« Fetch Progressif » avec les délégués et StrataFrame

Les Pinter

Cliquez pour télécharger le code source

Récemment, un des lecteurs de ma lettre d'information m'a demandé comment faire une manip qui est tout à fait ordinaire en FoxPro. Il souhaitait remplir un grid avec une partie des enregistrements provenant d'une table SQL, et continuer ce remplissage en arrière-plan. De la sorte, les utilisateurs pourraient tout de suite commencer à lire les enregistrements du grid sans attendre la fin du chargement. Bref, rien d'extraordinaire dans cette demande. Oui mais voilà, ça existe en FoxPro, et pas en .NET.

Allons donc, on peut *tout* faire en .NET. Sans que ce soit trop compliqué ?

Ça n'était pas vraiment complexe, mais tout de même pas aussi simple que de cocher une case comme on le fait en Fox. Il m'a fallu utiliser les *délégués*. Et également StrataFrame, pour alléger le travail comme d'habitude. Voilà comment j'ai procédé.

Un exemple de test

Pour notre démonstration, je vais récupérer les 100 premières commandes de la table Orders, dans la base de données NorthWind (il en existe d'autres ?). Comme elle ne contient que 730 enregistrements, il nous faudra simuler le temps d'attente que vous auriez pour obtenir les enregistrements depuis SQL. Mais ça vous permettra de comprendre le principe de fonctionnement, et vous pourrez ainsi faire vos propres tests sur une grosse table, que vous choisirez vous-même dans quelques instants.

<Avertissement> J'entends souvent des développeurs FoxPro, qui sont étonnés de la lenteur avec laquelle un dataset est peuplé par les enregistrements provenant d'une grosse table en .NET. le problème ne provient pas de SQL Server. Essayez avec l'Analyseur de Requêtes, ou bien lancez une requête SELECT dans SQL Server Management Studio, et vous verrez que les performances sont proches de celles de FoxPro. C'est le chargement du Dataset qui est lent. Utilisez un SQLDataReader, et vous n'aurez aucun problème de vitesse. C'est vrai que l'approche du DataSet semble plus logique quand on a l'habitude de récupérer une table et de la manipuler en mémoire. Mais réfléchissez-y : le DataReader existe parce que bien souvent, c'est la façon la plus efficace d'obtenir le résultat que vous escomptez. Alors, utilisez-le !</Avertissement>

Pour cette démo, j'utilise le formulaire présenté ci-dessous (figure 1) :

