6

Délégués et traitement d'événements

Dans ce chapitre, nous allons nous intéresser aux événements. Ceux-ci jouent un rôle considérable en programmation Windows et Web : lorsqu'il se passe quelque chose dans un programme (par exemple un clic sur un bouton mais aussi pour bien d'autres motifs), Windows nous signale un événement. Et pour cela, il appelle une fonction bien particulière du programme.

Un programme, qu'il soit Web ou Windows, consiste essentiellement à traiter des événements.

Événements et traitements d'événements ne sont cependant pas propres ou liés exclusivement à la programmation Windows ou Web. Il s'agit, avec les délégués et les événements, de mécanismes généraux dont n'importe quel programme peut tirer profit.

Introduisons d'abord le sujet avec les délégués puisque les événements sont fondés sur eux.

C# version 2 a également introduit les fonctions anonymes (voir la section 6.3).

6.1 Les délégués

Un délégué (*delegate* en anglais) est un objet (en fait, de la classe `Delegate` ou d'une classe dérivée de celle-ci, mais peu importe) qui permet d'appeler une fonction, qui n'est pas nécessairement toujours la même. Il peut même s'agir d'un appel à une série de fonctions. Un délégué présente des similitudes avec les pointeurs de fonction du C/C++ mais

permet d'aller plus loin (car un délégué peut exécuter plusieurs fonctions les unes à la suite des autres) tout en faisant preuve de moins de laxisme.

Pour ceux qui connaissent le C et/ou C++, rappelons qu'un pointeur de fonction désigne une variable qui fait référence à une fonction. À partir d'un pointeur de fonction, on peut appeler la fonction pointée.

Prenons un exemple simple, que nous commenterons ensuite :

Les délégués Ex1.cs

```
using System;
class Prog
{
 static void f() {Console.WriteLine("Fonction f");}
 static void g() {Console.WriteLine("Fonction g");}
 delegate void T();       // ceci est une définition de type
 static void Main()
 {
  T de = new T(f);        // déclaration de variable. de fait référence à f
  de();                      // exécute f
  de = new T(g);          // de fait maintenant référence à g
  de();                   // exécute g
 }
}
```

Rien de nouveau en ce qui concerne les fonctions `f`, `g` et `Main`. Elles sont qualifiées de `static` car, pour simplifier, nous ne créons aucun objet.

Nous créons ensuite un nouveau type, appelé ici `T`. Ne vous fiez pas aux apparences : `T` désigne un type d'objet (au même titre que `int`, `double` ou une classe) et non une déclaration de variable, une signature de fonction, ou quoi que ce soit d'autre, comme un coup d'œil rapide pourrait le faire croire. Dans la définition de `T`, nous avons le choix de son nom (`T`) et nous signalons, avec le mot réservé `delegate`, qu'il s'agit d'un type délégué. `de` est donc une variable de type délégué. On parle plus communément de délégué pour `de`.

Un objet de type `T`, comme `de`, peut faire référence à une ou plusieurs fonctions, mais avec cette restriction : ces fonctions doivent respecter la signature reprise dans la définition qu'est `delegate void T()`. Ici : aucun argument (parenthèses vides) et aucune valeur de retour (`void`). C'est pour cette raison que la ligne `delegate` ressemble à une signature de fonction.

Il devient alors possible d'exécuter cette ou ces fonctions via le délégué.

Avec :

```
T de = new T(f);
```

nous déclarons et créons une variable baptisée `de` et de type `T`. Comme nous spécifions la fonction `f` en argument du constructeur, notre objet `de` fait référence à la fonction `f`.