ADO.NET : Connection, Command et DataReader avec VB.NET

par leduke

Date de publication : 21/08/2003

Dernière mise à jour : 21/07/2009

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

I - Avant propos

Article écrit pour la revue programmez par leduke - Toute reproduction, même partielle doit-être soumise à l'accord de l'auteur.

II - Introduction

Avec .NET, Microsoft propose sa dernière technologie d'accès aux données ADO .Net. C'est un ensemble de classes permettant de récupérer et de manipuler des données et qui fait partie intégrante de la nouvelle plate-forme appelée .Net Framework.

Si vous êtes habitués aux développements utilisant les ADO, vous allez constater que ces nouvelles classes sont différentes tant dans la syntaxe que dans la démarche de conception proposée.

Nous allons présenter dans cet article les nouveautés ADO.Net, le modèle ADO.Net ainsi que les objets Connection, Command et DataReader en précisant quand ce sera nécessaire, les équivalences avec les ADO.

III - 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.

IV - 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 utilisée.

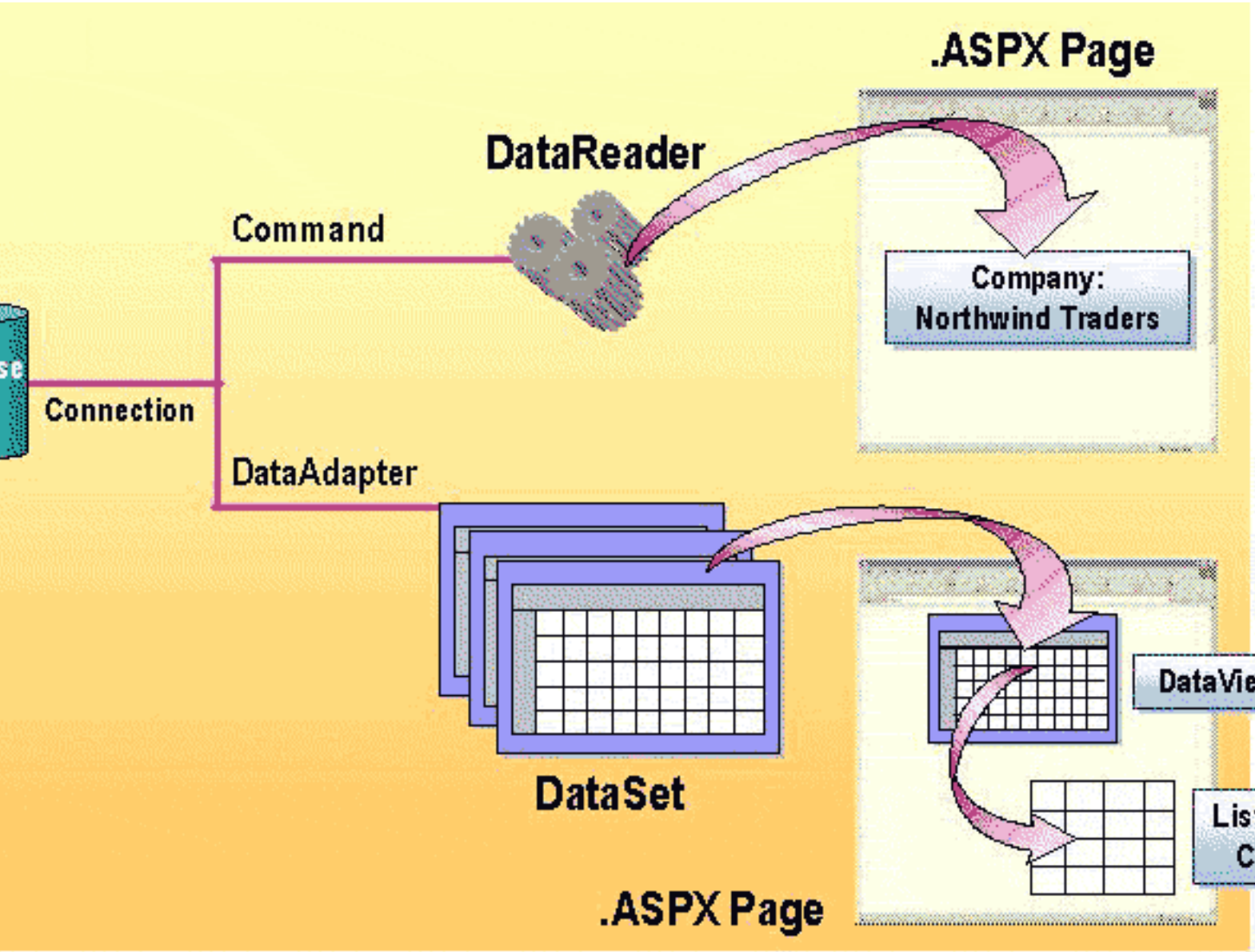
```
'Inclusion d'un namespace

Imports System.Data
Imports 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
System.Data.SQLTypes	Propose des classes pour des types de données spécifiques à Microsoft SQL Server

V - 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 spécifique
Command	Exécute une commande sur une source de données
DataReader	Lit un flux de données à partir d'une source de données en mode connecté. Le mode d'acc avant seulement.
DataSet	Représente un ensemble de données en mode déconnecté. Il peut être constitué de plusie contraintes existant entre elles.
DataAdapter	Remplit un DataSet et répercute les mises à jour dans la source de données.

VI - 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 choisi. 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

Imports System
Imports System.Data.SqlClient
Imports System.IO

namespace ExempleAdoNetVBnet

    public class SQLConnexion

        public static void Main()

            Dim strConnexion As
 String = "Data Source=localhost; Integrated Security=SSPI;" & "Initial Catalog=Northwind"
            try

                Dim oConnection As SqlConnection = New SqlConnection(strConnexion)
                oConnection.Open()
                Console.WriteLine("Etat de la connexion : " & oConnection.State)
                oConnection.Close()

            catch e as Exception

                Console.WriteLine("L'erreur suivante a été rencontrée :" & e.Message)
            End Try
        End Sub
    End Class
End Namespace
```

