



## Sommaire > Fondements de Linq

- Qu'est-ce que Linq to Objects ?
- Comment réaliser ma première requête Linq to Objects ?
- Peut-on requêter n'importe quelle collection d'objets ?
- Comment afficher le résultat d'une requête Linq dynamiquement ?
- Comment exécuter une requête sur une collection non générique ?
- Comment exécuter une requête sur une collection non générique contenant des objets de types différents ?
- Qu'est-ce qu'une séquence ?
- Qu'est-ce que l'exécution différée de requête ?
- Qu'est-ce que la réutilisation de requête ?
- Qu'est-ce qu'un opérateur de requête ?
- Quels sont les opérateurs de requête standards disponibles ?
- Qu'est-ce qu'une expression de requête (Query Expression) ?
- Est-ce que tous les opérateurs standards de requête ont leur équivalence en tant qu'expression de requête ?
- Comment utiliser Linq to Objects dans vos projets ?
- Qu'est-ce que la 'sugar syntax' en Linq ?
- Comment grouper des éléments sur base de critères multiples ?
- Comment rendre paramétrable un critère de filtre ?

Précédent Sommaire  Rechercher

Version PDF (Miroir) Version hors-ligne (Miroir)

### Qu'est-ce que Linq to Objects ? [haut]

auteur : Jérôme Lambert

Linq to Objects permet de requêter des collections d'objets en mémoire de manière simple et efficace. Là où vous utilisiez des boucles for/while/foreach pour trier, grouper ou tout autres actions plus ou moins compliquée, Linq va vous permettre d'écrire des requêtes concises et lisibles.

### Comment réaliser ma première requête Linq to Objects ? [haut]

auteur : Jérôme Lambert

Pour notre première Linq, nous allons récupérer la liste des processus actifs sur la machines triés par nom.

Avant toute chose, assurez-vous que vous avez bien installé Visual Studio 2008. Si vous n'avez pas de version officielle, vous pouvez télécharger une des versions Express de Visual Studio 2008 entièrement gratuite en vous rendant à l'adresse suivante : <http://msdn.microsoft.com/fr-fr/express/aa975050.aspx>

A présent, lancez votre version de Visual Studio 2008. Une fois l'application lancée, créez un projet de type console. Pour cela, cliquez sur le menu "File>New>Project".

Dans la boîte de dialogue qui apparaît, sélectionnez "Windows" comme type de projet et "Application Console" comme modèle.

Assurez-vous qu la cible du Framework est bien la version 3.5 car sans ça, vous ne pourrez utiliser Linq.

Appelez votre projet "MonPremierProjetLinq" et validez en appuyant sur le bouton "Ok".

Une fois le projet console créé, allez dans le fichier Program.cs et positionnez-vous dans la méthode "Main".

Comme annoncé précédemment, il nous faut récupérer la liste des processus actifs de la machine. Pour cela, le Framework .NET met à disposition la classe statique "Process" avec une méthode "GetProcesses" permettant de récupérer la liste des processus actifs d'une machine dans un tableau d'objets "System.Diagnostics.Process".

Nous allons donc écrire une requête Linq qui nous permet d'interroger la collection renvoyée par la méthode "GetProcesses" en triant par le nom du processus.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MonPremierProjetLinq2
{
    class Program
    {
        static void Main(string[] args)
        {
            var query = from process in System.Diagnostics.Process.GetProcesses()
                        orderby process.ProcessName ascending
                        select process;

            foreach (var currentItem in query.ToList())
            {
                Console.WriteLine(currentItem.ProcessName);
            }

