**[ADO.NET] Introduction à ADO.NET en C#**

ADO.NET est un ensemble de composants présents de base dans le framework .NET permettant l’accès et la gestion de données situées sur une base de données relationnelle (SQL Server, Oracle, etc…) ou non. ADO.NET est une évolution de [ADO](http://fr.wikipedia.org/wiki/ActiveX_Data_Objects) (ActiveX Data Objects).

Les classes ADO.NET peuvent être divisées en 2 parties. Les classes permettant de se connecter à la source de données et les classes utilisées pour gérer les données.

**La connexion à la source de données**

Pour se connecter à une base de données via ADO.NET un *Data Provider*(fournisseur de données) correspondant à la base de données utilisée. Ainsi le *Data Provider* de SQL Server est optimisé pour fonctionner avec les bases de données SQL Server, idem pour le *Data Provider* Oracle. Tous les *Data Providers* sont dérivés d’une seule et même classe, ils implémentent les mêmes méthodes, propriétés, et fonctionnent donc de la même manière. Dans certains cas, les *Data Providers* peuvent implémenter des fonctionnalités spécifiques à la base de données utilisée (exemple des requêtes XML pour SQL Server).

Les 4 fournisseurs de données inclus dans le framework .NET :

* *SQL Server Provider*: pour SQL Server 7.0 et plus).
* *OLE DB provider* : pour toutes les bases de données disposant d’un pilote [OLE DB](http://msdn.microsoft.com/fr-fr/library/bb469883.aspx) (Object Linking and Embedding).

Note:

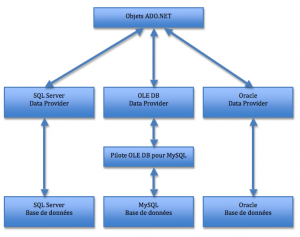
OLE DB est une interface basée sur [COM](http://fr.wikipedia.org/wiki/Component_Object_Model) (Component Object Model) utilisée pendant plusieurs années avec ADO (ActiveX Data Objects), la plupart des bases de données ont donc un pilote OLE DB (MySQL, Access, Oracle SQL Server, etc..). Si la source de données ne propose pas de pilote OLE DB, vous pouvez [le créer vous-même](http://msdn.microsoft.com/en-us/library/yffccxb5(VS.71).aspx).

Comme dit précédemment, OLE DB propose des pilotes pour SQL Server et Oracle qui ont déjà des *Data Providers* dédiés. Il est donc tout à fait possible d’utiliser OLE DB à la place du*SQL Server Provider*. La seconde méthode (*SQL Server Provider*) est préférable car elle offrira de meilleures performances.

* *Oracle provider* : pour Oracle (version 8i et plus).
* *ODBC provider* : pour toutes les sources de données qui ont un pliote [ODBC](http://www.commentcamarche.net/contents/odbc/odbcintro.php3) (Open DataBase Connectivity).

Note :

ODBC est une suite de pilotes permettant la communication avec la pulpart des [SGBD](http://fr.wikipedia.org/wiki/Syst%C3%A8me_de_gestion_de_base_de_donn%C3%A9es)(Système de Gestion de Base de Données) du marché comme [MySQL](http://www.mysql.fr/), [Oracle](http://www.oracle.com/), [DB2](http://www-01.ibm.com/software/data/db2/),[PostgreSQL](http://www.postgresql.fr/), etc…

[](http://www.aymericlagier.com/wp-content/uploads/2010/01/schema_ado.png)

Côté code, les classes des *Data Providers*se trouvent dans les *namespaces* suivants :

* *System.Data.OleDb*
* *System.Data.SqlClient*
* *System.Data.OracleClient*
* *System.Data.Odbc*

Il ne faut pas oublié de rajouter à ces *namespaces*, *System.Data* qui contient les classes essentielles au fonctionnement d’ADO.NET.

