

Présentation d'ADO.NET Data Services

Cet article a pour objectif de vous présenter ADO.NET Data Services, anciennement connu sous le nom de code « Astoria », dont le but est de permettre aux développeurs de requêter des sources de données exposées sur le Web.

Sommaire

Introduction	3
Pourquoi ADO.NET Data Services ?.....	3
Les pré-requis nécessaires	4
Création d'un service de données Web	5
Consommation du service de données.....	12
Un peu plus loin avec ADO.NET Data Services.....	15
Conclusions	20

Introduction

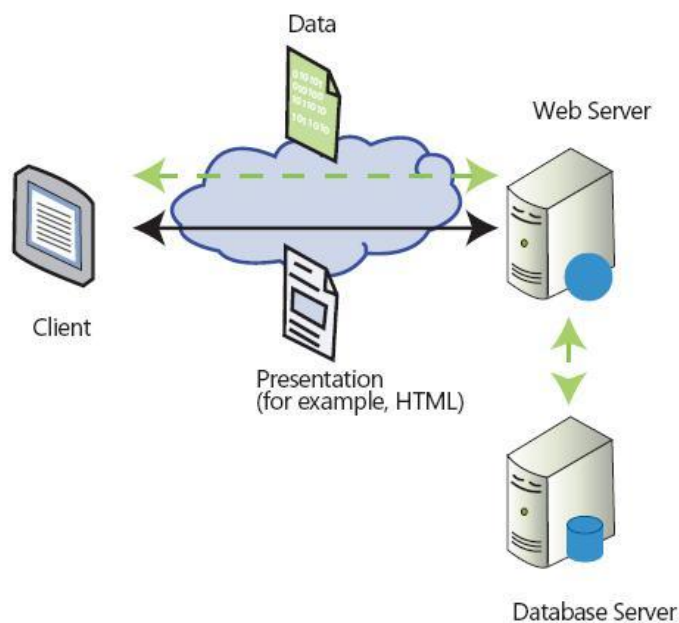
ADO.NET Data Services est un framework, respectant la technologie REST, permettant aux données d'être exposées, via le Web, comme un service flexible. Il est ainsi possible, lors de vos développements, de consommer ces données et cela au travers d'un réseau local ou d'Internet.

Le format des données échangées entre le serveur et les clients est l'un des standards de l'industrie, tel qu'**AtomPub** et **JSON** (*JavaScript Object Notation*).

Pour accéder au service, il suffit d'utiliser des requêtes HTTP, et les ordres GET, POST, PUT et DELETE pour faire des opérations de création, lecture, mise à jour et suppression sur le service de données.

Pourquoi ADO.NET Data Services ?

ADO.NET Data Services a pour objectif de répondre à un besoin de plus en plus fréquent lors du développement de nos applications : accéder à des données, des informations, à travers le Web sans pour autant devoir passer par des applications ou des services Web.



Que l'on développe une « *Rich Desktop Application* » (WPF, WindowsForms) ou une « *Rich Internet Application* » (ASP.NET AJAX, Silverlight), cette interaction avec les données est obligatoire.

Ce framework va donc être utilisé pour simplifier le processus d'accès aux données et le rendre plus flexible qu'il n'est actuellement. Pour cela, on va retrouver une librairie nous permettant de mapper, automatiquement, les données de notre modèle en objets .NET.

Les pré-requis nécessaires

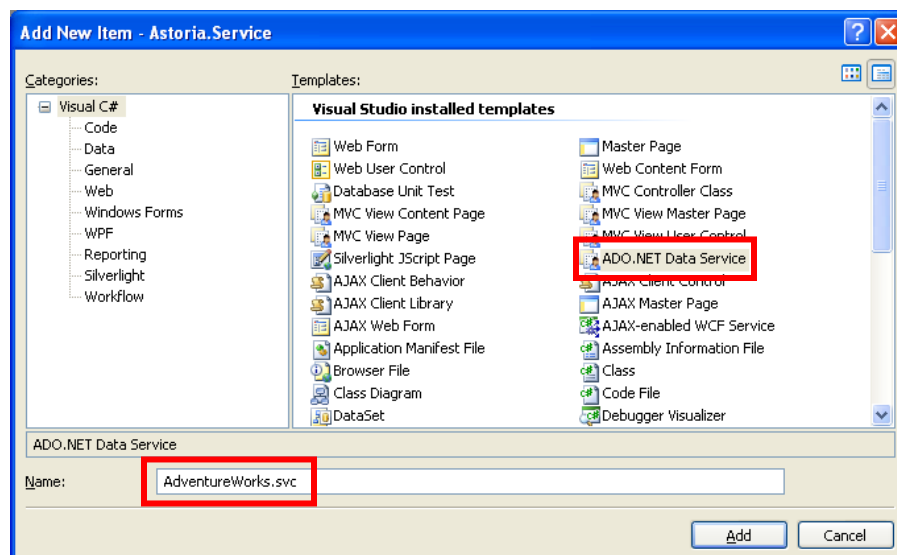
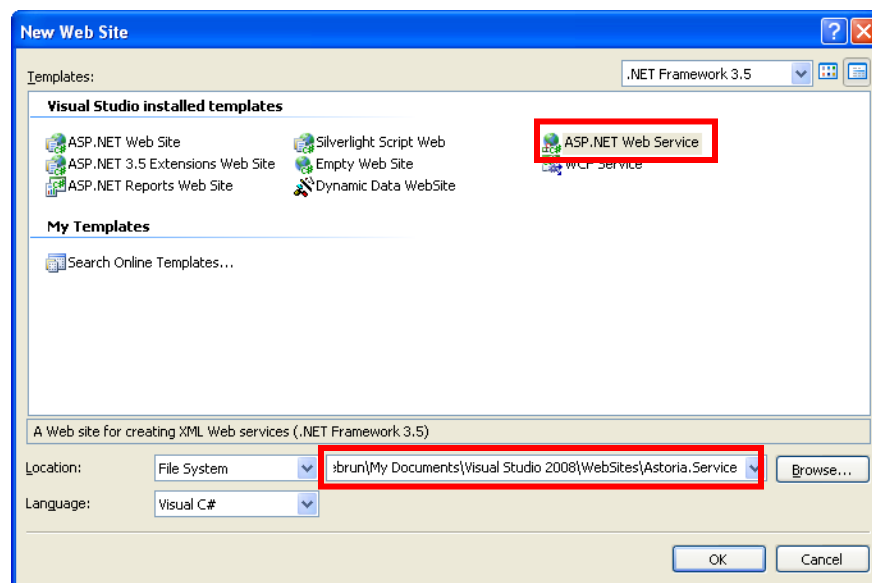
Pour travailler avec ADO .NET Data Services, il vous faudra plusieurs choses :