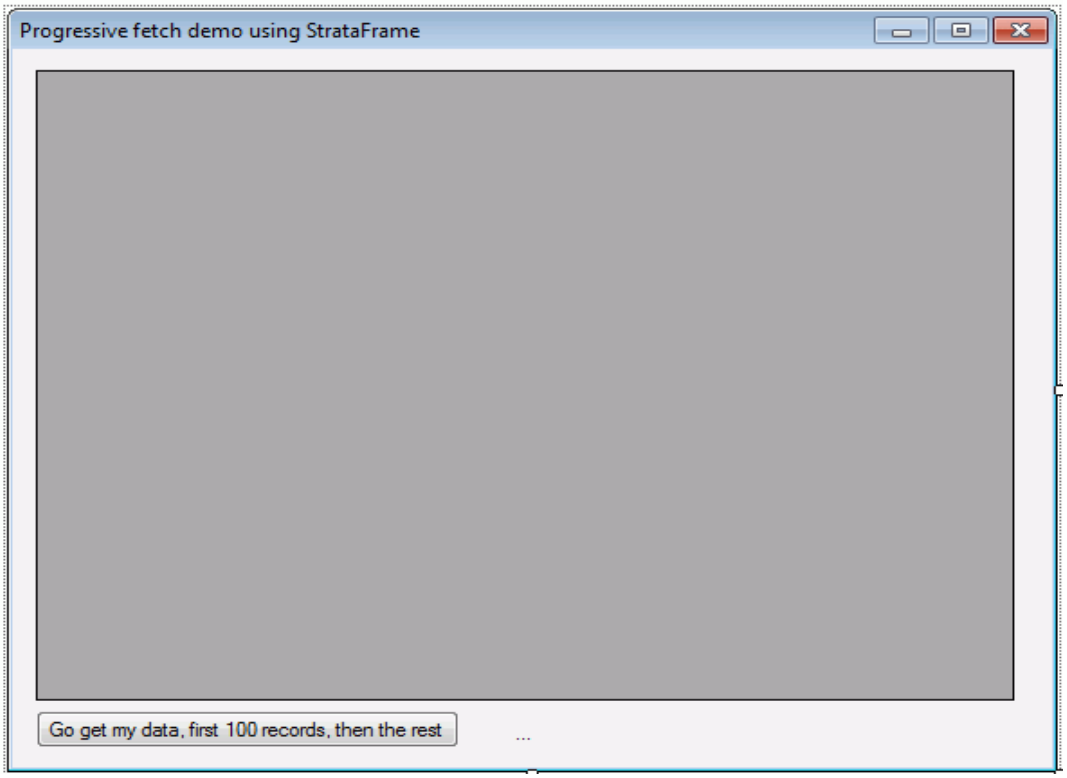


Figure 1 – un Formulaire avec un grid pour les commandes des clients

Le grand bouton au pied du formulaire permet de récupérer immédiatement les 100 premiers enregistrements, et les suivants en tâche d'arrière-plan, pendant que vous faites défiler le grid vers le bas. Lancez le programme d'exemple, il ne fait que cela, avec une pause d'une seconde. Voici le code du bouton :

```
//Form1 code-behind:
using System.Threading;

namespace AsyncSQL {
  public partial class Form1 : MicroFour.StrataFrame.UI.Windows.Forms.StandardForm {

    public Form1() { InitializeComponent(); }

    public delegate void RefreshTheGrid();
    public delegate void SetTextCallback(string text);

    private void Button1_Click(object sender, System.EventArgs e) {

      ordersBO1.First100Orders();
      dataGridView1.DataSource = ordersBO1.CurrentView;
      label1.Text = "First 100 orders loaded"; label1.Refresh();

      Thread t = new Thread(GetRest);
      t.Start();
    }

    public void GetRest() {

      Thread.Sleep(1000);  // simule une attente; à commenter pour tester sur de grosses tables

      ordersBO1.RestOfOrders();
      string TextLoaded = "Rest of orders loaded ( " + ordersBO1.Count.ToString() + " records )";

      RefreshTheGrid cb1 = new RefreshTheGrid(GridRefresher); // délégué, déclaré ci-dessus
      Invoke(cb1);
      SetTextCallback cb2 = new SetTextCallback(SetText);     // délégué, déclaré ci-dessus
      Invoke(cb2, new object[] { TextLoaded });
    }

    private void GridRefresher() { dataGridView1.DataSource = ordersBO1.CurrentView; }

    private void SetText(string text) { label1.Text = text; }

  }
}

// Business object code:
using System;
using System.ComponentModel;
using System.Runtime.Serialization;
using MicroFour.StrataFrame.Business;

namespace AsyncSQL {
  [Serializable()]
  public partial class OrdersBO : MicroFour.StrataFrame.Business.BusinessLayer {

    public int lastorder;

    #region Constructors     (aucune modification)

    #region Data Retrieval Methods

    public void First100Orders()
    { FillDataTable("SELECT TOP 100 * FROM Orders");
      MoveLast(); lastorder = (int)this["OrderID"];
    }

    public void RestOfOrders()
    { string s = "SELECT * FROM ORDERS WHERE OrderID > " + lastorder.ToString();
      AppendDataTable(s, AppendDataTableOptions.ReplaceExistingDuplicates);
    }

    #endregion

    #region Event Handlers     (aucune modification)

  }
}
```

Voici les explications de ce code, au fil de chaque bloc:

```
using System.Threading;

namespace AsyncSQL {
  public partial class Form1 : MicroFour.StrataFrame.UI.Windows.Forms.StandardForm {

    public Form1() { InitializeComponent(); }
```

Je re-condense en quelques lignes le code aéré par l'éditeur de C#, là où ça ne prête pas à conséquence. Mais si vous préférez le voir ainsi :

```
    public Form1()
      {
        InitializeComponent();
      }
```

Libre à vous de le remettre en forme comme vous le dicte votre morale. J'essaye seulement de gagner un peu de place en hauteur dans mes bureaux !

