



Dotnet France  
Technologies Sharepoint, SQL Server & .NET

Association Dotnet France

# Consommation de services de données ADO .NET

*Version 1.0*

## Sommaire

---

1. Introduction.....	3
2. La consommation d'un service Ado.Net Data Services.....	4
2.1 Démarrage du projet.....	4
2.2 Consommation .....	5
2.2.1 Organisation des données.....	5
2.2.2 Consommation de manière Statique.....	7
2.2.3 Consommation dynamique .....	9
3. Conclusion .....	14

## 1. Introduction

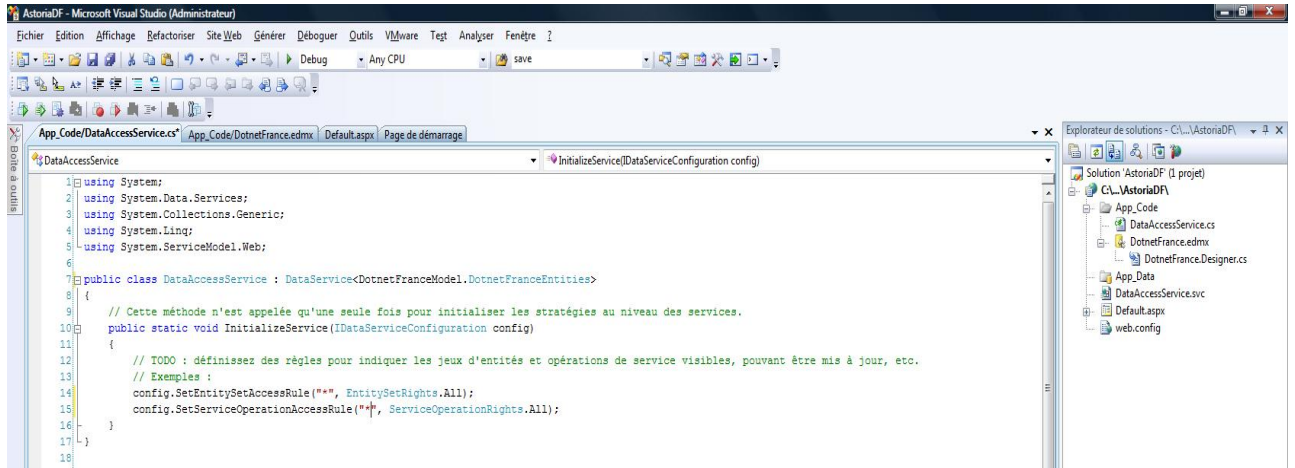
Dans cette partie nous allons voir comment consommer une application ADO.net Data Services dans un client riche. Pour ce faire, nous verrons son utilisation asynchrone dans une application Silverlight puis nous l'implémenterons dans une application winforms plus classique.

## 2. La consommation d'un service Ado.Net Data Services

### 2.1 Démarrage du projet

Nous allons rapidement passer sur la création d'un service de données que l'on a vu dans les deux précédents chapitres.

Une fois que vous êtes sur un écran de ce type avec votre service de prêt, nous pouvons directement attaquer la partie Silverlight.



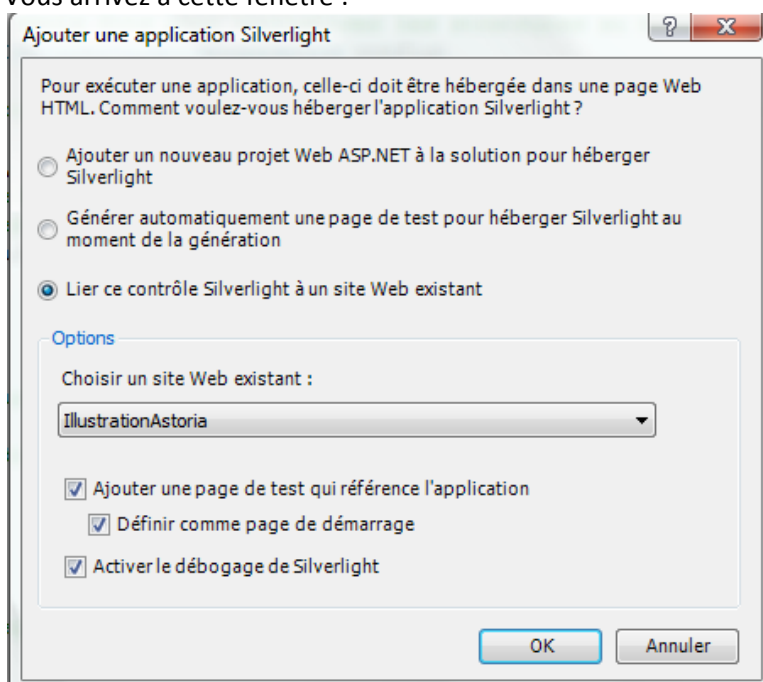
Nous allons voir comment consommer un service ADODS de deux manières :