            Console.Read();
        }
    }
}
```



Comme vous pouvez le constater la syntaxe de Linq se rapproche énormément de la syntaxe SQL.

Appuyez sur F5 pour lancer votre application et vous verrez la liste des processus actifs sur votre machine triés par ordre alphabétique croissant.

### Peut-on requêter n'importe quelle collection d'objets ? [haut]

auteur : Jérôme Lambert

Le but de Linq to Objects est de pouvoir interroger des collections d'objets en mémoire. Cependant, cela ne veut pas dire que vous pouvez interroger n'importe quel type de collection. Depuis le Framework .NET 3.5, Microsoft a introduit une interface IEnumerable<T> qui permettra d'exécuter des requêtes Linq to Objects sur toutes collections implémentant cette nouvelle interface. Heureusement pour nous, les listes et collections que vous connaissiez avec le Framework .NET 2.0 ont été mises à jour pour implémenter cette nouvelle interface.

Voici la liste des collections qui vous est possible d'utiliser avec Linq to Objects :

- Array => IEnumerable<T>
- System.Collections.Generic.List<T> => IEnumerable<T>
- System.Collections.Generic.LinkedList<T> => IEnumerable<T>
- System.Collections.Generic.Queue<T> => IEnumerable<T>
- System.Collections.Generic.Stack<T> => IEnumerable<T>
- System.Collections.Generic.HashSet<T> => IEnumerable<T>
- System.Collections.ObjectModel.Collection<T> => IEnumerable<T>
- System.ComponentModel.BindingList<T> => IEnumerable<T>
- System.Collections.Generic.Dictionary<TKey, TValue> => IEnumerable<KeyValuePair<TKey, TValue>>
- System.Collections.Generic.SortedDictionary<TKey, TValue> => IEnumerable<KeyValuePair<TKey, TValue>>
- System.Collections.Generic.SortedList<TKey, TValue> => IEnumerable<KeyValuePair<TKey, TValue>>
- System.String => IEnumerable<char>


Concrètement, vous pourrez aussi requêter vos propres collections à condition qu'elles implémentent l'interface IEnumerable<T>.

Cependant, il y a aussi les autres collections du Framework .NET qui sont non générique et donc implémentent uniquement l'interface IEnumerable.

Par exemple :

- System.Collections.ArrayList
- System.Collections.HashTable
- ...

Pourtant, vous trouverez dans cette FAQ une alternative pour utiliser Linq to Objects avec ce type de collections.

lien :  [faq](#) Comment exécuter une requête sur une collection non générique ?

Comment afficher le résultat d'une requête  
Linq dynamiquement ? [haut]

auteur : Jérôme Lambert  
Object Dumper est une bibliothèque fournie par Microsoft permettant d'afficher les résultats de vos requêtes Linq dynamiquement. Le but étant d'aller plus loin dans vos tests.

Pour en savoir plus sur cette bibliothèque et la télécharger, je vous invite à vous rendre sur MSDN.

Comment exécuter une requête sur une  
collection non générique ? [haut]

auteur : Jérôme Lambert  
Comme expliqué dans une précédente question/réponse, il est à priori possible d'interroger n'importe quel type de collections à condition qu'elle implémente l'interface IEnumerable<T>, ce qui n'est pas le cas des collections non génériques, tel que "System.Collections.ArrayList". Il semblerait donc qu'il n'est pas possible d'exécuter une requête Linq to Objects sur une collection de type ArrayList et pourtant, il y a une astuce.

Tout d'abord, il faut bien comprendre que Linq to Objects repose sur l'interface IEnumerable<T> car c'est grâce à cette interface que le compilateur va pouvoir déterminer le type d'objet contenu dans une collection. Or avec une collection de type ArrayList, il n'est pas possible de déterminer le type des objets contenus dans la collection. Pour résoudre ce problème, Microsoft a introduit une méthode d'extension appelée "Cast".

```
public static IEnumerable<T> Cast<T>(this IEnumerable source);
```

Comme vous pouvez le constater en regardant la définition de cette méthode, elle prend en paramètre une collection qui implémente l'interface IEnumerable et renvoie un objet de type IEnumerable<T>. Cette méthode est évidemment générique pour que vous puissiez spécifier explicitement le type d'objet contenu dans votre collection.

Si on considère que nous avons un ArrayList contenant des objets de type Process, nous allons pouvoir écrire le code suivant :