Il nous faut deux déclarations de délégués, dont nous nous servirons ensuite. L'un d'entre eux prend un paramètre, parce que les threads ne peuvent pas voir les champs déclarés dans le code appelant.

```
    public delegate void RefreshTheGrid();
    public delegate void SetTextCallback(string text);
```

La méthode click du bouton appelle la méthode First100_Orders du Business Object OrdersBO, laquelle méthode charge dans le Business Object OrdersBO les 100 premiers enregistrements en s'appuyant sur la méthode StrataFrame FillDataTable.

Note : StrataFrame propose deux types de méthodes pour peupler les Business Objects : les méthodes Fill, qui sont rattachées à la liaison des contrôles et des données, et les méthodes Get, qui ne déplacent pas le pointeur d'enregistrement et n'ont rien à voir avec l'Interface Utilisateur. Vous utiliserez ces dernières (Get) quand vous aurez par exemple besoin de parcourir en boucle une table sans que cela n'affecte le pointeur d'enregistrement à l'écran. Ici, comme tout ce que nous faisons est directement lié au Grid, c'est bien une méthode Fill qu'il nous faut utiliser. La CurrentView d'un Business Object est comme la DefaultView d'un DataSet, c'est elle que nous définissons comme DataSource du DataGridView.

Et enfin, je rafraichis le texte de la petite étiquette que vous pouvez deviner à droite du bouton (remarquez les … ?)

```
    private void Button1_Click(object sender, System.EventArgs e) {

      ordersBO1.First100Orders();
      dataGridView1.DataSource = ordersBO1.CurrentView;
      label1.Text = "First 100 orders loaded"; label1.Refresh();
```

C'est maintenant que ça devient intéressant. Je commence un nouveau Thread, tout simplement en le déclarant et en appelant sa méthode Start :

```
      Thread t = new Thread(GetRest);
      t.Start();
    }
```

Vous pouvez mettre ce que vous voulez à la place de GetRest ; ça démarrera et s'exécutera de façon indépendante, et ça finira quand ça devra finir. Vos utilisateurs peuvent continuer à faire ce qu'ils veulent pendant ce temps là. C'est à vous seul qu'il appartient de les informer de la fin de l'opération. Ici, je me contente de mettre à jour une étiquette sur le formulaire, comme vous le verrez ci-dessous.

Vous vous souvenez de la méthode First100Orders que j'ai écrite dans la classe OrdersBO ? à la fin de cette méthode, j'enregistre le contenu du champ OrderID du 100ème enregistrement (le dernier) dans une propriété *int* que j'ai créé sur le Business Object, et que j'ai appelé LastRecord.

```
    public void First100Orders()
    { FillDataTable("SELECT TOP 100 * FROM Orders");
      MoveLast(); lastorder = (int)this["OrderID"]; }
```

Étant donné que la table Orders du database Northwind ne compte pas beaucoup d'enregistrements, j'ai été oblige de simuler une attente. C'est le but de la ligne Thread.Sleep(1000). Vous n'en aurez pas besoin quand vous testerez cette technique vous-mêmes (et c'est ce que vous allez faire dès que vous aurez terminé la lecture de cet article, afin d'acquérir définitivement la maitrise parfaite de cette technique, n'est-ce pas ?)