- Statique. En utilisant des `textblock` et des boutons
- Dynamique. En utilisant un `DataGridView`.

Commençons par créer le projet Silverlight.

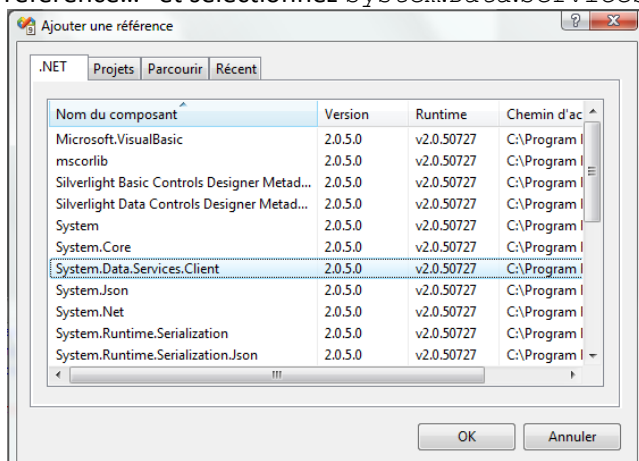
Clic droit sur notre solution puis "ajouter un nouveau projet", et sélectionnez "Application Silverlight".

Vous arrivez à cette fenêtre :



Cliquez sur OK.

Faites ensuite un clic droit sur votre nouveau projet SilverLight puis cliquez sur "Ajouter une référence..." et sélectionnez System.Data.Services.Client.



Pour finir faites un clic droit sur votre projet Silverlight dans l'explorateur de fichier puis sur "ajouter" et un "Elément existant..." et récupérez votre classe Proxy.cs dans son dossier d'enregistrement.

Ajoutez aussi une référence de service en faisant un clic droit sur le projet Silverlight et entrez dans la barre d'adresse, l'adresse de votre service. Dans mon cas : <http://57905:localhost/Service1.svc>.

N'oubliez pas de faire les "using" nécessaire en haut de votre page.xaml.cs. Notamment :

```
//C#
using System.Data.Services.Client;
using "nomDeVotreProjetSilverlight"."NomDeVotreService";
using "nomDeVotreProjetSilverlight" ;
```

```
'VB.Net
Imports System.Data.Services.Client
Imports "nomDeVotreProjetSilverlight"."NomDeVotreService"
Imports "nomDeVotreProjetSilverlight"
```

## 2.2 Consommation

### 2.2.1 Organisation des données

Pour commencer, nous allons organiser nos données dans un objet de type Dictionary que nous pourrons ensuite parcourir.

```
//C#
public Page ()
{
    InitializeComponent();
    VotreSourceDeDonnees entities = new VotreSourceDeDonnees (new
        Uri("votreService.svc",
        UriKind.Relative));

    var query = (from u in entities.VotreTable select u);
    DataServiceQuery<VotreTable> userQuery =
        (DataServiceQuery<VotreTable>) query;

    userQuery.BeginExecute(new AsyncCallback(OnLoadComplete),
        query);
}
```

```
'VB
Public Sub New()
    InitializeComponent()
    Dim entities As New VotreSourceDeDonnees(New Uri("votreService.svc",
UriKind.Relative))

    Dim query = (From u In entities.VotreTable _
Select u)
    Dim userQuery As DataServiceQuery(Of VotreTable) = DirectCast(query,
DataServiceQuery(Of VotreTable))

    userQuery.BeginExecute(New AsyncCallback(OnLoadComplete), query)
End Sub
```

La première ligne permet l'initialisation de notre composant Silverlight.