```
System.Collections.ArrayList myArrayList = new System.Collections.ArrayList();  
// Ajout d'instances de type Process dans notre ArrayList  
// ...  
var query = from process in myArrayList.Cast<System.Diagnostics.Process>()  
            select process;
```

Comment exécuter une requête sur une  
collection non générique contenant des  
objets de types différents ? [haut]

auteur : Jérôme Lambert  
Lorsque vous avez une collection non générique, qui n'implémente donc pas l'interface IEnumerable<T>, et qui en plus contient des objets de type différents, il n'est pas possible d'utiliser la méthode d'extension Cast<T>. Alors comment faire pour exécuter une requête Linq to Objects sur une collection de type ArrayList qui contient des objets de type Color et Point mélangés ?

Et bien tout simplement en utilisant une autre méthode d'extension nommée OfType :

```
public static IEnumerable TResult OfType<TResult>(this IEnumerable source);
```

Si on reprend notre ArrayList contenant des objets Color et Point, il va être possible d'exécuter une requête Linq pour manipuler par exemple les objets de type Color :

```
System.Collections.ArrayList myArrayList = new System.Collections.ArrayList();  
myArrayList.Add(System.Drawing.Color.Red);  
myArrayList.Add(new System.Drawing.Point(0, 0));  
myArrayList.Add(System.Drawing.Color.Green);  
myArrayList.Add(new System.Drawing.Point(10, 20));  
myArrayList.Add(System.Drawing.Color.Blue);  
myArrayList.Add(new System.Drawing.Point(20, 30));  
var query = from color in myArrayList.OfType<System.Drawing.Color>()  
            select color;  
foreach (var currentResult in query)  
    Console.WriteLine(currentResult.Name);
```

Vous remarquerez qu'à la compilation le code est accepté et mieux encore à l'exécution, aucun plantage de l'application. On retrouve d'ailleurs bien le résultat attendu :

*Red*  
*Green*  
*Blue*

Qu'est-ce qu'une séquence ? [haut]

auteur : Jérôme Lambert  
Une séquence désigne tout objet dont le type implémente l'interface IEnumerable<T> ou l'interface IQueryable<T>.

Qu'est ce que l'exécution différée de requête  
? [haut]

auteur : Jérôme Lambert  
Soit l'exemple suivant :

```
class Program  
{  
    static void Main(string[] args)  
    {  
        List<string> aMembres= new List<string>()  
        {  
            "Jérôme",  
            "Louis-Guillaume",  
            "Vincent",  
            "Benjamin"  
        };  
        var query = from membre in aMembres  
                    orderby membre ascending  
                    select membre;  
        foreach (string currentMembre in query)  
        {  
            Console.WriteLine(currentMembre);  
        }  
        Console.ReadLine();  
    }  
}
```

A l'exécution, on obtient le résultat suivant, c'est à dire liste des membres triés par ordre alphabétique :

*Benjamin*  
*Jérôme*  
*Louis-Guillaume*  
*Vincent*

Maintenant, imaginons que nous désirons ajouter un membre à notre liste juste après avoir construit notre requête. La logique voudrait que ce nouveau membre ne soit pas pris en compte si on repartcourt les éléments de notre objet "query". Le code devient donc :

```
class Program  
{  
    static void Main(string[] args)  
    {  
        List<string> aMembres= new List<string>()  
        {  
            "Jérôme",  
            "Louis-Guillaume",  
            "Vincent",  
            "Benjamin"  
        };  
        var query = from membre in aMembres  
                    orderby membre ascending  
                    select membre;  
        Console.WriteLine("Résultat query Avant");  
        Console.WriteLine("*****");
```

```
        foreach (string currentMembre in query)
        {
            Console.WriteLine(currentMembre);
        }

        aMembres.Add("Thomas");

        Console.WriteLine("\nRésultat query Après");
        Console.WriteLine("*****");
        foreach (string currentMembre in query)
        {
            Console.WriteLine(currentMembre);
        }