**Déclaration d’un *Data Provider* pour SQL Server :**

Déclaration des *namespaces*

using System.Data;

using System.Data.SqlClient;

Création de la connexion

SqlConnection connexion = new SqlConnection();

connexion.ConnectionString = @"Data Source=localhost\SQLEXPRESS;";

connexion.ConnectionString += @"Initial Catalog=Northwind;Integrated Security=SSPI";

On construit donc la connexion avec la ***ConnectionString***qui indique le serveur de base de données (localhost\SQLEXPRESS), la base de données (Northwind) et la manière de s’authentifier sur le serveur (SSPI).

Note :

L’exemple ci-dessus utilise le *Data Provider* de SQL Server. Pour les autres *Data Providers*, il suffit de remplacer *SqlConnection* par *OracleConnecion*, *OleDbConnection* ou *OdbcConnection*. Le même principe s’applique pour les commandes qui suivront dans cet article.

**La gestion des données**

L’un des avantages d’ADO.NET est sa généricité. Peu importe la source de données, le conteneur qui accueillera les données sera le même, le ***DataSet***. Pour faire une analogie avec le C# de base, un*DataSet* est en quelque sorte un tableau (*array*), mais spécialement adapté pour accueillir des données venant d’une source de données relationnelle.

ADO.NET peut fonctionner de 2 manières diffèrentes. En mode connecté (Direct Data Access), le programme se connecte à la base de données et fait un certain nombre d’opérations (*SELECT*,*INSERT*, *UPDATE*, etc…), puis la connexion est fermée. Aucune donnée n’est donc stockée en mémoire sur le client.  
En mode déconnecté, le programme se connecte à la base de données durant un temps très bref pour rapatrier les données en mémoire. La connexion est ensuite interrompue. Une fois les opérations (*INSER*T, *DELETE*, etc…) réalisées sur le *DataSet*, le programme se connecte à nouveau à la base de données et sauvegarde les changements.

Quel mode choisir ?

L’utilisation du mode connecté est recommandé pour les sites web en ASP.NET puisque le cycle de vie d’un programme est totalement différent par rapport à une application de bureau. Dans une page web, le cycle de vie du programme n’est que de quelques secondes, les données n’ont donc pas besoin d’être stockées en mémoire. Concernant les applications lourdes, le mode déconnecté est utile lorsqu’il y a de nombreux traitements à faire sur les données avant de les modifier dans la base de données.

Le mode Connecté (Direct Data Access)

La commande SELECT

SqlConnection connexion = new SqlConnection();

connexion.ConnectionString = @"Data Source=localhost\SQLEXPRESS;";

connexion.ConnectionString += @"Initial Catalog=Northwind;Integrated Security=SSPI";

// Requête SQL

SqlCommand selectCommand = new SqlCommand();

selectCommand.Connection = connexion; // Connexion instanciée auparavant

selectCommand.CommandText = "SELECT ContactName FROM Customers";

SqlDataReader reader; // Contiendra les données

try

{

connexion.Open(); // Ouverture de la connexion

reader = selectCommand.ExecuteReader(); // Exécution de la requête SQL

while (reader.Read())

{

// Affichage des données

Console.WriteLine(reader["ContactName"]);

}

reader.Close();

}

catch (Exception ex)

{

// Affiche des erreurs

Console.WriteLine(ex.Message);

}

finally

{

// Fermeture de la connexion à la base de données

connexion.Close();

}

Détails du code :

Premièrement, il faut déclarer la connexion à la base de données, puis définir la requête à exécuter. On déclare un *SqlDataReader* qui contiendra nos données une fois la commande SQL exécutée. La suite du code est imbriqué dans un bloc *try/catch/finally* afin de gérer les éventuelles erreurs qui pourraient survenir. La connexion à la base de données est ouverte et la requête exécutée. La boucle *while* permet de parcourir tous les enregistrements afin de les afficher dans la console. A la fin de l’opération, on ferme le *SqlDataReader* ainsi que la connexion à la base de données. Le bloc *finally* permet de fermer la connexion à la base de données dans tous les cas (succès de la requête ou erreur).