- Visual Studio 2008
- Un serveur de base de données (tel que SQL Server, y compris une version Express)
- ADO.NET Data Services et le Runtime de l'**Entity Framework Beta 3** : ces deux produits sont actuellement installés en même temps lorsque vous installez ADO.NET Data Services, via les [extensions d'ASP.NET 3.5](#) (disponible en CTP de Décembre à l'écriture de cet article).
Cependant, sachez que les outils pour l'Entity Framework sont disponibles, via un téléchargement à part, à l'adresse suivante :
<http://www.microsoft.com/downloads/details.aspx?FamilyId=D8AE4404-8E05-41FC-94C8-C73D9E238F82&displaylang=en>
- La CTP de la Beta 3 des **Entity Framework Tools**, disponible à l'adresse suivante :
<http://www.microsoft.com/downloads/details.aspx?FamilyId=D8AE4404-8E05-41FC-94C8-C73D9E238F82&displaylang=en>

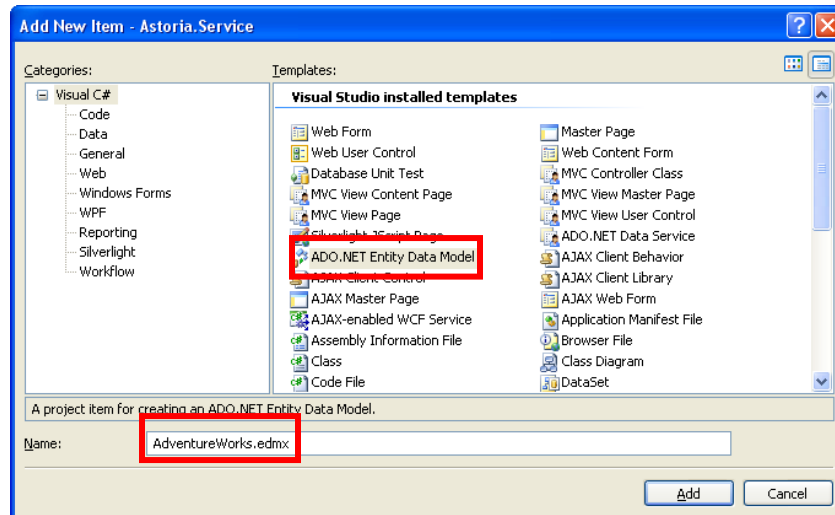
Création d'un service de données Web

La première chose à faire, pour travailler avec ADO.NET Data Services, est de créer un service de données Web. C'est ce service, qui sera exposé sur Internet, et que nous utiliserons dans une application.

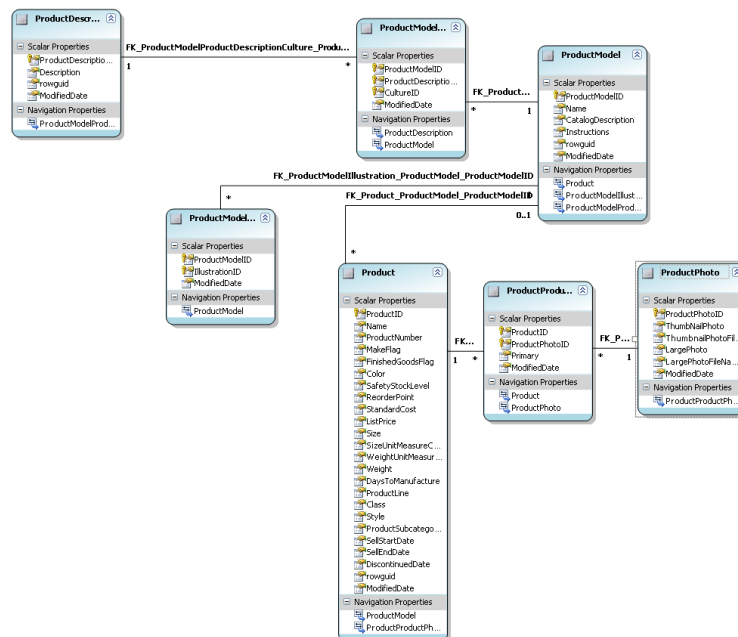
Pour créer ce service, nous allons créer un simple site Web sur lequel nous allons ajouter un élément de type « ADO.NET Data Services ». Notez que, pour le moment, nous créons un site Web car il n'y a pas, dans Visual Studio, de modèle de projet spécifique au service de données Web :



Nous allons à présent créer le modèle de données de notre application. Pour cela, ajoutez un nouvel élément, de type « *ADO.NET Entity Data Model* », à votre projet :



Suivez les différentes étapes de l'assistant qui survient à l'écran et ajoutez les tables qui vous seront nécessaires. Pour cet article, j'ai utilisé un serveur SQL Server sur lequel j'ai ajouté la base de données « *AdventureWorks* » :



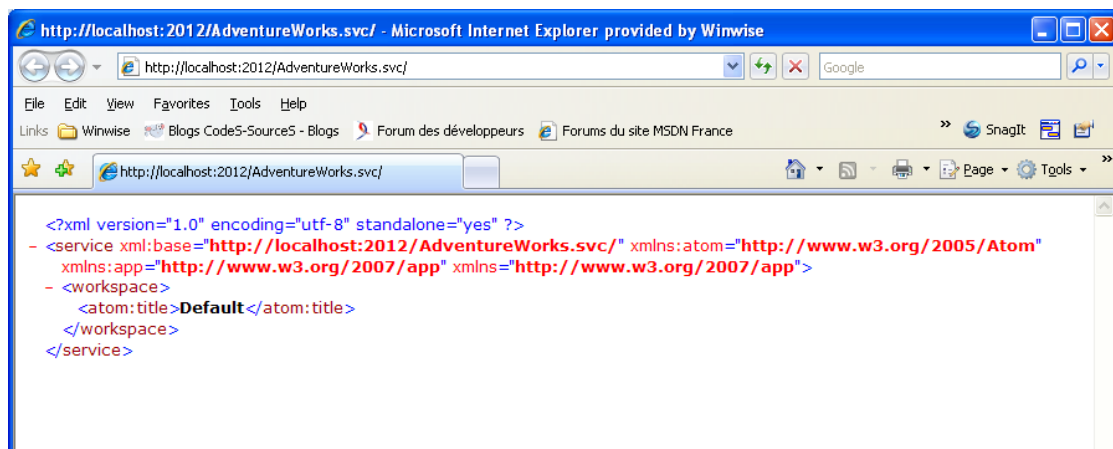
A présent, il nous faut éditer le code de notre service pour y ajouter plusieurs choses. Ouvrez le fichier « **AdventureWorks.svc.cs** » et repérez la ligne suivante :

```
/* TODO: put your data source class name here */
```

