



# Introduction à ADO.NET

---

## Sommaire

Introduction à ADO.NET .....	1
1 Introduction.....	2
1.1 Qu'est ce que l'ADO.NET ? .....	2
1.2 Les nouveautés d'ADO.NET .....	2
1.3 Les avantages d'ADO.NET.....	2
1.4 Les inconvenants d'ADO.NET .....	3
2 Les fournisseurs.....	4
3 Exemple d'implémentation .....	6
4 LINQ.....	8
4.1 Qu'est ce que LINQ? .....	8
4.2 Pourquoi LINQ ? .....	8
4.3 Les nouveautés du C# 3.0.....	8
4.3.1 Typage implicite d'une variable locale .....	9
4.3.2 Expressions lambda.....	10
5 Conclusion .....	11

## 1 Introduction

### 1.1 Qu'est ce que l'ADO.NET ?

ADO.NET (*ActiveX Data Objects.Net*) est un ensemble de classes d'accès aux données compris dans le Framework .NET 3.5. Il permet de gérer de façon simplifiée et organisée des données stockées en base (relationnelle ou non), dans des fichiers XML (*eXtensible Markup Language*) et API (*Application Programming Interface*).

ADO.NET permet au développeur de faciliter la manipulation des données en établissant la liaison entre les données et l'application. Il existe 2 modes de connexion : connecté et déconnecté. Le mode connecté donne un accès en permanence aux données donc une synchronisation quasi immédiate mais demande une bonne prise en charge du réseau. Le mode déconnecté permet de stocker temporairement le résultat de nos requêtes en se connectant juste le temps de la récupération, de les modifier et de mettre à jour la base en se reconnectant.

ADO.NET nous offre 4 fournisseurs pour accéder aux données (SQL Server, Oracle, ODBC et OLE DB ; que nous détaillerons dans le cours). Ceux-ci peuvent permettre de stocker dans un DataSet les résultats de nos requêtes.

### 1.2 Les nouveautés d'ADO.NET

Les grosses nouveautés d'ADO.NET qui vont nous intéresser sont :

- la prise en charge de SQL Server 2008 par le fournisseur SQL Server.
- LINQ qui permet de requêter directement des données de façon simplifiée et optimisée. Notamment avec LINQ to DataSet qui facilitera la manipulation de notre DataSet.
- EDM (Entity Data Model) et l'EntityFramework permettant de mapper nos classes pour les compléter automatiquement avec nos données

Et quelques autres nouveautés que vous pouvez retrouver sur le [msdn](#).

### 1.3 Les avantages d'ADO.NET

- Le plus gros avantage d'ADO.NET est sa simplicité de mise en place grâce à ses 4 fournisseurs offrant un accès quasi direct avec nos données.
- ADO.NET permet de garder en mémoire un DataSet de plusieurs tables alors qu'ADO ne pouvait garder qu'un jeu d'enregistrement ne représentant qu'une table et augmentant les performances.
- L'interopérabilité se fait grâce au transfert des données sur le réseau en XML. Ainsi la seule obligation est la capacité de lire (et/ou parser) le XML.

- Le typage direct d'un membre non typé d'un DataSet. Ainsi :

```
//C#  
if (salaireM > (double)dataSet1.Tables["Employe"].Rows[n]["Salaire"])  
  
'VB  
If salaireM > CType(dataSet1.Tables("Employe").Rows(n)("Salaire"), Double)  
Then
```

Deviendra :

```
//C#  
if (salaireM > dataSet2.Employe[n].Salaire)  
  
'VB  
If salaireM > dataSet2. Employe(n).Salaire Then
```

