

# ADO.NET : les objets Connection, Command et Datareader en C#

Par [leduke](#)

Date de publication : 1 juin 2003

Cet article a pour but de présenter les objets Connection, Command et Datareader dans ADO.NET

I - Que nous apporte ADO.Net ?.....	3
II - Choix d'un fournisseur de données .NET.....	3
III - Le modèle objet ADO.Net.....	4
IV - L'objet Connection.....	4
V - L'objet Command.....	5
VI - L'objet DataReader.....	6
VII - Comment appeler une procédure stockée ?.....	8
VIII - Conclusion.....	10

## I - Que nous apporte ADO.Net ?

Avec ADO.Net Microsoft s'est efforcé de répondre efficacement aux besoins des applications Web en apportant les nouveautés suivantes :

### Une architecture plus optimisée :

Avec .Net, de nouveaux fournisseurs de données voient le jour. Certains fonctionnent en mode natif, supprimant des couches intermédiaires entre l'application et sa source de données. ADO.Net pourra ainsi être plus rapide accédant directement à la source de données. Par exemple le .NET Framework comprend le fournisseur de données SQL Server .NET en mode natif via le namespace **System.Data.SqlClient**.

### Un meilleur support du mode déconnecté :

Le design de ADO.Net répond aux exigences des modèles de développement des applications actuelles. Dans une application Web, le maintien d'une connexion SGBD ouverte doit être la plus courte possible, car le nombre de connexions ouvertes vers une source de données peut constituer un frein à la capacité de montée en charge d'une application. Le modèle ADO dans son temps, avait évolué afin de répondre à ce besoin par un Recordset déconnecté. Aujourd'hui Microsoft encourage le mode déconnecté et propose des classes spécialisées supportant les deux modes : connecté (via l'objet DataReader) et déconnecté (via l'objet DataSet).

### Un meilleur support de XML :

Le XML est utilisé au sein du Framework .NET comme le standard de description et de persistance des données. Dans ADO.Net il est utilisé comme le format de transmission universel de données.

## II - Choix d'un fournisseur de données .NET

Pour pouvoir faire appel aux classes proposées par ADO.Net il est nécessaire d'inclure une référence à l'espace de noms correspondant. Vous pouvez soit inclure l'espace System.Data soit inclure des classes de cet espace comme System.Data.OleDb ou System.Data.SqlClient ; tout dépend de la source de données utilisées.

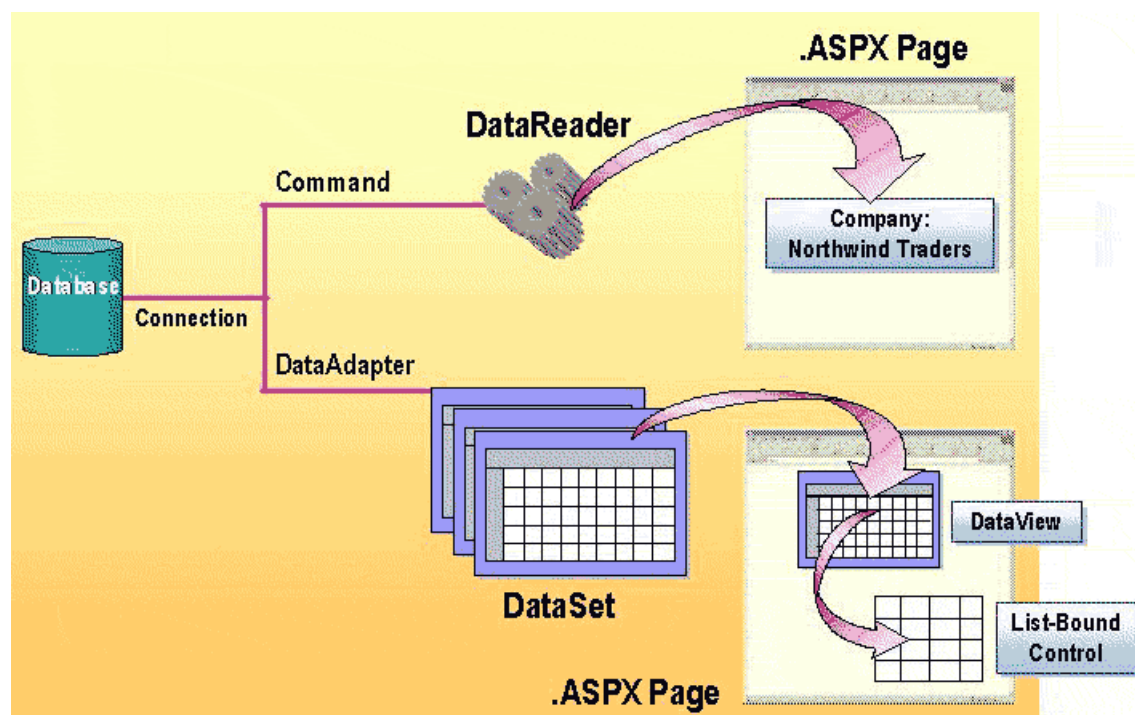
```
//Inclusion d'un namespace  
  
using System.Data;  
using System.Data.SqlClient;
```