Comme indiqué dans le commentaire, vous devez remplacer cette ligne par le nom de la classe qui sera utilisée comme source de données. Etant donné que nous avons utilisé LINQ To Entities, nous allons utiliser « **AdventureWorksEntities** ». Votre code devient donc :

```
public class AdventureWorks : WebDataService<AdventureWorksModel.AdventureWorksEntities>
```

Il ne reste plus qu'à tester notre service. Pour cela, un simple appui sur la touche F5 devrait produire un résultat similaire à celui-ci :



Comme vous pouvez le constater, on ne voit rien de spécifique apparaître. La raison est simple : dans la **CTP** (*Community Technology Preview*) de Décembre 2007 d'ADO.NET Data Services, un contrôle d'accès a été intégré, empêchant n'importe qui de voir les données, à moins de le spécifier explicitement, ce que nous allons faire immédiatement.

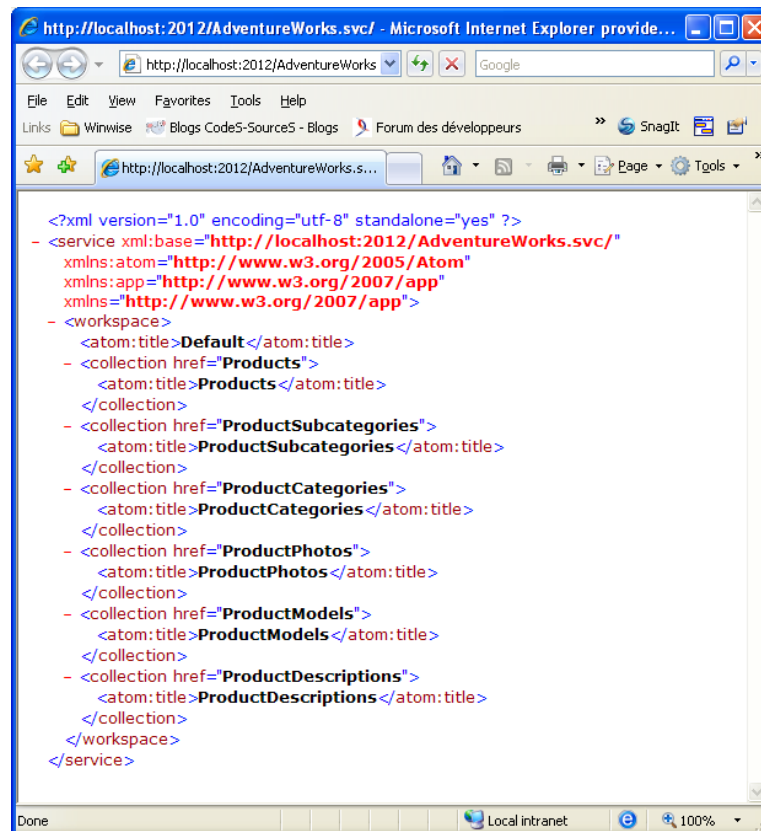
Retournez dans le code de votre service et repérez la méthode nommée **InitializeService** (il s'agit de la seule méthode de la classe). C'est dans cette méthode que vous allez pouvoir paramétrer la visibilité et le niveau d'accessibilité des ressources auxquelles vous voulez accéder.

Pour faire cela, il vous suffit d'appeler la méthode **SetResourceContainerAccessRule** de l'objet nommé « *config* » (de type **IWebDataServiceConfiguration**) qui est passé en paramètre. Cette méthode prend en paramètre 2 choses :

- Une chaîne de caractères représentant le nom de la ressource à laquelle vous voulez donner accès. Il est dommage que l'on soit obligé de passer une chaîne de caractères « en dur » plutôt qu'une propriété ou une énumération. En effet, étant donné que nous avons indiqué que la source des données serait de type *AdventureWorksDataContext*, il devrait être possible d'avoir accès aux différentes propriétés (représentant les tables). Peut-être cela sera-t-il corrigé/amélioré dans les prochaines versions ?
- Une des valeurs de l'énumération **ResourceContainerRights**, permettant d'indiquer quels droits seront appliqués à la ressource indiquée.

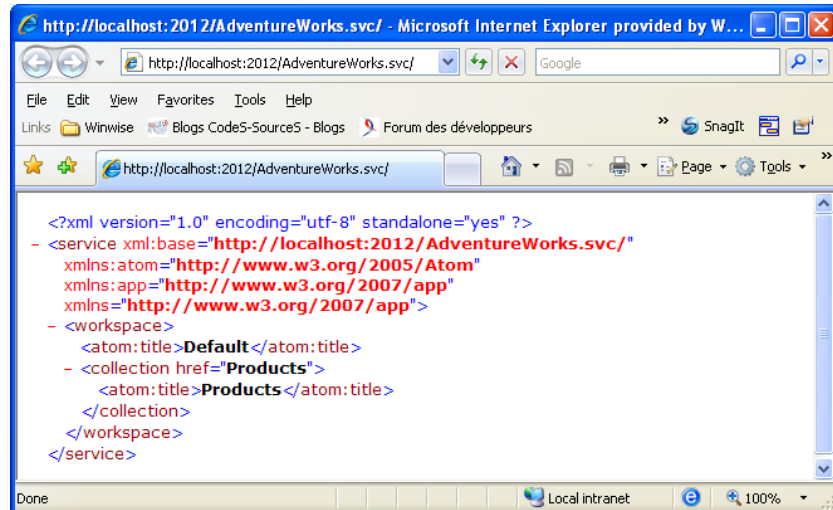
A des fins de tests, vous pouvez utiliser la ligne suivante pour donner accès à toutes les ressources (en lecture). Cependant, gardez à l'esprit que cela n'est valable qu'à des fins de tests :

```
config.SetResourceContainerAccessRule ("*", ResourceContainerRights.AllRead);
```