        Console.ReadLine();
    }
}
```

Et contre toute attente, on retrouve bien notre membre "Thomas" !

*Résultat query Avant*  
\*\*\*\*\*

Benjamin  
Jérôme  
Louis-Guillaume  
Vincent

*Résultat query Après*  
\*\*\*\*\*


Benjamin  
Jérôme  
Louis-Guillaume  
Thomas  
Vincent

C'est ça l'exécution différée de requête (ou "Deferred Query Execution" en anglais) ! Il est important de savoir que notre objet "query" ne contient absolument pas le résultat de notre requête mais plutôt l'expression permettant de récupérer un résultat une fois qu'on en aura besoin.

Ce mécanisme permet d'éviter de consommer des ressources quand ceci est inutile. Par exemple, on pourrait construire une requête permettant de récupérer la liste des membres et plus tard, demander le premier de ces éléments. Dans ce cas, on évite de récupérer tous les membres pour ne prendre que le premier.

Qu'est-ce que la réutilisation de requête ? [haut]

auteur : Jérôme Lambert

Avant de commencer la lecture de cette réponse, allez consulter  [FAQ](#) Qu'est-ce que l'exécution différée de requête ?

La réutilisation d'une requête (ou "Reuse Query" en anglais) est le fait de réutiliser une requête différentes fois dans une application. Cependant, il n'est pas impossible qu'une même requête donne des résultats différents entre deux exécutions. En voici un exemple :

```
class Program
{
    static void Main(string[] args)
    {
        List<string> aMembres = new List<string>()
        {
            {
                "Jérôme",
                "Louis-Guillaume",
                "Vincent",
                "Benjamin"
            }
        };

        var query = from membre in aMembres
                    orderby membre ascending
                    select membre;

        Console.WriteLine("Résultat query Avant");
        Console.WriteLine("*****");
        foreach (string currentMembre in query)
        {
            Console.WriteLine(currentMembre);
        }

        // Mise à jour des membres
        aMembres[0] = aMembres[0] + " Lambert";
        aMembres[1] = aMembres[1] + " Morand";
        aMembres[2] = aMembres[2] + " Lainé";
        aMembres[3] = aMembres[3] + " Broux";

        Console.WriteLine("\nRésultat query Après");
        Console.WriteLine("*****");
        foreach (string currentMembre in query)
        {
            Console.WriteLine(currentMembre);
        }

        Console.ReadLine();
    }
}
```


Ce qui donne le résultat suivant :

*Résultat query Avant*  
\*\*\*\*\*

Benjamin  
Jérôme  
Louis-Guillaume  
Vincent

*Résultat query Après*  
\*\*\*\*\*

Benjamin Broux  
Jérôme Lambert  
Louis-Guillaume Morand  
Vincent Lainé

lien :  [FAQ](#) Qu'est-ce que l'exécution différée de requête ?

Qu'est-ce qu'un opérateur de requête ? [haut]

auteur : Jérôme Lambert


Les opérateurs de requêtes (ou "Query Operators" en anglais) sont un ensemble de méthodes d'extension qui rendent possible les possibilités de Linq. Ainsi, vous bénéficiez de requêtes permettant le tri, le filtrage, l'aggrégation, la concaténation, la projection, etc.

Voici un exemple de requête Linq utilisant trois opérateurs de requête :

```
var query = System.Diagnostics.Process.GetProcesses()
    .Where(p => p.ProcessName.ToLower().StartsWith("a"))
    .OrderBy(p => p.ProcessName)
    .Select(p => p.ProcessName);
```

Le premier opérateur de requête est **Where** qui permet de filtrer uniquement les processus dont le nom commence par 'a'.  
Le second opérateur de requête est **OrderBy** qui permet de trier la séquence par nom de processus ascendant.  
Le troisième et dernier opérateur est **Select** qui permet de sélectionner pour chaque élément de la séquence résultante uniquement le nom du processus.

 Ces méthodes d'extensions se trouvent dans la classe *System.Linq.Enumerable*.

lien :  [Lien MSDN vers System.Linq.Enumerable](#)

Quels sont les opérateurs de requête standards disponibles ? [haut]


auteur : Jérôme Lambert

Type	Opérateur de requête
Tri des données	OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse
Opérations ensemblistes	Distinct, Except, Intersect, Union
Filtrage des données	OfType, Where
Opérations de quantificateur	All, Any, Contains
Opérations de projection	Select, SelectMany
Partitionnement des données	Skip, SkipWhile, Take, TakeWhile
Opérations de jointure	Join, GroupJoin
Regroupement de données	GroupBy, ToLookup
Opérations de génération	DefaultIfEmpty, Empty, Range, Repeat
Opérations d'égalité	SequenceEqual
Opérations d'élément	ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last,

	LastOrDefault, Single, SingleOrDefault
Conversion de types de données	AsEnumerable, AsQueryable, Cast, OfType, ToArray, ToDictionary, ToList, ToLookup
Opérations de concaténation	Concat
Opérations d'agrégation	Aggregate, Average, Count, LongCount, Max, Min, Sum

Qu'est-ce qu'une expression de requête (Query Expression) ? [haut]

auteur : Jérôme Lambert

Les expressions de requêtes (ou "Query Expressions" en anglais) représentent une réelle extension du langage. Comme vous devez le savoir, les opérateurs standards de requête ( faq Qu'est-ce qu'un opérateur de requête ?) sont un ensemble de méthodes statiques introduites avec C# 3.0 et VB.NET 9.0 pour permettre de construire des requêtes Linq sur des sources de données.

Ainsi, une requête Linq avec les opérateurs standards de requête ressemblera à ceci :