VI - 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'enregistrements 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
Imports System
Imports System.Data.SqlClient
Imports System.IO


Namespace ExempleAdoNetVBNET
    Public Class CommandeSQL
        Public Shared Sub Main()
            Dim strConnexion As
 String = "Data Source=localhost; Integrated Security=SSPI;" + "Initial Catalog=Northwind"
            Dim strRequete As String = "INSERT INTO Region VALUES (5,'Sud')"
            Try
                Dim oConnection As New SqlConnection(strConnexion)
                Dim oCommand As New SqlCommand(strRequete, oConnection)
                oConnection.Open()
                oCommand.ExecuteNonQuery()
                oConnection.Close()

            Catch e As Exception
                Console.WriteLine(("L'erreur suivante a été rencontrée :" + e.Message))
            End Try
        End Sub 'Main
    End Class 'CommandeSQL
End Namespace 'ExempleAdoNetVBNET
```

VIII - 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 important 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 leurs noms soit par leurs références ordinales. Une solution plus performante est proposée permettant d'accéder aux valeurs dans leurs types de données natifs (GetInt32, GetDouble, GetString .). Par exemple si la première colonne de la ligne indicée par 0 est de type int, alors il est possible de la récupérer à l'aide de la méthode **GetInt32** de l'objet **DataReader**.

```
Dim iColonne As Integer
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 retours, ils ne pourront être récupérés qu'a l'issue de la fermeture du DataReader.