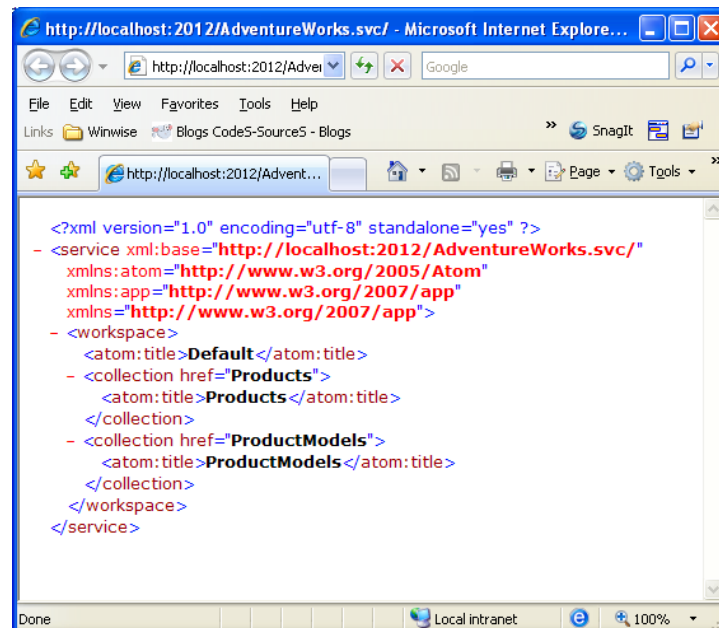


Voici différents exemples et le résultat que l'on observe à l'exécution :

```
// Accès à la ressource "Products" en lecture
config.SetResourceContainerAccessRule("Products", ResourceContainerRights.AllRead);
```

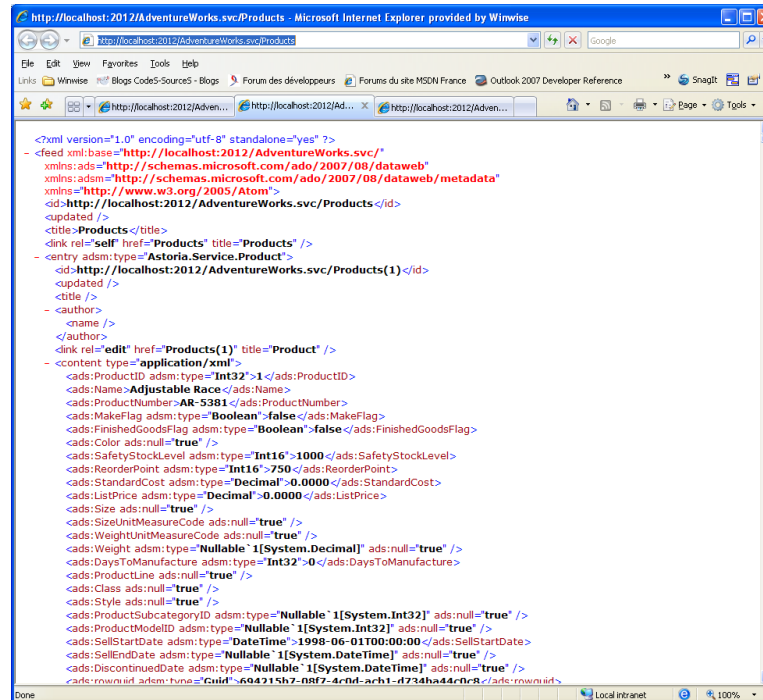


```
// Accès à la ressource "Products" en lecture
// Et à la ressource ProductModel en lecture/Ecriture
config.SetResourceContainerAccessRule("Products", ResourceContainerRights.AllRead);
config.SetResourceContainerAccessRule("ProductModels", ResourceContainerRights.All);
```



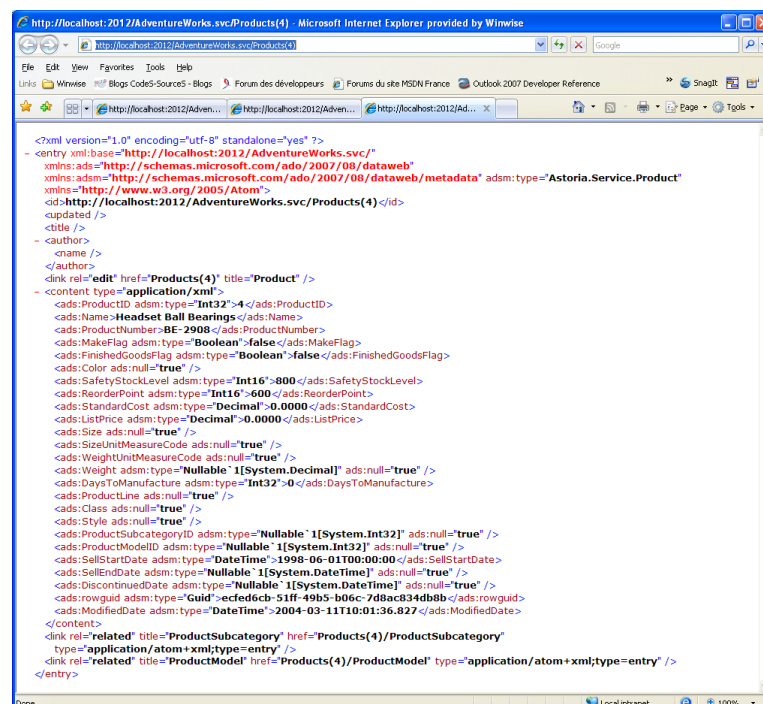
Si vous êtes attentifs au XML écrit à l'écran, vous avez sans doute remarqué que certains des éléments, nommés **collection**, portent l'attribut **href**. Cela signifie tout simplement qu'il s'agit d'une collection avec laquelle vous allez pouvoir travailler, autrement dit dans laquelle vous pourrez naviguer :