Plusieurs espaces de noms sont disponibles avec ADO.Net parmi lesquels :

Espace de nom	Description
System.Data.SqlClient	Fournisseur de données spécifiques pour SQLServer V7 ou supérieure
System.Data.OleDb	Propose la gestion de sources de données accédées via un driver OleDb
System.Data.Odbc	Propose la gestion de sources de données accédées via un driver Odbc
System.Data.OracleClient	Propose un accès à des sources de données Oracle (v8.1.7 ou supérieure)
System.Data.XML	Propose des classes permettant d'accéder à la fonctionnalité XML sous SQL Server 2000 ou supérieur
System.Data.SqlTypes	Propose des classes pour des types de données spécifiques à Microsoft SQL Server

Dans la suite de notre article, nous utiliserons le fournisseur de données SqlClient et accéderons à une base SQLServer 2000 fournie en exemple nommé « Northwind ».

### III - Le modèle objet ADO.Net



ADO.Net comprend quelques objets similaires aux ADO (comme les objets Connection et Command) dont la syntaxe a évolué. L'objet Recordset n'existe plus, il a été remplacé par les objets DataReader et DataSet.

Regardons de plus près chacun de ces objets.

Objet	Description
Connection	Ouvre une connexion vers une source de données spécifiques
Command	Exécute une commande sur une source de données
DataReader	Lit un flux de données à partir d'une source de données en mode <b>connecté</b> . Le mode d'accès est en lecture seule avec un curseur en avant seulement.
DataSet	Représente un ensemble de données en mode <b>déconnecté</b> . Il peut être constitué de plusieurs tables ainsi que des relations et contraintes existant entre elles.
DataAdapter	Remplit un <b>DataSet</b> et répercute les mises à jour dans la source de données.

### IV - L'objet Connection

La connectivité à SQLServer 2000 est assurée par l'objet SqlConnection de l'espace de noms **System.Data.SqlClient**. Le framework .Net propose ainsi des objets de connexion différents en fonction du type de

fournisseur de données choisies. Par exemple vous devrez utiliser l'objet **OleDbConnection** si votre fournisseur est un fournisseur **OleDb**. La connectivité à SQLServer 2000 est assurée par l'objet **SqlConnection** de l'espace de noms **System.Data.SqlClient**. Le framework .Net propose ainsi des objets de connexion différents en fonction du type de fournisseur de données choisies. Par exemple vous devrez utiliser l'objet **OleDbConnection** si votre fournisseur est un fournisseur **OleDb**.

L'ouverture d'une connexion est réalisée par la méthode **Open** et la fermeture par la méthode **Close**.

```
// Exemple de gestion d'une connexion

using System;
using System.Data.SqlClient;
using System.IO;

namespace ExempleAdoNetCSharp
{
    public class SQLConnexion
    {
        public static void Main()
        {
            string strConnexion = "Data Source=localhost; Integrated Security=SSPI;" + "Initial
Catalog=Northwind";
            try
            {
                SqlConnection oConnection = new SqlConnection(strConnexion);
                oConnection.Open();
                Console.WriteLine("Etat de la connexion : " + oConnection.State);
                oConnection.Close();
            }
            catch (Exception e)
            {
                Console.WriteLine("L'erreur suivante a été rencontrée :" + e.Message);
            }
        }
    }
}
```

## V - L'objet Command

Une fois la connexion vers une base de données effectuée, vous pouvez exécuter une requête et récupérer son résultat en utilisant l'objet **Command**. Contrairement au fonctionnement des ADO où il était possible d'exécuter une requête sans utiliser l'objet **Command**, vous devez désormais systématiquement le faire.