```
var query = System.Diagnostics.Process.GetProcesses()
    .Where(p => p.ProcessName.StartsWith("A"))
    .OrderBy(p => p.ProcessName)
    .Select(p => p);
```


La lecture de cette requête s'avère encore aisée grâce à sa simplicité mais imaginez une requête avec des jointures sur différentes collections, des groupements... Ce serait plus facile à écrire mais surtout à lire en langage SQL ! Et bien les expressions de requête c'est un peu ça car cela va vous permettre d'écrire vos requête Linq un peu comme vous écririez vos requêtes SQL.


L'exemple précédent peut donc s'écrire de la manière suivante en expression de requête :

```
var query = from process in System.Diagnostics.Process.GetProcesses()
            where process.ProcessName.StartsWith("A")
            orderby process.ProcessName ascending
            select process;
```

Comme vous pouvez le remarquer, la lecture se fait presque comme si on lisait une requête SQL et lorsque je vous parlais tout au début que les expressions de requête étaient une extension du langage, on est en plein dedans : from, where, orderby, ascending, select, ... Des nouveaux mots clés ont été introduits pour permettre cette nouvelle syntaxe. Lors de la compilation, le compilateur s'occupera de retranscrire vos expressions de requête en opérateurs de requête.

Cependant, tous les opérateurs standards de requête n'ont pas nécessairement leur correspondance en tant qu'expression de requête et ça dépend aussi du langage. Par exemple, l'opérateur "Take" n'existe pas en C# en tant qu'expression de requête, alors qu'en VB.NET il existe.

lien : faq Qu'est-ce qu'un opérateur de requête ?

lien : faq Est-ce que tous les opérateurs standards de requête ont leur équivalence en tant qu'expression de requête ?

Est-ce que tous les opérateurs standards de requête ont leur équivalence en tant qu'expression de requête ? [haut]

auteur : Jérôme Lambert

C# 3.0 et VB.NET 9.0 étant développés par deux équipes différentes, tous les opérateurs standards de requête n'ont pas nécessairement leur équivalence dans les deux langages en tant qu'expression de requête. Vous trouverez ci-dessous un tableau répertoriant les opérateurs standards avec une indication s'ils sont supportés en tant qu'expression de requête.

Opérateur standard	Equivalence en expression de requête avec C#	Equivalence en expression de requête avec VB.NET
All	Non	Oui
Any	Non	Oui
Average	Non	Oui
Cast	Oui	Oui
Count	Non	Oui
Distinct	Non	Oui
GroupBy	Oui	Oui
GroupJoin	Oui	Oui
Join	Oui	Oui
LongCount	Non	Oui
Max	Non	Oui
Min	Non	Oui
OrderBy	Oui	Oui
OrderByDescending	Oui	Oui
Select	Oui	Oui
SelectMany	Oui	Oui
Skip	Non	Oui
SkipWhile	Non	Oui
Sum	Non	Oui
Take	Non	Oui
TakeWhile	Non	Oui
ThenBy	Oui	Oui
ThenByDescending	Oui	Oui
Where	Oui	Oui

Comment utiliser Linq to Objects dans vos projets ? [haut]

auteur : Jérôme Lambert

1. Assurez-vous que votre projet cible bien le Framework .NET 3.5
2. Ajoutez à votre projet la référence à l'assembly *System.Core.dll*, si ce n'est déjà fait.
3. Ajoutez *using System.Linq*; en début de fichier de votre classe

Qu'est-ce que la "sugar syntax" en Linq ? [haut]

auteur : Jérôme Lambert


Avec Linq, vous pouvez écrire vos requêtes à l'aide des opérateurs de requêtes disponibles grâce aux méthodes d'extensions de *System.Linq.Enumerable*. Ainsi, si on désire récupérer la liste des processus actifs sur votre ordinateur et commençant par la lettre 'A', il suffit d'écrire :