- Afficher la liste de tous les produits : <http://localhost:2012/AdventureWorks.svc/Product>



```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <feed xml:base="http://localhost:2012/AdventureWorks.svc/"
  xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb"
  xmlns:adsm="http://schemas.microsoft.com/ado/2007/08/dataweb/metadata"
  xmlns="http://www.w3.org/2005/Atom">
  <id>http://localhost:2012/AdventureWorks.svc/Products</id>
  <updated />
  <title>Products</title>
  <link rel="self" href="Products" title="Products" />
  <entry adsm:type="Astoria.Service.Product">
    <id>http://localhost:2012/AdventureWorks.svc/Products(1)</id>
    <updated />
    <title />
    <author>
      <name />
    </author>
    <link rel="edit" href="Products(1)" title="Product" />
    <content type="application/xml">
      <ads:ProductID adsm:type="Int32">1</ads:ProductID>
      <ads:Name>Adjustable Race</ads:Name>
      <ads:ProductNumber>AR-5381</ads:ProductNumber>
      <ads:MakeFlag adsm:type="Boolean">false</ads:MakeFlag>
      <ads:FinishedGoodsFlag adsm:type="Boolean">false</ads:FinishedGoodsFlag>
      <ads:Color ads:null="true" />
      <ads:SafetyStockLevel adsm:type="Int16">1000</ads:SafetyStockLevel>
      <ads:ReorderPoint adsm:type="Int16">750</ads:ReorderPoint>
      <ads:StandardCost adsm:type="Decimal">0.0000</ads:StandardCost>
      <ads>ListPrice adsm:type="Decimal">0.0000</ads>ListPrice>
      <ads:Size ads:null="true" />
      <ads:SizeUnitMeasureCode ads:null="true" />
      <ads:WeightUnitMeasureCode ads:null="true" />
      <ads:Weight adsm:type="Nullable`1[System.Decimal]" ads:null="true" />
      <ads:DaysToManufacture adsm:type="Int32">0</ads:DaysToManufacture>
      <ads:ProductLine ads:null="true" />
      <ads:Class ads:null="true" />
      <ads:ProductSubcategoryID adsm:type="Nullable`1[System.Int32]" ads:null="true" />
      <ads:ProductModelID adsm:type="Nullable`1[System.Int32]" ads:null="true" />
      <ads:SellStartDate adsm:type="DateTime">1998-06-01T00:00:00</ads:SellStartDate>
      <ads:SellEndDate adsm:type="Nullable`1[System.DateTime]" ads:null="true" />
      <ads:DiscontinuedDate adsm:type="Nullable`1[System.DateTime]" ads:null="true" />
      <ads:rowguid adsm:type="Guid">694713b7-bdf7-4c0d-8c11-d734ba44c1b9</ads:rowguid>
    </content>
  </entry>
</feed>
```

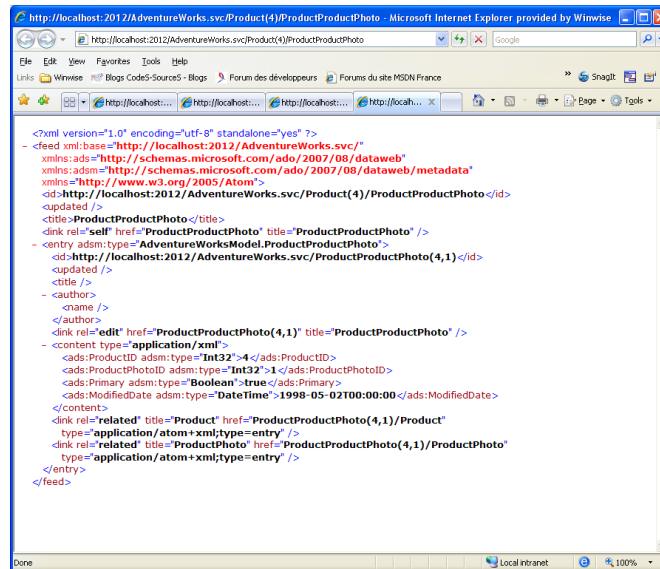
- Afficher le 4^{ème} produit de la liste : [http://localhost:2012/AdventureWorks.svc/Product\(4\)](http://localhost:2012/AdventureWorks.svc/Product(4))



```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <entry xml:base="http://localhost:2012/AdventureWorks.svc/"
  xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb"
  xmlns:adsm="http://schemas.microsoft.com/ado/2007/08/dataweb/metadata" adsm:type="Astoria.Service.Product"
  xmlns="http://www.w3.org/2005/Atom">
  <id>http://localhost:2012/AdventureWorks.svc/Products(4)</id>
  <updated />
  <title />
  <author>
    <name />
  </author>
  <link rel="edit" href="Products(4)" title="Product" />
  <content type="application/xml">
    <ads:ProductID adsm:type="Int32">4</ads:ProductID>
    <ads:Name>Headset Ball Bearings</ads:Name>
    <ads:ProductNumber>BE-2908</ads:ProductNumber>
    <ads:MakeFlag adsm:type="Boolean">false</ads:MakeFlag>
    <ads:FinishedGoodsFlag adsm:type="Boolean">false</ads:FinishedGoodsFlag>
    <ads:Color ads:null="true" />
    <ads:SafetyStockLevel adsm:type="Int16">800</ads:SafetyStockLevel>
    <ads:ReorderPoint adsm:type="Int16">600</ads:ReorderPoint>
    <ads:StandardCost adsm:type="Decimal">0.0000</ads:StandardCost>
    <ads>ListPrice adsm:type="Decimal">0.0000</ads>ListPrice>
    <ads:Size ads:null="true" />
    <ads:SizeUnitMeasureCode ads:null="true" />
    <ads:WeightUnitMeasureCode ads:null="true" />
    <ads:Weight adsm:type="Nullable`1[System.Decimal]" ads:null="true" />
    <ads:DaysToManufacture adsm:type="Int32">0</ads:DaysToManufacture>
    <ads:ProductLine ads:null="true" />
    <ads:Class ads:null="true" />
    <ads:ProductSubcategoryID adsm:type="Nullable`1[System.Int32]" ads:null="true" />
    <ads:ProductModelID adsm:type="Nullable`1[System.Int32]" ads:null="true" />
    <ads:SellStartDate adsm:type="DateTime">1998-06-01T00:00:00</ads:SellStartDate>
    <ads:SellEndDate adsm:type="Nullable`1[System.DateTime]" ads:null="true" />
    <ads:DiscontinuedDate adsm:type="Nullable`1[System.DateTime]" ads:null="true" />
    <ads:rowguid adsm:type="Guid">ecfed6cb-51ff-49b5-b06c-7d8ac834db8b</ads:rowguid>
    <ads:ModifiedDate adsm:type="DateTime">2004-03-11T10:01:36.827</ads:ModifiedDate>
  </content>
  <link rel="related" title="ProductSubcategory" href="Products(4)/ProductSubcategory"
    type="application/atom+xml;type=entry" />
  <link rel="related" title="ProductModel" href="Products(4)/ProductModel" type="application/atom+xml;type=entry" />
</entry>
```