La création d'un objet **Command** nécessite l'instanciation d'un objet **SqlCommand**. Cet objet expose différentes méthodes **Execute** à utiliser selon le résultat attendu :

- La méthode **ExecuteReader** peut être utilisée pour récupérer un jeu d'enregistrement et retourne un objet **DataReader** ;
- La méthode **ExecuteScalar** récupère une valeur unitaire ;
- La méthode **ExecuteNonQuery** exécute une commande ne retournant pas de lignes.

```
//Exemple d'utilisation d'un objet Command

using System;
using System.Data.SqlClient;
using System.IO;

namespace ExempleAdoNetCSharp
{
    public class CommandesSQL
    {
        public static void Main()
        {

```

```
string strConnexion = "Data Source=localhost; Integrated Security=SSPI;" + "Initial Catalog=Northwind";
string strRequete = "INSERT INTO Region VALUES (5,'Sud')";
try
{
    SqlConnection oConnection = new SqlConnection(strConnexion);
    SqlCommand oCommand = new SqlCommand(strRequete,oConnection);
    oConnection.Open();
    oCommand.ExecuteNonQuery();
    oConnection.Close();
}
catch (Exception e)
{
    Console.WriteLine("L'erreur suivante a été rencontrée :" + e.Message);
}
}
```

## VI - L'objet DataReader

L'objet DataReader permet de récupérer d'une source de données un flux en lecture seule en avant seulement (read only, forward only). Il résulte de l'exécution de la méthode **ExecuteReader** sur un objet Command.

L'objet DataReader ne stocke en mémoire qu'une seule ligne à la fois, permettant ainsi d'augmenter les performances d'une application et d'en réduire la charge.

Il est recommandé d'utiliser cet objet si :

- Vous n'avez pas besoin de réaliser un cache des données ;
- Vous traitez un jeu d'enregistrements trop importants pour être stocké en mémoire ;
- Vous souhaitez accéder à des données rapidement en lecture seule en avant seulement.

Comme l'objet DataReader a été conçu pour accéder aux données selon un mode connecté, il ne peut être transmis entre différents tiers applicatifs ce que réalisait un Recordset déconnecté.

Par défaut, un DataReader charge une ligne entière en mémoire à chaque appel de la méthode **Read**. Il est possible d'accéder aux valeurs de colonnes soit par leur nom soit par leurs références ordinales. Une solution plus performante est proposée permettant d'accéder aux valeurs dans leurs types de données natives (GetInt32, GetDouble, GetString ...). Par exemple si la première colonne de la ligne indiquée par 0 est de type int, alors il est possible de la récupérer à l'aide de la méthode **GetInt32** de l'objet **DataReader**.

```
int iColonne ;
iColonne = oDataReader.GetInt32(0)
```

La méthode **Close** ferme un objet DataReader. Précisons que si l'objet Command utilisé contient des paramètres en sortie ou des valeurs de retour, ils ne pourront être récupérés qu'à l'issue de la fermeture du DataReader.

Pour augmenter les performances, il est parfois nécessaire de soumettre plusieurs requêtes à la fois. L'objet DataReader répond à ce besoin avec la méthode **NextResult** permettant de passer d'un jeu d'enregistrement à un autre. Pour les habitués des ADO, cette méthode est équivalente au **NextRecordset**.

Après vous avoir présenté les principales méthodes de l'objet DataReader, regardons plus précisément leurs mises en œuvre à l'aide d'un exemple d'extraction de données.