La seconde ligne crée notre objet entities à partir de votre source de données et de son uri (ainsi que du type d'uri, relatif ou absolu). La source de donnée de l'exemple était `VotreSourceDeDonnees`.

Ensuite on crée une requête de recherche qui récupère le contenu d'une table.

Et à la ligne en dessous, on stocke cette requête de recherche, dans un objet qui représente une requête dans un service de données.

Finalement on exécute cette requête. Silverlight étant asynchrone, on fait appel à un callback qui nous permet d'attendre que les données soient retournées.

Regardons maintenant le code du callback `OnLoadComplete` :

```
//C#
private void OnLoadComplete(IAsyncResult ar)
{
    DataServiceQuery<VotreTable> query =
        (DataServiceQuery<VotreTable>)ar.AsyncState;
    list = new Dictionary<int, VotreTable>();
    foreach (var p in query.EndExecute(ar).ToList())
    {
        list.Add(p.ID, p);
    }
}
```

```
'VB
Private Sub OnLoadComplete(ByVal ar As IAsyncResult)
    Dim query As DataServiceQuery(Of VotreTable) =
DirectCast(ar.AsyncState, DataServiceQuery(Of VotreTable))
    list = New Dictionary(Of Integer, VotreTable)()
    For Each p In query.EndExecute(ar).ToList()
        list.Add(p.ID, p)
    Next
End Sub
```

Dans la première ligne on récupère l'état de l'opération asynchrone.

Ensuite on crée un objet de type `Dictionary` pour récupérer et stocker les données de notre table. Le type `Dictionary` permet à la différence de la Liste d'avoir les index du `Dictionary` qui correspondent aux ID de la table. (Par exemple, si les ID démarrent à 2, l'élément 1 d'une liste correspondrait à l'ID 2 ce qui entrainerait un décalage).

Puis on stocke simplement avec un `foreach` les valeurs dans le `Dictionary`.

On n'oublie pas de déclarer l'objet `list` de type `Dictionary` en début de classe, et tant qu'on y est, le début de l'index du `Dictionary`.

```
//C#
public partial class Page : UserControl
{
    Dictionary<int, VotreTable> list;
    int index = 1;
    ...
    ...
}
```

```
'VB
Public Class Page
    Inherits UserControl
    Private list As Dictionary(Of Integer, VotreTable)
    Private index As Integer = 1
    ...
    ...
End Class
```

## 2.2.2 Consommation de manière Statique

### 2.2.2.1 Lecture de données

Nous allons maintenant voir comment nous pouvons afficher ces données.

Commencez par créer un bouton et une `TextBlock` dans votre fichier `.xaml`, en mettant le code suivant dans votre `Grid`:

```
<!-- xaml -->
<TextBlock Height="32" Margin="146,113,167,0"
VerticalAlignment="Top" Text="" x:Name="Afficheur" TextWrapping="Wrap"/>
<Button Height="32" HorizontalAlignment="Left"
Margin="121,36,0,0" VerticalAlignment="Top" Width="62" Content="Bouton"
Click="Bouton"/>
```

Retournez ensuite dans votre `Page.xaml.cs` puis créez la fonction adéquate :

```
//C#
void Bouton(object sender, RoutedEventArgs e)
{
    Afficheur.Text = list[index]."donneesQueVousSouhaitez";
    index ++;
}
```

```
'VB
Private Sub Bouton(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Afficheur.Text = list(index).donneesQueVousSouhaitez
    index += 1
End Sub
```

De cette manière, à chaque appui sur ce bouton vous afficherez la donnée suivante de votre base. Il faudrait sécuriser avec des `try/catch` en C# et `Try/Catch` en VB.net pour les cas où les index seront absents. Mais ce n'est qu'un exemple pour illustrer les possibilités.