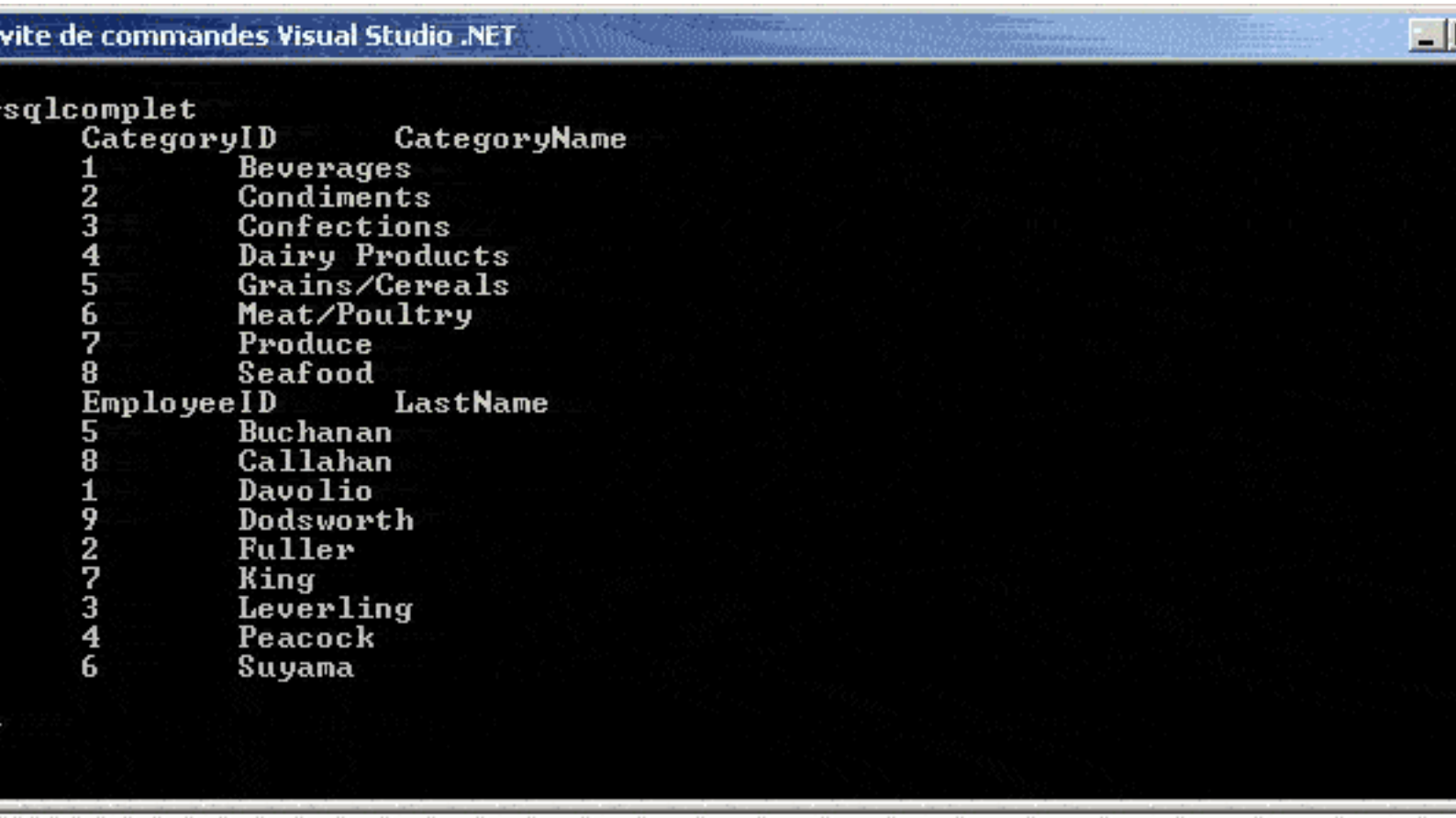
Pour augmenter les performances, il est parfois nécessaire de soumettre plusieurs requêtes à la fois. L'objet DataReader répond à ce besoin avec la méthode **NextResult** permettant de passer d'un jeu d'enregistrement à un autre. Pour les habitués des ADO, cette méthode est équivalente au **NextRecordset**.