- Afficher la photo du 4^{ème} produit :

[http://localhost:2012/AdventureWorks.svc/Products\(4\)/ProductProductPhoto](http://localhost:2012/AdventureWorks.svc/Products(4)/ProductProductPhoto)



- Récupérer seulement les produits dont le ProductID est inférieur à 1 :

[http://localhost:2012/AdventureWorks.svc/Product?\\$filter=ProductID lt 2](http://localhost:2012/AdventureWorks.svc/Product?$filter=ProductID lt 2)

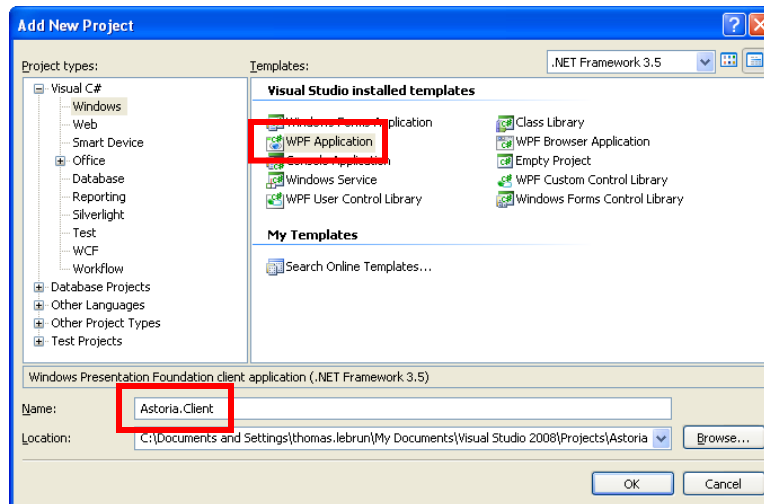
```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <feed xmlns:base="http://localhost:2012/AdventureWorks.svc/"
  xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb"
  xmlns:adsm="http://schemas.microsoft.com/ado/2007/08/dataweb/metadata"
  xmlns="http://www.w3.org/2005/Atom">
  <id>http://localhost:2012/AdventureWorks.svc/Products</id>
  <updated />
  <title>Products</title>
  <link rel="self" href="http://localhost:2012/AdventureWorks.svc/Products" title="Products" />
  <entry adsm:type="Astoria.Service.Product">
    <id>http://localhost:2012/AdventureWorks.svc/Products(1)</id>
    <updated />
    <title />
    <author>
      <name />
    </author>
    <link rel="edit" href="http://localhost:2012/AdventureWorks.svc/Products(1)" title="Product" />
    <content type="application/xml">
      <ads:ProductID adsm:type="Int32">1</ads:ProductID>
      <ads:ProductPhotoID adsm:type="Int32">1</ads:ProductPhotoID>
      <ads:Primary adsm:type="Boolean">true</ads:Primary>
      <ads:ModifiedDate adsm:type="DateTime">1998-05-02T00:00:00</ads:ModifiedDate>
    </content>
    <link rel="related" title="Product" href="http://localhost:2012/AdventureWorks.svc/Products(1)/Product"
      type="application/atom+xml;type=entry" />
    <link rel="related" title="ProductPhoto" href="http://localhost:2012/AdventureWorks.svc/Products(1)/ProductPhoto"
      type="application/atom+xml;type=entry" />
  </entry>
</feed>
```

Pour en savoir plus sur les différentes possibilités de filtre, tri, etc. par URI, rendez-vous à cette adresse :

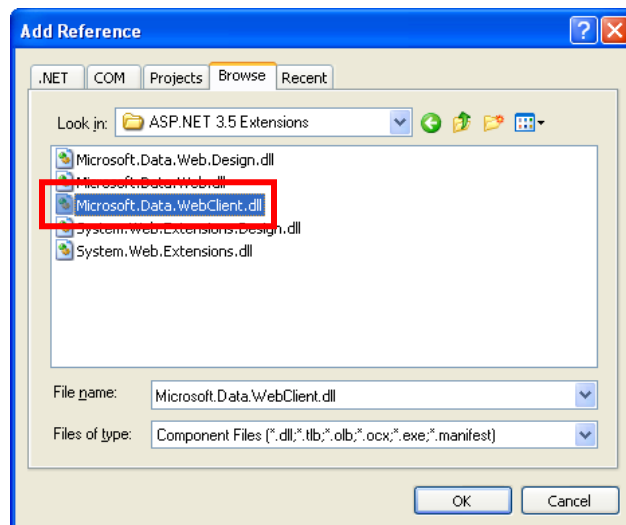
<http://quickstarts.asp.net/3-5-extensions/adonetdataservice/SimpleAddressingSchemeForDataWithUniformURLs.aspx>

Consommation du service de données

Pour consommer notre service de données via le Web, nous allons développer une simple application WPF (mais nous aurions tout aussi bien pu choisir de développer une application WindowsForms ou une application Web).



Ajoutez alors une référence à la librairie client d'ADO.NET Data Services, nommée **Microsoft.Data.WebClient.dll** (celle-ci se trouve dans le répertoire : C:\Program Files\Reference Assemblies\Microsoft\Framework\ASP.NET 3.5 Extensions) :



Cette librairie .NET est constituée de plusieurs classes dont **WebDataContext** et **WebDataQuery**.

WebDataContext représente le contexte d'exécution pour un service de données spécifié. C'est, pour faire l'analogie avec quelque chose que vous connaissez peut-être, l'équivalent du *DataContext* que l'on retrouve dans LINQ To SQL.

WebDataQuery est la classe que nous allons utiliser pour effectuer des requêtes, à travers le Web, sur notre objet de type *WebDataContext*. Pour effectuer la requête, nous allons utiliser la syntaxe dite « ADO.NET Data Service URI », autrement dit la syntaxe que nous avons vu précédemment avec les « *\$orderby* », « *\$filter* », etc...

Afin de représenter, sous forme d'objets .NET, chacune des entités présentes dans notre service de données, nous allons devoir définir des classes qui nous serviront à faire le mapping et grâce auxquelles nous pourrons manipuler le résultat de nos requêtes sur la source de données distante.