De même, vous avez la possibilité de faire un bouton précédent qui décrémentera la variable `index`.

### 2.2.2.2 Ajout de données

Voyons maintenant comment ajouter des données à notre base. En reprenant l'exemple d'un ajout de membres.

```
//C#
void Add(object sender, RoutedEventArgs e)
{
    entities.MergeOption =
        System.Data.Services.Client.MergeOption.OverwriteChanges;

    TestAstoriaModel.TableMembre _membres = new
        TestAstoriaModel.TableMembre();

    _membres.password = "Nouveau mdp";
    _membres.nomMembre = "Nouvel utilisateur";
    _membres.mail = "mail@domaine.extension";

    entities.AddObject("TableMembre", _membres);
    MessageBox.Show("Utilisateur ajouté");

    entities.BeginSaveChanges(OnSaveChangesCompleted, _membres);
}

void OnSaveChangesCompleted(IAsyncResult result)
{
    entities.EndSaveChanges(result);
}
```

```
'VB
Private Sub Add(ByVal sender As Object, ByVal e As RoutedEventArgs)
    entities.MergeOption =
        System.Data.Services.Client.MergeOption.OverwriteChanges

    Dim _membres As New TestAstoriaModel.TableMembre()

    _membres.password = "Nouveau mdp"
    _membres.nomMembre = "Nouvel utilisateur"
    _membres.mail = "mail@domaine.extension"

    entities.AddObject("TableMembre", _membres)
    MessageBox.Show("Utilisateur ajouté")

    entities.BeginSaveChanges(OnSaveChangesCompleted, _membres)
End Sub
```

La première ligne s'occupe de gérer les droits de modification.

Ensuite on crée une instance de `TableMembre`.

On met les données que l'on souhaite. On pourra toujours les récupérer depuis des `textBox`, ou même un `grid`.

L'objet `entities` représente la source de donnée, il est déclaré en début de classe :

```
TestAstoriaEntities entities = new TestAstoriaEntities(new
    uri("Service1.svc", UriKind.Relative));
```

```
Dim entities As New TestAstoriaEntities(New uri("Service1.svc",
    UriKind.Relative))
```

On appelle la méthode `AddObject` qui prend en paramètre la table que l'on souhaite modifier et l'objet instancié précédemment.

Puis la fonction `BeginSaveChanges` qui appellera le callback `OnSaveChangesCompleted` avec en argument l'objet instancié que l'on souhaite enregistrer.

### 2.2.3 Consommation dynamique

Pour la mise à jour et la suppression de données, il est plus intéressant d'utiliser un datagrid pour afficher, sélectionner et modifier ces données.

Voici le code de la partie xaml :

```

<!-- XAML -->
<UserControl x:Class="gridbox.Page"
  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="683" Height="300">
  <Grid x:Name="LayoutRoot" Background="White">
    <Grid.RowDefinitions>
      <RowDefinition
        Height="8*" />
      <RowDefinition />
      <RowDefinition />
      <RowDefinition />
    </Grid.RowDefinitions>
    <data:DataGrid
      x:Name="dataGrid"
      Margin="10"
      AutoGenerateColumns="True"
      ItemsSource="{Binding}" />

    <Button Margin="145,0,0,0" Grid.Row="1" Content="Supprimer"
      Click="Supprimer" Width="98" HorizontalAlignment="Left"/>
    <Button Margin="10,-1,0,0" Grid.Row="1" Content="Mise à jour"
      Click="Maj" Width="98" HorizontalAlignment="Left"/>
  </Grid>
</UserControl>

```

Et le code de génération du datagrid :

```

//C#
namespace gridbox
{
  public partial class Page : UserControl
  {
    TestAstoriaEntities entities = new TestAstoriaEntities(new
      Uri("Service.svc", UriKind.Relative));
    List<TableMembre> list;

    public Page()
    {
      InitializeComponent();
      list = new List<TableMembre>();
      Generer();
    }

    void Generer()
    {
      var query = (from u in entities.TableMembre select u);
      DataServiceQuery<TableMembre> userQuery =
        (DataServiceQuery<TableMembre>)query;
      userQuery.BeginExecute(new AsyncCallback(OnLoadComplete),
        query);
    }
  }
}
//Suite page suivante