Les commandes INSERT, UPDATE et DELETE

Le principe est le même, à l’exception du *DataReader* qui n’est plus utile et de la méthode permettant d’exécuter la requête : *ExecuteReader()* qui devient *ExecuteNonQuery()* puisque la requête ne renvoie que le nombre de lignes modifiées.

//Connexion à la base de données

SqlConnection connexion = new SqlConnection();

connexion.ConnectionString = @"Data Source=localhost\SQLEXPRESS;";

connexion.ConnectionString += @"Initial Catalog=Northwind;Integrated Security=SSPI";

//Requête SQL

SqlCommand updateCommand = new SqlCommand(

"UPDATE Customers SET ContactName = 'Aymeric Lagier' WHERE CustomerID = 'ALFKI'");

updateCommand.Connection = connexion;

int lignesModifiees = 0;

try

{

connexion.Open();

lignesModifiees = updateCommand.ExecuteNonQuery(); // Exécution de la requête SQL

Console.WriteLine("{0} ligne(s) modifiée(s)", lignesModifiees.ToString());

}

catch (Exception ex)

{

// Affiche des erreurs

Console.WriteLine(ex.Message);

}

finally

{

// Fermeture de la connexion à la base de données

connexion.Close();

}

Le processus est identique pour le INSERT et le DELETE.

La protection des données dans les requêtes

Dans la requête ci-dessus, ‘ALFKI’ est inscrit en dur dans la requête, cela ne pause donc pas de problème. Si maintenant le *CustomerID* est une valeur entrée par l’utilisateur du programme. Si l’utilisateur entre la valeur *‘ALFK’I*, le programme affichera une erreur. Deuxième problème, les injections SQL. L’utilisateur du programme peut délibérément entrer des paramètres spéciaux qui détourneront la requête de son but initial.

Pour contrer ce type de problème, ADO.NET met en place les requêtes paramètrées :

Requête normale :

SqlCommand requete = new SqlCommand(

"SELECT \* FROM Customers WHERE CustomerID = 'ALFKI'");

SqlCommand requete = new SqlCommand(

"SELECT \* FROM Customers WHERE CustomerID = @CustomerID");

// myVar contient une chaine entrée par l'utilisateur

requete.Parameters.AddWithValue("@CustomerID", myVar);

De cette manière un utilisateur peut entrer n’importe quelle chaine de caractères dans *myVar*, il n’y aura pas de problème d’injection SQL ni de chaine invalide. Ce processus fonctionne également pour le mode déconnecté expliqué ci-dessous.

Le mode Deconnecté (Direct Data Access)

La commande SELECT

Pour le mode déconnecté la processus est diffèrent. Le programme établit une connexion avec la base de données, récupère les données et referme immédiatement la connexion. Les opérations sur les données (affichage dans la console) ne se font qu’une fois la connexion fermée.

// Connexion à la base de données

SqlConnection connexion = new SqlConnection();

connexion.ConnectionString = @"Data Source=localhost\SQLEXPRESS;";

connexion.ConnectionString += @"Initial Catalog=Northwind;Integrated Security=SSPI";

// Requête SQL

SqlCommand selectCommand = new SqlCommand();

selectCommand.Connection = connexion; // Connexion instanciée auparavant

selectCommand.CommandText = "SELECT \* FROM Customers";

SqlDataAdapter adapter = new SqlDataAdapter(selectCommand); // Permet de lire les données

DataSet data = new DataSet(); // Contiendra les données

try

{

connexion.Open(); // Ouverture de la connexion

adapter.Fill(data, "Customers"); // Récupère les données

}

catch (Exception ex)

{

// Affiche des erreurs

Console.WriteLine(ex.Message);

}

finally

{

// Fermeture de la connexion à la base de données

connexion.Close();

}

// Affichage dans la console

foreach (DataRow row in data.Tables["Customers"].Rows)

{

Console.WriteLine(row["ContactName"]);

}

Détails du code :

Comme pour le mode connecté, la connexion à la base de données et la requête SQL sont déclarées. A la place du *SqlDataReader*, le *SqlDataAdapter* et le *DataSet* sont déclarés. Le*SqlDataAdapter* va permettre au programme de récupérer les données qui seront stockées dans le*DataSet*. Une fois les données enregistrées en mémoire, la connexion à la base de données est interrompue. Après la fermeture de la connexion, le programme peut traiter les données (affichage sur la console) contenues en mémoire.