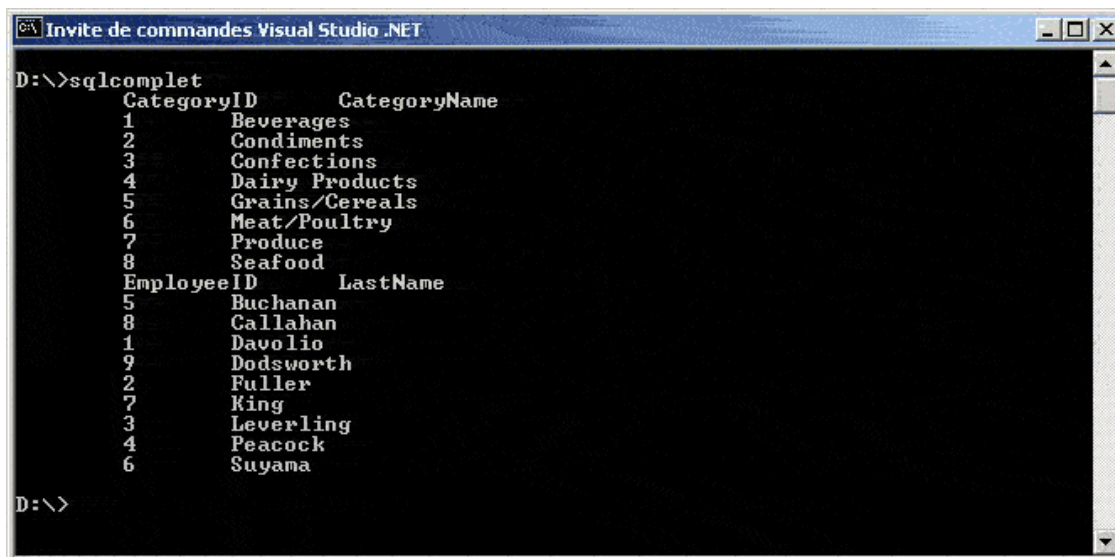
```
// Exemple d'extraction de données avec l'objet DataReader

using System;
using System.Data.SqlClient;
```

```
using System.IO;

namespace ExempleAdoNetCSharp
{
    public class CommandeSQL
    {
        public static void Main()
        {
            string strConnexion = "Data Source=localhost; Integrated Security=SSPI;" + "Initial Catalog=Northwind";
            string strRequete = "SELECT CategoryID, CategoryName FROM Categories;" + "SELECT EmployeeID, LastName FROM Employees";
            try
            {
                SqlConnection oConnection = new SqlConnection(strConnexion);
                SqlCommand oCommand = new SqlCommand(strRequete, oConnection);
                oConnection.Open();
                SqlDataReader oReader = oCommand.ExecuteReader();
                do
                {
                    Console.WriteLine("\t{0}\t{1}", oReader.GetName(0), oReader.GetName(1));
                    while (oReader.Read())
                    {
                        Console.WriteLine("\t{0}\t{1}", oReader.GetInt32(0), oReader.GetString(1));
                    }
                } while (oReader.NextResult());
                oReader.Close();
                oConnection.Close();
            }
            catch (Exception e)
            {
                Console.WriteLine("L'erreur suivante a été rencontrée :" + e.Message);
            }
        }
    }
}
```

Ce qui donne à l'exécution le résultat suivant :



```
D:\>sqlcomplet
CategoryID      CategoryName
1 Beverages
2 Condiments
3 Confections
4 Dairy Products
5 Grains/Cereals
6 Meat/Poultry
7 Produce
8 Seafood
EmployeeID      LastName
5 Buchanan
8 Callahan
1 Davolio
9 Dodsworth
2 Fuller
7 King
3 Leverling
4 Peacock
6 Suyama
D:\>
```

## VII - Comment appeler une procédure stockée ?

Voyons maintenant comment appeler une procédure stockée en utilisant les objets Command et la collection Parameters.

Les procédures stockées sont un ensemble d'ordre SQL compilés sur un moteur SGBD. Grâce aux procédures stockées, il est possible de centraliser l'ensemble du code SQL de votre application à un seul endroit, ce qui en facilite la maintenance. D'autre part, une procédure stockée est plus performante qu'une requête dynamique dont le plan d'exécution et la compilation doivent être effectués par le moteur SGBD avant toute exécution.