```

```
//Suite de la page précédente

void OnLoadComplete (IAsyncResult ar)
{
    try
    {
        DataServiceQuery<TableMembre> query =
            (DataServiceQuery<TableMembre>) ar.AsyncState;
        dataGrid.Columns.Clear();
        list.Clear();
        foreach (TableMembre p in query.EndExecute(ar).ToList())
        {
            list.Add(p);
        }
        dataGrid.ItemsSource = null;
        dataGrid.ItemsSource = list;
    }

    catch (Exception ex)
    {
        MessageBox.Show (ErreurOnLoadComplete + " " +
            ex.ToString());
    }
}
}
```

```
'VB
Namespace gridbox
    Partial Public Class Page
        Inherits UserControl
        Private entities As New TestAstoriaEntities(New
            Uri("Service.svc", UriKind.Relative))

        Private list As List(Of TableMembre)

        Public Sub New()
            InitializeComponent()
            list = New List(Of TableMembre) ()
            Generer()
        End Sub

        Private Sub Generer()
            Dim query = (From u In entities.TableMembre _
                Select u)
            Dim userQuery As DataServiceQuery(Of TableMembre) =
                DirectCast(query, DataServiceQuery(Of TableMembre))
            userQuery.BeginExecute(New AsyncCallback(OnLoadComplete),
                query)
        End Sub
    End Class
End Namespace
'Suite Page Suivante
```

```
'Suite de la page précédente
Private Sub OnLoadComplete(ByVal ar As IAsyncResult)
    Try
        Dim query As DataServiceQuery(Of TableMembre) =
            DirectCast(ar.AsyncState, DataServiceQuery(Of TableMembre))
        dataGrid.Columns.Clear()
        list.Clear()
        For Each p As TableMembre In
            Query.EndExecute(ar).ToList()
                list.Add(p)
        Next
        dataGrid.ItemsSource = Nothing
        dataGrid.ItemsSource = list
    Catch y As Exception
        MessageBox.Show("ErreurOnLoadComplete: " & y.ToString())
    End Try
End Sub
End Class
End Namespace
```

On retrouve la méthode `Generer()` qui s'occupe de récupérer les données de la table dans une liste en appelant le callback `OnLoadComplete`.

Les nouveautés viennent de :

- La ligne `"list.Clear()"` qui s'occupe de purger la liste avant de recharger des données dedans.
- Des trois lignes commençant par `dataGrid`. La première efface toutes les colonnes, la seconde met (ou remet) à null la source de données du `dataGrid`, et la dernière recharge les données dans le `dataGrid`.