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

```
' Exemple d'extraction de données avec l'objet DataReader
Imports System
Imports System.Data.SqlClient
Imports System.IO


Namespace ExempleAdoNetVBNET
    Public Class CommandeSQL
        Public Shared Sub Main()
            Dim strConnexion As
 String = "Data Source=localhost; Integrated Security=SSPI;" + "Initial Catalog=Northwind"
            Dim strRequete As
 String = "SELECT CategoryID, CategoryName FROM Categories;" + "SELECT EmployeeID, LastName FROM Employees"
            Try
                Dim oConnection As New SqlConnection(strConnexion)
                Dim oCommand As New SqlCommand(strRequete, oConnection)
                oConnection.Open()
                Dim oReader As SqlDataReader = oCommand.ExecuteReader()
                Do
                    Console.WriteLine(ControlChars.Tab + "{0}" + ControlChars.Tab + "{1}",
 oReader.GetName(0), oReader.GetName(1))
                    While oReader.Read()
                        Console.WriteLine(ControlChars.Tab + "{0}" + ControlChars.Tab + "{1}",
 oReader.GetInt32(0), oReader.GetString(1))
                    End While
                Loop While oReader.NextResult()
                oReader.Close()
                oConnection.Close()
            Catch e As Exception
                Console.WriteLine(("L'erreur suivante a été rencontrée :" + e.Message))
            End Try
        End Sub 'Main
    End Class 'CommandeSQL
End Namespace 'ExempleAdoNetVBNET
```

Ce qui donne à l'exécution le résultat suivant :



IX - 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'ordres 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 facile 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
Dim oParam As SqlParameter = oCmd.Parameters.Add("@nom", SqlDbType.Char, 25, Nom)
oParam.Value = "John"