## 1.4 Les inconvenants d'ADO.NET

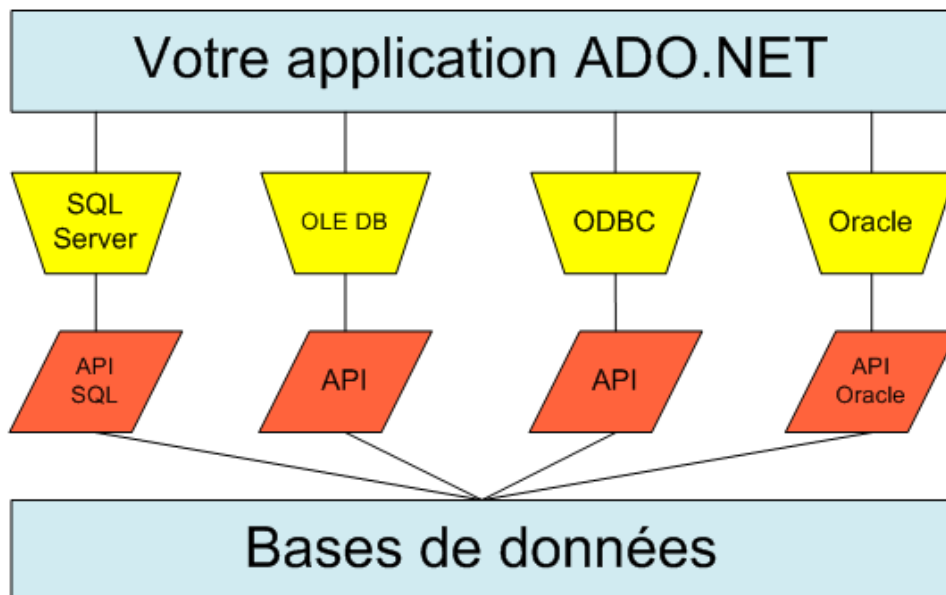
Le plus gros défaut de ADO.NET est le typage d'une requête SQL en string donc sans IntelliSense ni coloration syntaxique ce qui peut devenir contraignant avec de grosses requêtes.

## 2 Les fournisseurs

Comme dit précédemment, il existe 4 fournisseurs qui sont des interfaces entre votre base et votre code. Chaque fournisseur de données permet la communication avec un type de base de données au travers d'une API. Ces fournisseurs permettent de récupérer et de transférer des modifications entre l'application et une base de données. Toutes les classes permettant d'utiliser ces fournisseurs se trouvent dans l'espace de nom *System.Data*.

Sur le Framework 3.5, il existe quatre types de fournisseurs :

- Sql Server
- OLE DB
- ODBC
- Oracle



Chaque fournisseur est relié à une base de données propre, c'est-à-dire qu'il est compatible à l'API de sa base de données. Cependant, 2 sont spécifiques : SQL Server pour les bases SQL Server et Oracle pour les bases Oracle. Les 2 autres sont plutôt génériques car s'adaptent nécessitant cependant une installation logicielle.

Fournisseur	Description
SQL Server	Les classes de ce fournisseur se trouvent dans l'espace de nom <i>System.Data.SqlClient</i> , chaque nom de ces classes est préfixé par <i>Sql</i> . SQL Server à accès au serveur sans utiliser d'autres couches logicielles le rendant plus performant.
OLE DB	Les classes de ce fournisseur se trouvent dans l'espace de nom <i>System.Data.OleDb</i> , chaque nom de ces classes est préfixé par <i>OleDb</i> . Ce fournisseur exige l'installation de MDAC (Microsoft Data Access Components). L'avantage de ce fournisseur est qu'il peut dialoguer avec n'importe quelle base de données le temps que le pilote OLE DB est installé ; Mais par rapport à SQL server, OLE DB utilise une couche logicielle nommée OLE DB. Il requiert donc plus de ressources diminuant par conséquent les performances.
ODBC	Les classes de ce fournisseur se trouvent dans l'espace de nom <i>System.Data.Odbc</i> , chaque nom de ces classes est préfixé par <i>Odbc</i> . Tout comme l'OLE DB, ODBC exige l'installation de MDAC. Il fonctionne avec le même principe qu'OLE DB mais au lieu d'utiliser une couche logicielle, il utilise le pilote ODBC.
Oracle	Les classes de ce fournisseur se trouvent dans l'espace de nom <i>System.Data.OracleClient</i> , chaque nom de ces classes est préfixé par <i>Oracle</i> . Il permet simplement de se connecter à une source de données Oracle.

