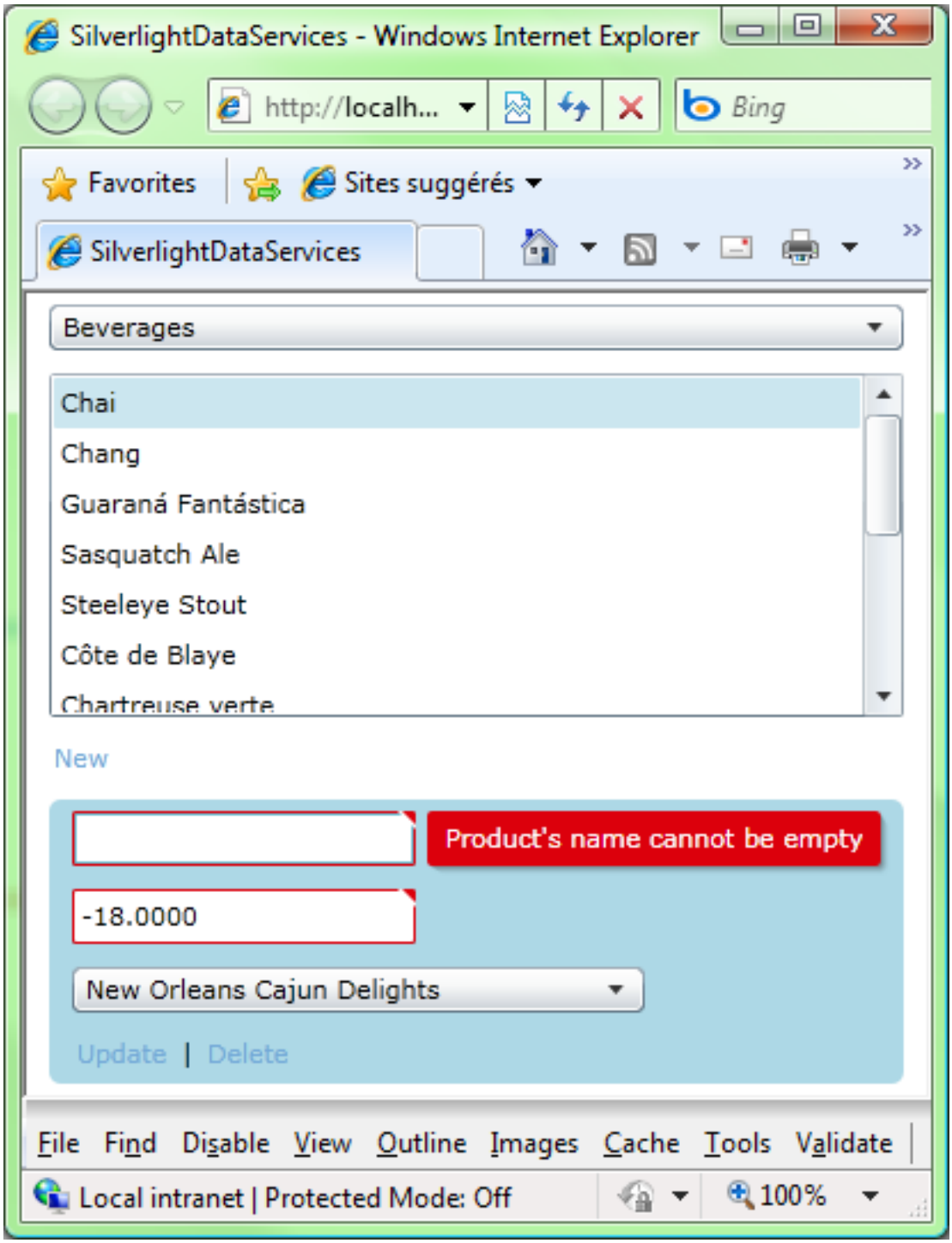
Lien : **http://social.msdn.microsoft.com/Forums/en-US/adodotnetdataservices/thread/8685680c-7591-48eb-9546-b64f856fe125**

Dans notre cas, notre objet est soit en *Unchanged*, soit en *Modified*, j'applique donc la dernière ligne.

Voyons maintenant le code pour ajouter/modifier un produit.

```
private void Update_Click(object sender, RoutedEventArgs e)
{
    Products product = (sender as FrameworkElement).DataContext as Products;
    Categories category = cbCategories.SelectedItem as Categories;

    txtName.GetBindingExpression(TextBox.TextProperty).UpdateSource();
    txtPrice.GetBindingExpression(TextBox.TextProperty).UpdateSource();

    // update the supplier combobox binding
    cbSuppliers.GetBindingExpression(ComboBox.SelectedItemProperty).UpdateSource();

    // check if the product is valid (name not empty and unit price > 0)
    if (product.IsValid == false) return;

    // new product
    if (product.ProductID == 0)
    {
        context.AddToProducts(product);
        category.Products.Add(product);

        // add a link between the category and the new product
        context.AddLink(category, "Products", product);
    }
    else
    {
        // we update the object
        context.UpdateObject(product);
    }

    // we explicitly set the link between the product and the new supplier
    context.SetLink(product, "Suppliers", cbSuppliers.SelectedItem);

    // we save the changes to the DB through the service
    context.BeginSaveChanges(SaveChangesOptions.Batch, (a) => context.EndSaveChanges(a), null);
}
```

Tout d'abord, nous récupérons les produits à sauvegarder ainsi que la catégorie associée. Ensuite, nous mettons les binding à jour afin que les valeurs contenues dans les *TextBox* et la *ComboBox* soient assignées à l'objet. Si les valeurs sont valides, alors nous continuons.