' Création d'un paramètre retour
Dim oParam As SqlParameter = oCmd.Parameters.Add("ReturnValue", SqlDbType.Int)
oParam.ParameterDirection = ParameterDirection.ReturnValue
```

Précisons :

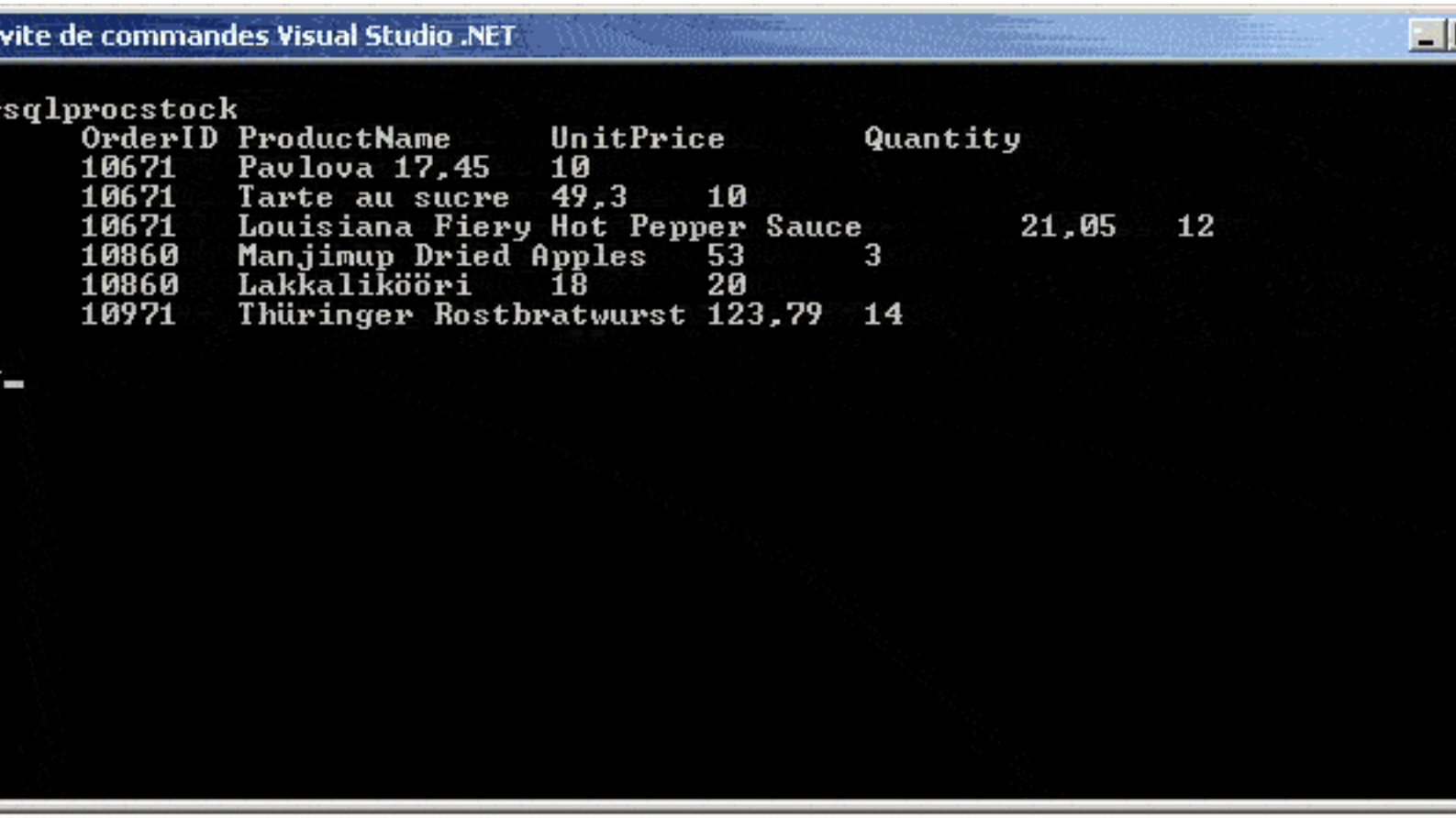
- qu'il est nécessaire d'affecter à la propriété Value de l'objet Parameter une valeur pour les paramètres qui le nécessitent.
- qu'il faut décrire explicitement le type de paramètres via la propriété **ParameterDirection** de l'objet Parameter (sauf pour les paramètres en entrée). Celle-ci peut prendre 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
Imports System
Imports System.Data.SqlClient
Imports System.Data
Imports System.IO


Namespace ExempleAdoNetVBNET
    Public Class SQLProcStock
        Public Shared Sub Main()
            Dim strConnexion As
 String = "Data Source=localhost; Integrated Security=SSPI;" + "Initial Catalog=Northwind"
            Dim strProcedureStockee As String = "GetCustomerOrders"
            Try
                Dim oConnection As New SqlConnection(strConnexion)
                Dim oCommand As New SqlCommand(strProcedureStockee, oConnection)
                oCommand.CommandType = CommandType.StoredProcedure
                Dim oParam As SqlParameter = oCommand.Parameters.Add("@CustomerName",
 SqlDbType.NVarChar, 50, "CustomerName")
                oParam.Value = "France restauration"
                oConnection.Open()
                Dim oReader As SqlDataReader = oCommand.ExecuteReader()
                Console.WriteLine(ControlChars.Tab + "{0}" + ControlChars.Tab + "{1}" +
 ControlChars.Tab + "{2}" + ControlChars.Tab + "{3}",
    oReader.GetName(0), oReader.GetName(3), oReader.GetName(4), oReader.GetName(5))
                While oReader.Read()
                    Console.WriteLine(ControlChars.Tab + "{0}" + ControlChars.Tab + "{1}" +
 ControlChars.Tab + "{2}" + ControlChars.Tab + "{3}",
     oReader.GetInt32(0), oReader.GetString(3), oReader.GetDecimal(4), oReader.GetInt16(5))
                End While oReader.Close()
                oConnection.Close()
            Catch e As Exception
                Console.WriteLine(("L'erreur suivante a été rencontrée :" + e.Message))
            End Try
        End Sub 'Main
    End Class 'SQLProcStock
End Namespace 'ExempleAdoNetVBNET
```

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



X - 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.