Remarque : SQL Server et Oracle sont tous deux des fournisseurs de données managés. C'est-à-dire qu'ils sont optimisés pour certains types de bases de données.

### 3 Exemple d'implémentation

Voici un extrait de code qui va permettre de lister le nom et le prénom de tous les stagiaires contenus dans la table Stagiaire de la base dnFrance (celle-ci ne contient que deux enregistrements). Le résultat s'affichera dans la console.

Pour faire fonctionner ce code, vous devrez importer les espaces de noms System.Data et System.Data.SqlClient

```
//C#
// chaine de connexion
string connectionString = "Data Source=(local);Initial
Catalog=Northwind;Integrated Security=SSPI";

// requete SQL
string requete = "SELECT nom, prenom FROM dnFrance.Stagiaire;";

SqlConnection connection = new SqlConnection(connectionString)
SqlCommand command = connection.CreateCommand();
command.CommandText = requete;
try
{
    connection.Open(); // ouverture de la connexion
    SqlDataReader reader = command.ExecuteReader();
    while (reader.Read())
    {
        Console.WriteLine("{0}\t{1}", reader[0], reader["prenom"]);
    }
    reader.Close(); // fermeture de la connexion
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
```

```
`VB
' chaine de connexion (que nous détaillerons dans le chapitre suivant)
Dim connectionString As String = "Data Source=(local);Initial
Catalog=Northwind;Integrated Security=SSPI";

' requete SQL
Dim requete As String = "SELECT nom, prenom FROM dnFrance.Stagiaire;";

' On crée la connexion a une base SQL Server
Using connection As New SqlConnection(connectionString)

    Dim command As SqlCommand = connection.CreateCommand()
    command.CommandText = queryString
    Try
        connection.Open() ' ouverture de la connexion
        Dim dataReader As SqlDataReader = command.ExecuteReader()
        Do While dataReader.Read()
            Console.WriteLine("{0}" & vbTab & "{1}", _
                dataReader(0), dataReader(1))
        Loop
        dataReader.Close()' fermeture de la connexion
    Catch ex As Exception
        Console.WriteLine(ex.Message)
    End Try
End Using
```



Après compilation, vous devriez obtenir quelque chose comme ceci :

```
C:\Windows\system32\cmd.exe
PALUDETTO      Damien
RON SIN Cedric
Appuyez sur une touche pour continuer... _
```

Dotnet-France Association

## 4 LINQ

### 4.1 Qu'est ce que LINQ?

Le LINQ, Language Integrated Query, est un modèle de programmation permettant d'établir des requêtes sur des données dans n'importe quel langage Microsoft .NET. Il permet au développeur d'accéder à des sources de données quelconques de manière unique et simplifiée. En effet, à aucun moment il n'y aurait à entrer du code SQL par exemple. Ce langage est une nouveauté du Framework 3.5 et profite des améliorations du C#3.0.

### 4.2 Pourquoi LINQ ?

Il existe de nos jours plusieurs sources de données : base de données, fichier Microsoft Access, fichier Microsoft Excel, fichier XML, etc., chacun possédant une manière spécifique d'accès aux données. Par exemple, on utilise très souvent du SQL pour accéder à une base de données ou encore du « Document Object Model » (DOM) pour le XML.

C'est ici qu'intervient LINQ. Pour faciliter la manipulation de données à travers un programme, vous ne devrez connaître plus qu'un langage : le LINQ. A travers ses différentes extensions (que nous aborderons dans différents chapitres), il permet un accès universel à toute source de données facilitant ainsi la tâche des développeurs.

Voici en résumé une petite énumération non exhaustive des objectifs de LINQ.

- Faciliter les recherches dans toutes les données.
- Pouvoir bénéficier de l'IntelliSense et de la coloration de Visual Studio.
- Eviter la connaissance d'une multitude de langages.
- Une syntaxe uniformisée quelque soit la source de données (à part pour ce qui est du XML qui garde une structure relativement spécifique).