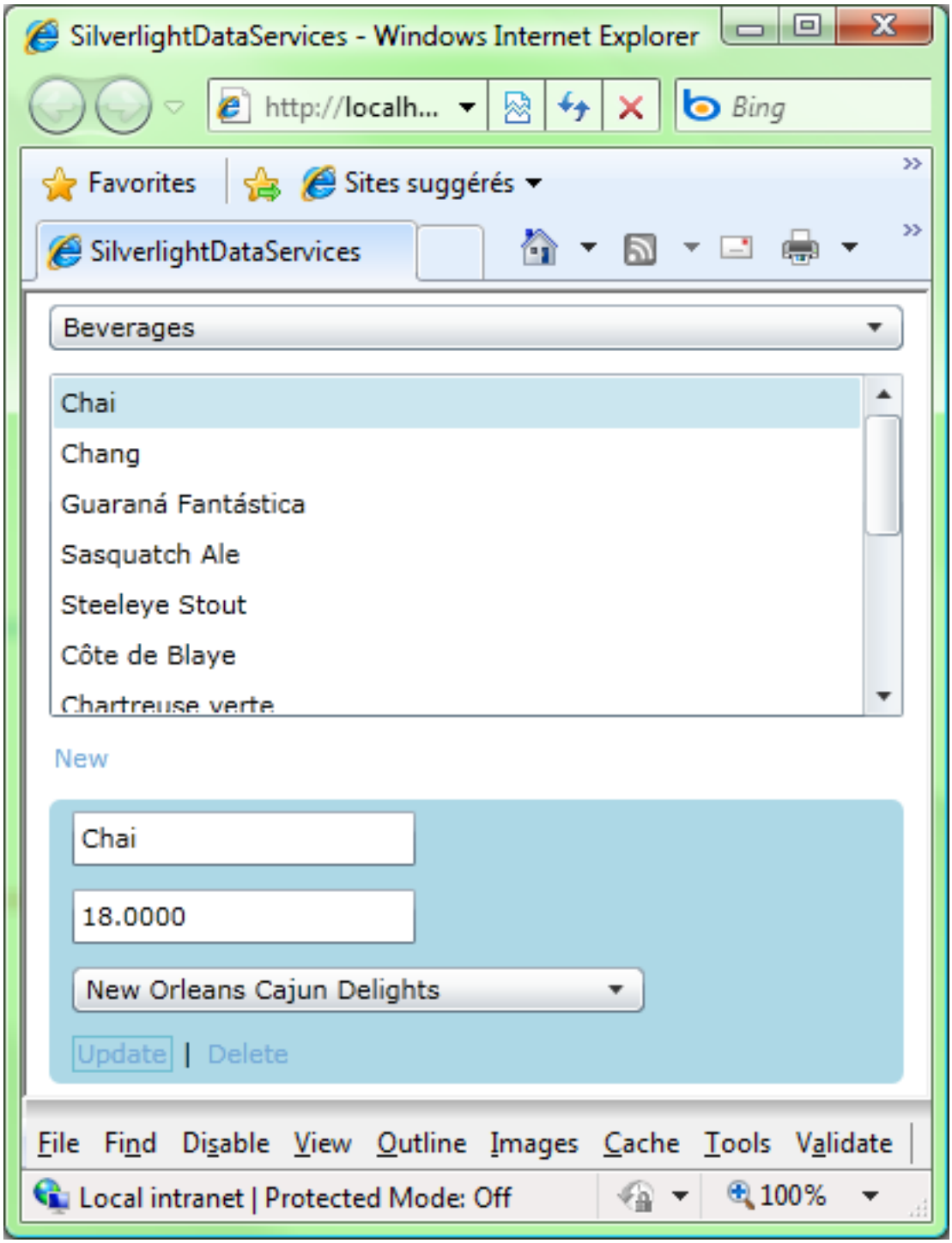


Si le produit est un nouveau produit, nous l'ajoutons à la table *Products*, puis aux produits de la catégorie, et enfin nous ajoutons un lien entre la catégorie et le produit. Sans cette dernière ligne l'ajout se fera bien au niveau de la base de données, mais il n'y aura pas de lien entre la catégorie et le produit.

Dans le cas d'une mise à jour de l'objet nous mettons son état à *Modified* via la méthode *UpdateObject*.

Pour finir nous mettons un lien entre le produit et le fournisseur choisi (sans cette ligne, le changement ne sera pas effectué au niveau de la base de données), puis nous sauvegardons.

Si vous testez, vous devriez obtenir ceci :



8 - Conclusion

Comme vous avez pu le constater, la version 1.5 de WCF Data Services est parfaitement adaptée à une utilisation avec Silverlight.

Le problème maintenant est de choisir la solution la plus adaptée à vos besoins (WCF Data Services ? RIA Services ? WCF Services ?).

Pour ma part, je préfère WCF Data Services à RIA Services !

9 - Remerciements

Merci à toute l'équipe .NET pour leurs relectures techniques, ainsi qu'à **eusebe19** et **jacques jean** pour leur relecture orthographique et grammaticale.