Pour faire cela, nous avons 2 possibilités :

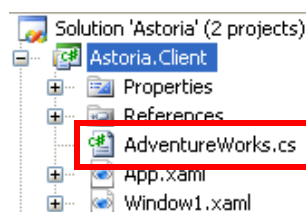
- Le faire à la main et produire un code similaire à celui que vous allez voir juste en dessous. Bien que fonctionnelle, cette technique peut cependant s'avérer longue et fastidieuse si vous avez beaucoup d'entités...

```
public class Product
{
    public string ProductID { get; set; }
    public string Name { get; set; }
    public string Color { get; set; }
}
```

- Utiliser l'outil **webdatagen.exe** (disponible dans le répertoire « *C:\Program Files\Microsoft ASP.NET 3.5 Extensions* »). Celui-ci se comporte comme l'outil « *sqlmetal.exe* » que l'on retrouve avec LINQ To SQL à savoir, il permet la génération du code .NET représentant vos entités. Son utilisation est simple :

```
Webdatagen.exe /uri:http://localhost:2012/AdventureWorks.svc/
/outobjectlayer:AdventureWorks.cs /mode:ClientClassGeneration
```

Uri est utilisé pour indiquer l'adresse du service de données et **outobjectlayer** correspond au paramètre permettant d'indiquer dans quel fichier les classes doivent-être créées. Une fois la génération terminée, vous obtenez un fichier qu'il ne vous reste plus qu'à ajouter à votre projet :



Il ne nous reste plus qu'à utiliser ce fichier pour lancer nos requêtes sur notre source de données distante. Nous allons commencer par créer l'interface de notre application :

```
<Window x:Class="Astoria.Client.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300"
    Loaded="Window_Loaded">
    <Window.Resources>
        <DataTemplate x:Key="ProductItemTemplate">
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="Product Name: " />
                <TextBlock Text="{Binding Name}" />
            </StackPanel>
        </DataTemplate>
    </Window.Resources>

    <Grid>
        <ListView x:Name="lv"
            ItemsSource="{Binding}"
            ItemTemplate="{StaticResource ProductItemTemplate}" />
    </Grid>
</Window>
```

Puis, dans le code, faisons appel à notre source de données :

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    WebDataContext dc = new WebDataContext("http://localhost:2012/AdventureWorks.svc");

    WebDataQuery<Product> products = dc.CreateQuery<Product>("/Product");

    this.lv.DataContext = products;
}
```

Nous commençons donc par créer notre *WebDataContext* puis, au moyen de la méthode **CreateQuery**, nous effectuons une requête sur la source de données.

Un peu plus loin avec ADO.NET Data Services

En plus de la méthode *CreateQuery*, vous avez la possibilité d'utiliser **LINQ** pour requêter votre source de données.

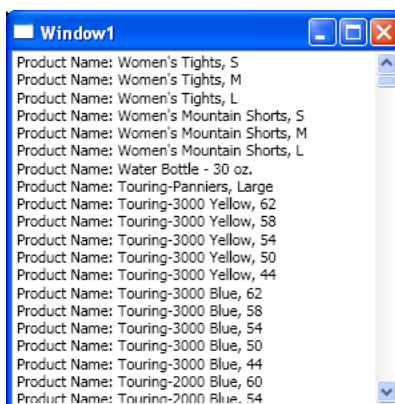
Ainsi, la requête précédente peut tout à fait s'écrire de cette façon :

```
AdventureWorksEntities ctx =  
    new AdventureWorksEntities("http://localhost:2012/AdventureWorks.svc");  
  
var query = from p in ctx.Product  
            select p;  
  
this.lv.DataContext = query;
```

Si vous souhaitez faire un tri, rien de plus simple : il vous suffit d'écrire la requête LINQ correspondante :

```
AdventureWorksEntities ctx =  
    new AdventureWorksEntities("http://localhost:2012/AdventureWorks.svc");  
  
var query = from p in ctx.Product  
            orderby p.Name descending  
            select p;  
  
this.lv.DataContext = query;
```

Le résultat est sans appel et correspond bien à ce que l'on attend. Pour le vérifier, il vous suffit de lancer l'application :



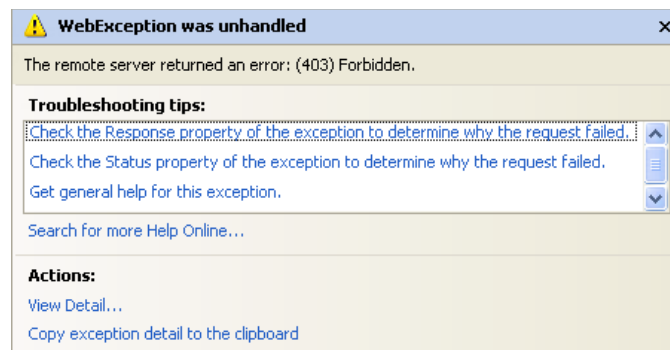
Pour mettre à jour une entité, là encore, le code est extrêmement simple :

```
AdventureWorksEntities ctx =  
    new AdventureWorksEntities("http://localhost:2012/AdventureWorks.svc");  
ctx.MergeOption = MergeOption.AppendOnly;  
  
Product firstProduct = (from p in ctx.Product  
                        select p).First();  
  
firstProduct.Name += " !!!!!";  
  
ctx.UpdateObject(firstProduct);  
ctx.SaveChanges();
```

Détaillons un petit peu ce code :

- Tout d'abord, on se connecte à notre source de données distante
- Ensuite, on indique quel sera le mode de fusion, en ce qui concerne les données
- Puis, on récupère le premier produit de la liste
- On change son nom
- On met à jour l'objet dans la base (UpdateObject)
- Et on valide les modifications (SaveChanges)

Simple non ?! Mais si vous exécutez ce code, vous risquez d'être déçu car vous rencontrerez l'exception suivante :



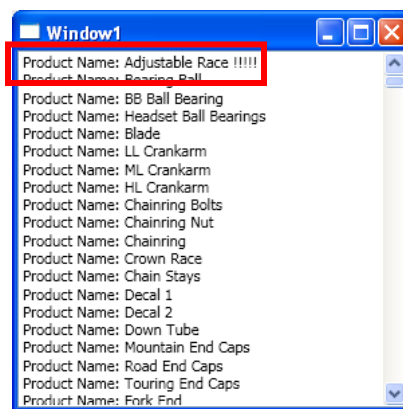
L'erreur est troublante au premier abord mais tout à fait logique si l'on y réfléchit bien. En effet, souvenez-vous des autorisations que nous avons appliquées aux différentes entités de notre source de données :