```
var query = System.Diagnostics.Process.GetProcesses().Where(p => p.ProcessName.ToLower().Sta
```

Cependant pour des cas complexes, cette syntaxe n'est pas toujours facile à écrire et encore moins à lire. C'est là qu'interviennent les expressions de requêtes. C'est une autre façon d'écrire une requête Linq bien plus proche de la syntaxe SQL. On appelle aussi cette façon d'écrire la "sugar syntax".

Ainsi, la requête précédente s'écrit en version expression de requête :

```
var query = from process in System.Diagnostics.Process.GetProcesses()
            where process.ProcessName.ToLower().StartsWith("a")
            select process;
```

 Il est important de ne pas oublier que les opérateurs de requêtes n'ont pas warning tous leur correspondance en expression de requête.

Comment grouper des éléments sur base de critères multiples ? [haut]

auteur : Jérôme Lambert

Il vous est possible de faire un groupement sur plusieurs champs en passant par un objet anonyme de la manière suivante :

```
static void Main(string[] args)
{
    var membres = GetMembres();

    var query = from membre in membres
                group membre by new { membre.Sexe, membre.Age }
                into grouping
                select new
                {
                    Sexe = grouping.Key.Sexe,
                    Age = grouping.Key.Age,

```

```
        Membres = grouping
    };

    foreach (var currentResult in query)
    {
        Console.WriteLine("Membres dont sexe est '{0}' et âge = '{1}'", currentResult.Sexe, currentResult.Age);
        foreach (var currentMembre in currentResult.Membres)
        {
            Console.WriteLine(currentMembre.Nom);
        }
        Console.WriteLine();
    }

    Console.Read();
}
```


Ce qui donnera le résultat suivant :

*Membres dont sexe est 'M' et âge = '30'*  
Marc Lussac  
Jérôme Lambert  
nico-pyright(c)  
Thomas Lebrun  
tomlev

*Membres dont sexe est 'M' et âge = '20'*  
Yogui  
Louis-Guillaume Morand  
Tofalu

*Membres dont sexe est 'F' et âge = '20'*  
Aspic  
Dev01  
The\_badger\_man

*Membres dont sexe est 'F' et âge = '30'*  
Skyyounet

 Les sexes et âges affichés dans le résultat de la requête ne reflètent en aucun warning cas la réalité, excepté pour Skyyounet :)

Comment rendre paramétrable un critère de filtre ? [\[haut\]](#)

Si par exemple, nous désirons récupérer tous les membres du sexe masculin de developpez.com, nous pouvons écrire la requête suivante :

```
var query = from membre in membres
            where membre.Sexe == 'N'
            select membre;
```

Seul soucis, c'est que si nous désirons récupérer la liste des membres du sexe féminin, à part récrire la requête, celle du-dessus est inutilisable. Cependant, rien empêche d'utiliser des variables dans nos requêtes Linq, ainsi, on va pouvoir encapsuler notre requête Linq dans une méthode qui reçoit en paramètre le sexe afin de le comparer dans la requête Linq.

```
static void Main(string[] args)
{
    var MembresHommes = GetMembres('M');
    var MembresFemmes = GetMembres('F');
}

static List<Membre> GetMembres(char sexe)
{
    var membres = GetMembres();

    var query = from membre in membres
                where membre.Sexe == sexe
                select membre;

    return query.ToList();
}
```

[Précédent](#) [Sommaire](#) 

[Version PDF \(Miroir\)](#) [Version hors-ligne \(Miroir\)](#)

[Consultez les autres F.A.Q's](#)

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright © 2008 Developpez Developpez LLC. Tous droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site ni de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.