### 4.3 Les nouveautés du C# 3.0

Pour tirer pleinement partie du LINQ, il est important de connaître quelques nouveautés du C#3.0. Je vais vous expliquer celles que nous allons utiliser durant ce tutoriel. (Les développeurs VB.Net peuvent passer cette partie, vu qu'ils ont des équivalents de longues dates...).

### 4.3.1 Typage implicite d'une variable locale

Avec l'apparition de la version 3.0 du C#, un nouveau mot-clé a été ajouté : il s'agit de « var ». Ce mot-clé permet de ne pas définir explicitement le type d'une variable dans le code et de laisser cette tâche au compilateur qui déterminera le type de celle-ci en fonction du contexte.

```
C#  
  
var texte = "texte";  
var objet = new {  
    texte = "texte",  
    numero = 12,  
    collection = new Dictionary<int, string>()  
};  
objet.collection.Add(1, texte);  
Console.WriteLine(objet.collection[1]);
```

Dans cet exemple, « var » permet de créer un objet (nom de la variable : « objet ») de classe anonyme. L'objet de classe anonyme a des attributs typés automatiquement. Ils sont définis entre les accolades en une série d'égalité (à gauche les attributs, à droite leurs valeurs) séparés par des virgules.

Les objets de classe anonyme ne doivent pas être utilisés comme retour de méthode car c'est des objets qui n'ont pas vraiment de classes définies, c'est donc principalement un moyen de ranger des variables de manière temporaire.

« var » m'a aussi permis d'instancier un « String » sans avoir à connaître le type. Lorsque l'on instancie avec « var », pour que le compilateur sache le type à mettre à l'objet, on doit impérativement lui associer un contenu typé. Si on essaie d'assigner une valeur « null » à une variable déclarée avec le mot-clé « var », une erreur est générée :

```
C#  
  
var variable = null;
```

Comme vous pouvez le voir dans l'extrait de code ci-dessus, Visual Studio avertit qu'il y a une erreur dans le code. Si vous pointez le mot « variable » où se trouve l'erreur, Visual Studio vous annonce qu'il est « Impossible d'assigner '<null>' à une variable locale implicitement typée ».

Enfin s'il y a un point important à retenir sur l'usage du mot clé « var » c'est que les variables instanciées avec celui-ci doivent avoir une valeur sur la même instruction pour permettre au compilateur d'assigner un type explicite à la variable.



### 4.3.2 Expressions lambda

Les expressions lambda sont un moyen simple de faire des méthodes anonymes avec une syntaxe plus rapide et plus claire. La syntaxe des expressions lambda est simple et assez lisible. Ci-dessous un exemple d'expression lambda :

```
C#  
  
(int i)=> { return i++; }
```

Dans cette expression, on peut distinguer 3 parties. La première « (int i) » est la partie des variables passées en paramètre à la fonction. Ici, la fonction attend une variable de type « int ». La seconde partie, est l'opérateur « => » qui permet d'introduire le corps de la fonction lambda. Enfin, la partie « { return i++; } » est ledit corps de la fonction. Ici, elle ne ferait qu'incrémenter « i » puis le retourner.

***Remarque :** Cette méthode d'expression lambda ne peut pas s'utiliser seule. Pour plus de détails, vous pourrez vous reporter au cours contenant « LINQ to Objects ». Il y aura quelques exemples de code qui seront expliqués, notamment sur l'utilisation des expressions lambda.*



## 5 Conclusion

Nous avons donc vu que ADO.NET nous offrait un panel de nouveautés qui nous faciliterons la tâche.

Dans les chapitres à venir, nous verrons les 2 modes (connecté et déconnecté) car chacun d'eux a ses avantages et inconvénients.

Nous verrons également comment utiliser les fournisseurs et les DataSet pour accéder et modifier le plus rapidement et facilement possible nos données et nous ferons aussi la liaison avec LINQ (Plus d'informations sur MSDN : <http://msdn.microsoft.com/fr-fr/library/system.linq.aspx>) qui facilitera la manipulation de données dans notre DataSet.

Dotnet-France Association