L'utilisation de la collection Parameters de l'objet Command permet de définir explicitement les paramètres en entrée et en sortie ainsi que le code retour d'une procédure stockée.

Tout d'abord informons l'objet Command du type de commande à traiter via la propriété **CommandType** ; et donnons-lui la valeur **StoredProcedure**.

```
// Définition du type de commande  
  
oCmd.CommandType = CommandType.StoredProcedure;
```

Définissons ensuite chacun des paramètres en utilisant la méthode **Add** de la collection Parameters (dont l'un des prototypes est décrit ci-dessous).

Description	Type	Exemple
Nom du paramètre	string	@nom
Type de données	SqlDbType	SqlDbType.Char
Taille de la colonne	int	25
Nom de la colonne source	string	nom

```
// Création d'un paramètre en entrée  
  
SqlParameter oParam = oCmd.Parameters.Add ("@nom", SqlDbType.Char, 25, Nom);  
oParam.Value = "John";  
  
// Création d'un paramètre retour  
  
SqlParameter oParam = oCmd.Parameters.Add ("ReturnValue", SqlDbType.Int);  
oParam.ParameterDirection = ParameterDirection.ReturnValue;
```

Précisons :

- Qu'il est nécessaire d'affecter à la propriété Value de l'objet Parameter une valeur pour les paramètres qui le nécessitent ;
- Qu'il faut décrire explicitement le type de paramètres via la propriété **ParameterDirection** de l'objet Parameter (sauf pour les paramètres en entrée). Celle-ci peut prendre l'une des valeurs suivantes : Input, InputOutput, Output ou ReturnValue.

Regardons un exemple implémentant l'exécution de la procédure stockée « GetCustomerOrders » de la base « Northwind ».

```
// Exemple d'appel à une procedure stockée  
  
using System;  
using System.Data.SqlClient;  
using System.Data;  
using System.IO;  
  
namespace ExempleAdoNetCSharp  
{
```



```
public class SQLProcStock
{
    public static void Main()
    {
        string strConnexion = "Data Source=localhost; Integrated Security=SSPI;" + "Initial
Catalog=Northwind";
        string strProcedureStockee = "GetCustomerOrders";
        try
        {
            SqlConnection oConnection = new SqlConnection(strConnexion);
            SqlCommand oCommand = new SqlCommand(strProcedureStockee,oConnection);
            oCommand.CommandType = CommandType.StoredProcedure;
            SqlParameter oParam =
oCommand.Parameters.Add("@CustomerName",SqlDbType.NVarChar,50,"CustomerName");
            oParam.Value = "France restauration";
            oConnection.Open();
            SqlDataReader oReader = oCommand.ExecuteReader();
            Console.WriteLine("\t{0}\t{1}\t{2}\t{3}", oReader.GetName(0),
oReader.GetName(3),oReader.GetName(4), oReader.GetName(5));
            while (oReader.Read())
            {
                Console.WriteLine("\t{0}\t{1}\t{2}\t{3}", oReader.GetInt32(0),
oReader.GetString(3),oReader.GetDecimal(4),oReader.GetInt16(5));
            }
            oReader.Close();
            oConnection.Close();
        }
        catch (Exception e)
        {
            Console.WriteLine("L'erreur suivante a été rencontrée :" + e.Message);
        }
    }
}
```

Une fois le code exécuté, la liste suivante s'affiche :

```
D:\>sqlprocstock
OrderID ProductName      UnitPrice      Quantity
10671    Pavlova 17.45      10
10671    Tarte au sucre 49,3      10
10671    Louisiana Fiery Hot Pepper Sauce 21,05      12
10860    Manjinup Dried Apples 53      3
10860    Lakkalikööri 18      20
10971    Thüringer Rostbratwurst 123,79      14
D:\>_
```

## VIII - Conclusion

Nous avons abordé dans cet article quelques-uns des objets disponibles dans ADO.Net. et plus particulièrement le mode connecté proposé par l'objet DataReader. Cet objet offre une solution performante pour accéder rapidement à des données et pour traiter des données dont la taille est trop importante pour être stockée en mémoire.