### 2.2.3.1 Mise à jour de données

Voici maintenant la méthode mise à jour appelée lors de l'appui sur le bouton "Mise à jour".

```
//C#
void Maj(object sender, RoutedEventArgs e)
{
    if (dataGrid.SelectedItem == null ||
        dataGrid.SelectedItems.Count > 1)
    {
        MessageBox.Show("Selectionnez une et une seule ligne à
            mettre à jour.");
        return;
    }
    TableMembre ligneSelect =
        (TableMembre) dataGrid.SelectedItem;
    try
    {
        entities.UpdateObject(ligneSelect);
        MessageBox.Show("Mise à jour en cours...");
        entities.BeginSaveChanges(OnSaveChangesCompleted,
            ligneSelect);
    }
    catch (DataServiceRequestException ex)
    {
        MessageBox.Show("ErreurMaj : " +
            ex.Response.ToString());
    }
}
```

```
'VB
    Private Sub Maj(ByVal sender As Object, ByVal e As
        RoutedEventArgs)
        If dataGridView.SelectedItem Is Nothing OrElse
            dataGridView.SelectedItems.Count > 1 Then
            MessageBox.Show("Selectionnez une et une seule ligne à
                mettre à jour.")

            Exit Sub
        End If
        Dim ligneSelect As TableMembre =
            DirectCast(dataGridView.SelectedItem, TableMembre)

        Try
            entities.UpdateObject(ligneSelect)
            MessageBox.Show("Mise à jour en cours...")
            entities.BeginSaveChanges(OnSaveChangesCompleted,
                ligneSelect)

        Catch ex As DataServiceRequestException
            MessageBox.Show("ErreurMaj : " & ex.ToString())
        End Try
    End Sub
```

```
//C#
void OnSaveChangesCompleted(IAsyncResult result)
{
    try
    {
        entities.EndSaveChanges(result);
        MessageBox.Show("Données mise à jour !");
    }
    catch (DataServiceRequestException ex)
    {
        MessageBox.Show("ErreurOnSaveChangesCompleted: " +
            ex.Response.ToString());
    }
    Generer();
}
```

```
'VB
    Private Sub OnSaveChangesCompleted(ByVal result As IAsyncResult)
        Try
            entities.EndSaveChanges(result)
            MessageBox.Show("Données mise à jour !")
        Catch ex As DataServiceRequestException
            MessageBox.Show("ErreurOnSaveChangesCompleted: " &
                ex.Response.ToString())
        End Try
        Generer()
    End Sub
```

On vérifie en premier lieu qu'une et une seule ligne du datagrid soit sélectionnée.

On récupère ensuite la ligne sélectionnée pour pouvoir mettre à jour les données correspondantes.

On met donc à jour les données de l'objet grâce à la méthode `UpdateObject`, puis on appelle un callback pour mettre à jour les données dans la base.

A la fin de ce callback, on rappelle la fonction `Generer()` qui s'occupera de réafficher le datagrid mis à jour.

### 2.2.3.2 Suppression de données

La fonction `Supprimer()` est très similaire à la fonction `Maj()` :

```
//C#
void Supprimer(object sender, RoutedEventArgs e)
{
    if (dataGridView.SelectedItem == null ||
        dataGridView.SelectedItems.Count > 1)
    {
        MessageBox.Show("Selectionnez une et une seule ligne à
                        supprimer.");
        return;
    }
    try{
        TableMembre selectedCategory =
            (TableMembre) dataGridView.SelectedItem;

        entities.DeleteObject(selectedCategory);
        entities.BeginSaveChanges (OnSaveChangesCompleted,
                                   selectedCategory);
    }
    catch (Exception ex) {
        MessageBox.Show("ErreurSupprimer: " + ex.ToString());
    }
}
```

```
'VB
Private Sub Supprimer(ByVal sender As Object, ByVal e As
                    RoutedEventArgs)
    If dataGridView.SelectedItem Is Nothing OrElse
        dataGridView.SelectedItems.Count > 1 Then
        MessageBox.Show("Select a single row for update.")
        Exit Sub
    End If
    Try
        Dim selectedCategory As TableMembre =
            DirectCast(dataGridView.SelectedItem, TableMembre)
        entities.DeleteObject(selectedCategory)
        entities.BeginSaveChanges (OnSaveChangesCompleted,
                                   selectedCategory)
    Catch ex As Exception
        MessageBox.Show("ErreurSupprimer: " & ex.ToString())
    End Try
End Sub
```

On retrouve donc la récupération de la ligne sélectionnée.

La différence est dans l'utilisation de la méthode `DeleteObject` à la place de `UpdateObject`.

### 2.2.3.3 Ajouter des données au DataGridView

Pour ajouter des données au `DataGridView`, il suffit de reprendre le même système que la méthode statique, à savoir des `TextBox`.

La méthode `OnSaveChangesCompleted` se chargeant de recharger le `DataGridView`.

### 3. Conclusion

Nous avons donc vu dans ce chapitre comment gérer de plusieurs manières l'affichage et la gestion de données selon le principe du CRUD dans une application Silverlight.

Vous pouvez maintenant gérer l'affichage de votre service ADODS dans un `DataGrid` mais aussi permettre son parcours de manière linéaires grâce à des boutons Suivant/précédent.