Il ne nous reste plus qu'à charger le reste des enregistrements des commandes, en commençant juste après la dernière obtenue par la méthode First100Orders, à rafraichir le grid, et à réafficher le nombre d'enregistrements. Voici comment procéder :

```
    ordersBO1.RestOfOrders();
    string TextLoaded = "Rest of orders loaded ( " + ordersBO1.Count.ToString() + " records )";

    RefreshTheGrid cb1 = new RefreshTheGrid(GridRefresher); // délégué, déclaré ci-dessus
    Invoke(cb1);
    SetTextCallback cb2 = new SetTextCallback(SetText);     // délégué, déclaré ci-dessus
    Invoke(cb2, new object[] { TextLoaded });
  }
```

La méthode RestOfOrders utilise la valeur du dernier OrderID des 100 que nous avons obtenus par la méthode Fill100Orders. Elle s'en sert pour charger dans le Business Object OrdersBO les enregistrements suivants. Ces enregistrements sont tout simplement ajoutés à la fin des 100 premiers : si vous ne videz pas expressément le Business Object, les données existantes y persistent et les données nouvelles y sont ajoutées. Ça semble évident…

```
  public void RestOfOrders()
  { string s = "SELECT * FROM ORDERS WHERE OrderID > " + lastorder.ToString();
    AppendDataTable(s, AppendDataTableOptions.ReplaceExistingDuplicates);
  }
```

L'utilisation des délégués

L'appel d'une fonction dans un thread nécessite de créer une instance du délégué, en utilisant la méthode que vous voulez appeler en tant que paramètre. Je le redis plus lentement :

```
RefreshTheGrid              cb1 = new RefreshTheGrid(GridRefresher); Invoke(cb1);
      créez un truc, appelez-le cb1,   utilisez-le pour appeler GridRefresher ;     et en avant !
```

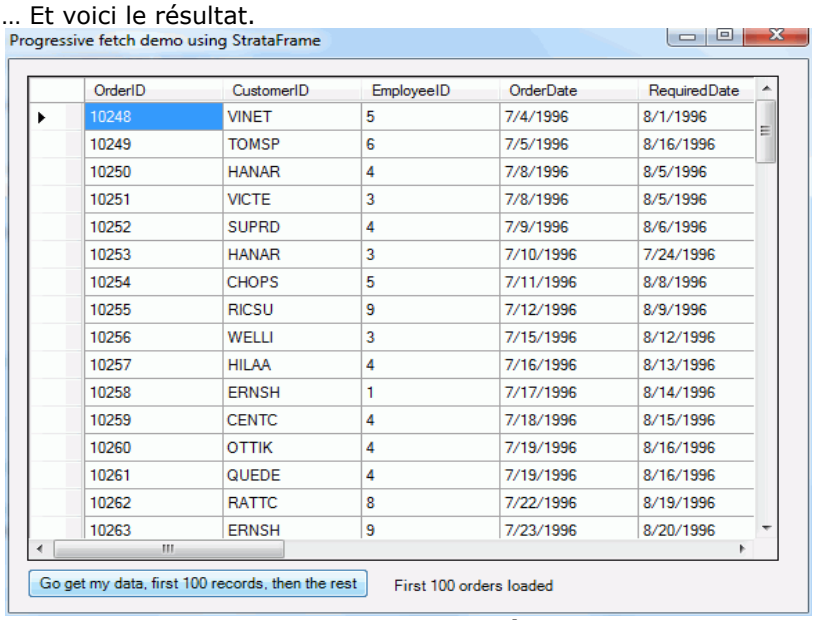
D'accord, ça semble un peu bizarre, mais je ne vois pas de meilleure façon de faire. Ça n'est pas du tout intuitif, mais la bonne nouvelle, c'est que ça fonctionne comme prévu.

Dans chacune des méthodes appelées par les délégués, il n'y a qu'une ligne de code. Je vous le redis, si vous préférez que ça en fasse quatre, à vous de jouer. Remarquez bien que après avoir appelé la méthode RestOfOrders() du BO OrdersBO, nous devons utiliser le délégué pour affecter la CurrentView de OrdersBO au DataSource du grid : les threads ne peuvent pas communiquer directement avec les objets du formulaire, il faut passer par les délégués, ils servent à ça.

```
  private void GridRefresher() { dataGridView1.DataSource = ordersBO1.CurrentView; }

  private void SetText(string text) { label1.Text = text; }
```

… Et voici le résultat.



OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate
10248	VINET	5	7/4/1996	8/1/1996
10249	TOMSP	6	7/5/1996	8/16/1996
10250	HANAR	4	7/8/1996	8/5/1996
10251	VICTE	3	7/8/1996	8/5/1996
10252	SUPRD	4	7/9/1996	8/6/1996
10253	HANAR	3	7/10/1996	7/24/1996
10254	CHOPS	5	7/11/1996	8/8/1996
10255	RICSU	9	7/12/1996	8/9/1996
10256	WELLI	3	7/15/1996	8/12/1996
10257	HILAA	4	7/16/1996	8/13/1996
10258	ERNSH	1	7/17/1996	8/14/1996
10259	CENTC	4	7/18/1996	8/15/1996
10260	OTTIK	4	7/19/1996	8/16/1996
10261	QUEDE	4	7/19/1996	8/16/1996
10262	RATTC	8	7/22/1996	8/19/1996
10263	ERNSH	9	7/23/1996	8/20/1996

Figure 2 – le grid partiellement peuplé, en attente des enregistrements suivants

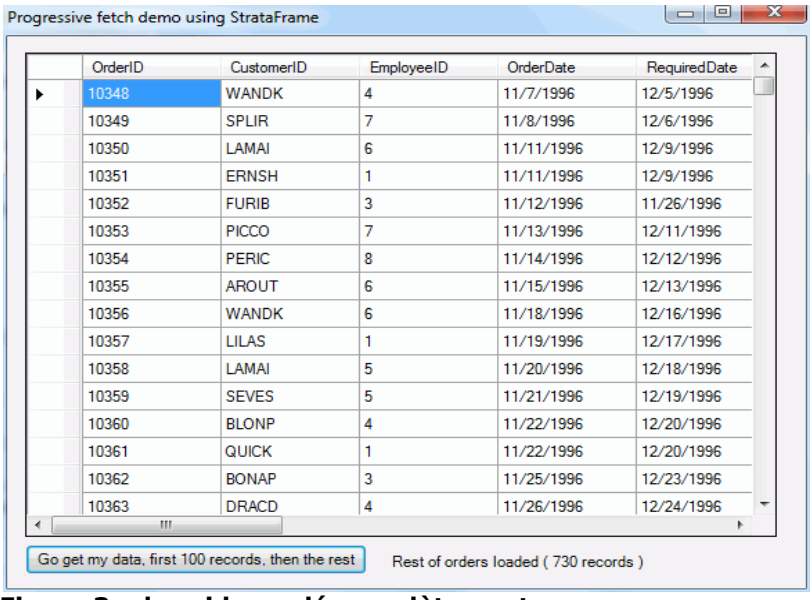


Figure 3 – le grid peuplé complètement

Le code source à télécharger contient les versions C# et VB, de telle sorte que vous puissiez ainsi apprendre l'autre langage. C'est bien ce que vous aviez l'intention de faire depuis.. euh… combien de temps ?

Si vous n'avez pas encore utilisé les délégués, cela vous demandera quelques secondes de concentration. Je dois admettre que c'est assez drôle. Mais c'est la seule façon de lancer quelque chose dans un thread, et de le faire communiquer avec le programme appelant. Allez faire un tour dehors, et rentrez chez vous quand ça ira mieux…

Conclusion

Pendant des années, je me suis soigneusement gardé d'utiliser les délégués, en m'arrangeant pour obtenir ce que je voulais sans m'en servir. Mais en réalité ça ne représente que quelques lignes de code, et les méthodes Fill de StratatFrame, tout comme leurs Business Objects, réduisent vraiment ce travail au strict minimum.

@+

Les ❑❑❑

---

 tél +1 (508) 650-3610

La conversion automatique entre C# et VB provient de Visible C#/visible VB de *Tangible Software Solutions*