```
config.SetResourceContainerAccessRule("*", ResourceContainerRights.AllRead);
```

Certes, nous pouvions tout voir mais nous n'avions qu'un accès en lecture ! Pour pouvoir modifier une entité, il nous faut l'autorisation. Nous allons donc la demander :

```
config.SetResourceContainerAccessRule("*", ResourceContainerRights.All);
```

A partir de là, il suffit simplement de ré-exécuter l'application, sans toucher à son code source :



En ce qui concerne l'ajout ou la suppression d'instance d'une entité, là encore, il ne faut pas faire grand chose mis à part appeler la méthode **DeleteObject** ou **AddObject** !

En termes d'authentification, vous devez savoir que la librairie .NET qui est utilisée se base sur le support du Framework .NET pour le protocole HTTP. Autrement dit, cela signifie que certaines propriétés du Framework .NET, pour ce protocole, sont utilisables directement. C'est par exemple le cas de la propriété « *Credentials* », qui vous permet d'indiquer les informations de connexion à utiliser pour effectuer la requête (il faut pour cela utiliser un objet de type **ICredentials**).

Toutes les opérations que nous avons effectuées jusqu'à maintenant fonctionnaient parfaitement mais avait un petit problème : les requêtes n'étaient pas asynchrones ce qui gelait l'interface graphique le temps que les données se chargent.

Heureusement, il est possible de « contourner » ce problème en faisant des requêtes asynchrones, comme le montre cet exemple :

```
private delegate void LoadProductsDelegateEventHandler(IEnumerable<Product> products);

private void LoadProducts(IEnumerable<Product> products)
{
    this.DataContext = products;
}

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    AdventureWorksEntities ctx =
        new AdventureWorksEntities("http://localhost:2012/AdventureWorks.svc");

    WebDataQuery<Product> query = ctx.Product;

    query.BeginExecute(delegate(IAsyncResult iar)
    {
        LoadProductsDelegateEventHandler productsDelegate = new LoadProductsDelegateEventHandler(LoadProducts);

        this.Dispatcher.BeginInvoke(System.Windows.Threading.DispatcherPriority.Normal, productsDelegate, query.EndExecute(iar));
    },
        null);
}
```

Si vous exécutez ce code, vous ne verrez, a priori, aucune différence : les données seront affichées dans l'application.

Cependant, l'interface graphique n'est pas figée/bloquée durant l'exécution : il vous est donc possible d'effectuer d'autres actions lors du chargement des données.

Nous avons vu qu'ADO.NET Data Services nous permettait d'avoir accès à toutes les entités que l'on voulait. Cependant, il est possible d'aller plus loin et d'implémenter vos propres opérations dans le service de données. Pour cela, retournez dans le code de votre service et ajoutez la méthode suivante :

```
[WebGet]
public IQueryable<AdventureWorksModel.Product> SelectAllProducts()
{
    var products = from p in this.CurrentDataSource.Product
                   select p;

    return products;
}
```

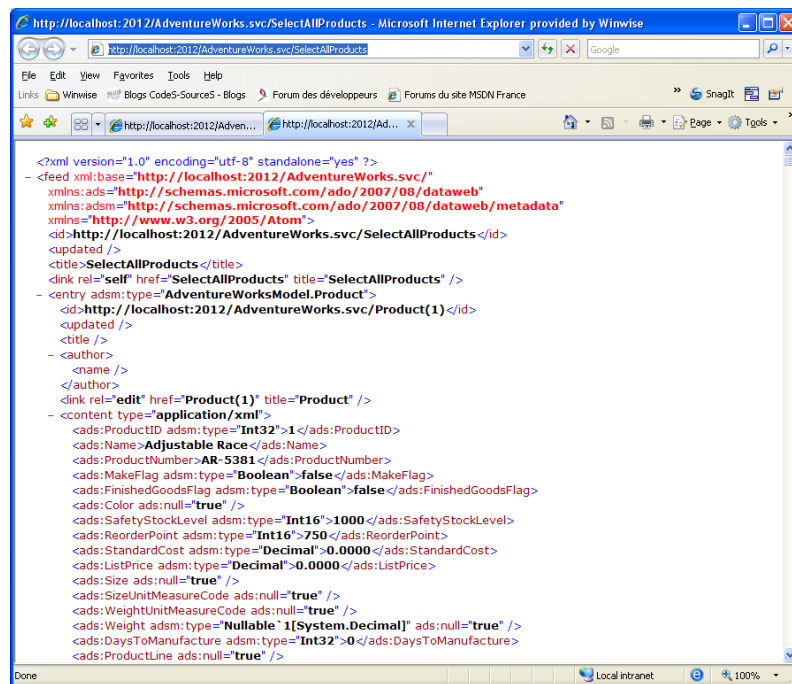
Cette méthode ne fait que sélectionner tous les éléments de la table Product de notre source de données actuelle. Afin de pouvoir faire en sorte qu'elle soit accessible en tant qu'opération, il faut lui imposer l'attribut **WebGet**. De plus, il vous faut modifier le code de la méthode *InitializeService* pour,

comme pour les entités, dire à quelle opération l'utilisateur peut avoir accès. Cela se fait au moyen de cette ligne :

```
config.SetServiceOperationAccessRule("*", ServiceOperationRights.All);
```

Là encore, il est possible de paramétrer plus finement les niveaux d'autorisations accordées à chaque opération.

Une fois cette étape terminée, vous pouvez dès lors utiliser cette opération dans votre navigateur Web :



Ou bien au sein de votre application :

```
WebDataContext dc = new WebDataContext("http://localhost:2012/AdventureWorks.svc");  
  
WebDataQuery<Product> products = dc.CreateQuery<Product>("/SelectAllProducts");  
  
this.lv.DataContext = products;
```

Conclusions

Au cours de cet article, nous avons couvert quelques une des fonctionnalités offertes par ADO.NET Data Services. Certes, le produit est encore en phase de développement mais les fonctionnalités que l'on peut déjà utilisées d'avèrent très prometteuses.

On espère donc beaucoup de ce projet qui, je pense, fera des heureux.... 😊