Les commandes INSERT, UPDATE, DELETE

Pour les requêtes INSERT, UPDATE et DELETE, le processus est le même. C’est-à-dire se connecter à la base de données pendant un très court instant afin de charger la table qui nous intéresse en mémoire pour ensuite travailler dessus.

// Connexion à la base de données

SqlConnection connexion = new SqlConnection();

connexion.ConnectionString = @"Data Source=localhost\SQLEXPRESS;";

connexion.ConnectionString += @"Initial Catalog=Northwind;Integrated Security=SSPI";

// Requête SQL (SELECT)

SqlCommand selectCommand = new SqlCommand();

selectCommand.CommandText = "SELECT City, Country FROM Customers WHERE Country = 'UK'";

selectCommand.Connection = connexion;

//Requête SQL (UPDATE)

SqlCommand updateCommand = new SqlCommand();

updateCommand.CommandText = "UPDATE Customers SET City = @City WHERE Country = @Country";

updateCommand.Connection = connexion;

// Paramètres

updateCommand.Parameters.Add("@City", SqlDbType.NVarChar, 100, "City");

updateCommand.Parameters.Add("@Country", SqlDbType.NVarChar, 100, "Country");

SqlDataAdapter adapter = new SqlDataAdapter(); // Permet de lire les données

adapter.SelectCommand = selectCommand; // Définitiion la requête SELECT

adapter.UpdateCommand = updateCommand; // Définitiion la requête UPDATE

DataSet data = new DataSet(); // Contiendra les données

try

{

connexion.Open(); // Ouverture de la connexion

adapter.Fill(data, "Customers"); // Récupère les données

}

catch (Exception ex)

{

// Affiche des erreurs

Console.WriteLine(ex.Message);

}

finally

{

// Fermeture de la connexion à la base de données

connexion.Close();

}

// Modification du champ "City"

foreach (DataRow row in data.Tables["Customers"].Rows)

{

row["City"] = "London";

}

int lignesModifiees = 0; // Nombre de lignes modifiées

// Modification dans la base de données

lignesModifiees = adapter.Update(data.Tables["Customers"]);

Console.WriteLine("{0} ligne(s) modifiée(s)", lignesModifiees);

Détails du code :

Comme dans la requête SELECT, la connexion et la requête SELECT sont définies. Petite nouveauté, la définition d’une commande UPDATE en plus de la commande SELECT ainsi que la déclaration de paramètres correspondants aux champs de la requête UPDATE. Après le stockage des données dans la mémoire, la modification des données peut s’opérer. Une fois les modifications réalisées, la méthode*Update()* de *DataAdapter* prend en paramètre la *DataTable* modifiée et envoie les données dans la base de données.

Les exemples de cet article sont repris dans ce [projet](http://www.aymericlagier.com/wp-content/uploads/2010/01/ADO.NET_.zip). Attention, pour fonctionner ce projet a besoin de la base de données Northwind disponible [ici](http://code.msdn.microsoft.com/Project/Download/FileDownload.aspx?ProjectName=northwind&DownloadId=2791).

**Conclusion**

Cet article introduit l’ensemble de classes utilisées pour interagir avec une base de données en .NET en utilisant le mode connecté principalement pour l’ASP.NET et le mode déconnecté pour les applications de bureau exécutant un nombre important d’opérations sur les données. ADO.NET offrent d’autres fonctionnalités comme les relations entre les tables avec la classe *[DataRelation](http://msdn.microsoft.com/fr-fr/library/system.data.datarelation(VS.80).aspx" \o "Classe DataRelation" \t "_blank)*.

Source de : <http://www.aymericlagier.com/2010/01/29/introduction-a-ado-net-en-c/>