A la découverte des événements en .NET

par Frederic Cantenot Olivier Delmotte

Date de publication : 25/10/2005

Dernière mise à jour :

Ce petit tutorial à pour but de montrer simplement comment créer, générer et intercepter vos événement en C#.

I - Introduction

Je pense que presque tout le monde connaît les événements. A chaque fois que vous cliquez sur un bouton dans une application Windows vous déclenchez un événement. Ici, le but est de créer des événements qui vont être propres à votre application. C'est-à-dire déclencher une action dans votre application lorsque quelque chose se produit et informer les objets abonnés à cet événement.

II - Créer votre événement à l'aide de la classe EventArgs

La classe *EventArgs* est la classe de base pour créer vos événements. A chaque fois que vous voulez créer un événement, il vous faut dériver une classe de *EventArgs*.

Membres de la classe *EventArgs* :

Constructeurs publics

Nom	Definition
EventArgs, constructeur Pris en charge par le .NET Compact Framework.	Initialise une nouvelle instance de la classe *EventArgs*.

Champs publics

Nom	Definition
Equals Pris en charge par le .NET Compact Framework.	Surchargé. Détermine si deux instances de Object sont égales.

Méthodes publiques

Nom	Definition
EventArgs, constructeur Pris en charge par le .NET Compact Framework.	Initialise une nouvelle instance de la classe *EventArgs*.
GetHashCode (hérité de Object) Pris en charge par le .NET Compact Framework.	Sert de fonction de hachage pour un type particulier, adapté à une utilisation dans des algorithmes de hachage et des structures de données telles qu'une table de hachage.
GetType (hérité de Object) Pris en charge par le .NET Compact Framework.	Obtient le Type de l'instance en cours.
ToString (hérité de Object)	Retourne un objet String qui représente l' Object en cours.

Nom	Definition
Pris en charge par le .NET Compact Framework.	

Notre événement va nous servir à afficher un message toutes les secondes.

Nous allons déclarer la classe comme ceci :

Langage C#

```
using System;

namespace TutoEvent
{
	/// <summary>
	/// A la responsabilité de contenir le text de l'évenement et de le rendre accessible
	/// </summary>
	public class GenerateTextEventArgs : EventArgs
	{
		private string myEventText = null;

		public GenerateTextArgs(string theEventText)
		{
			if (theEventText == null) throw new NullReferenceException();
			myEventText = theEventText;
		}

		public string EventText
		{
			get { return this.myEventText; }
		}
	}
}
```

Langage VB.NET

```
Imports System
''' <summary>
''' A la responsabilité de contenir le text de l'évenement et de le rendre accessible
''' </summary>
Public Class GenerateTextArgs
	Inherits EventArgs
	Private myEventText As String = Nothing
	Public Sub New(ByVal theEventText As String)
		If theEventText Is Nothing Then
			Throw New NullReferenceException()
		End If
		myEventText = theEventText
	End Sub
	Public ReadOnly Property EventText As String
		Get
			Return Me.myEventText
		End Get
	End Property
End Class
```

Notre classe s'appelle *GenerateTextEventArgs*. Par convention le nom d'une classe heritant de *EventArgs* devrait toujours se terminer par *EventArgs*. Elle contient comme donnée privée un attribut de type string myEventText qui est le texte que nous allons envoyer. Dans cette classe vous pouvez placer toutes les informations que vous souhaitez envoyer. Cela peut être aussi bien une string qu'un int qu'une bitmap... Ensuite il vous faut un ou plusieurs constructeurs pour construire vos événements. Et finalement il vous faut des accesseurs sur les données que vous voulez récupérer dans le *gestionnaire d'évenéments ("Event Handler")*.

Donc pour construire vos événements il suffit de suivre ce principe :

- Déclarer une classe héritant de *EventArgs*

- Ajouter les informations que vous souhaitez comme attribut de la classe

- Déclarer les accesseurs correspondants

III - Générer un événement

A - Le type delegate

Il n'est pas possible de parler des événements sans parler des *delegate*. Un *delegate* est un objet qui permet d'appeler une fonction ou une série de fonction. Un *delegate* est similaire aux pointeurs de fonctions du C/C++.

Une variable *delegate* va permettre d'exécuter une fonction ou plusieurs fonctions. Pour cela le *delegate* va stocker des références sur des méthodes (que nous appellerons un *gestionnaire d'évenéments ("Event handler")*).

La signature des méthodes référencées devra respecter les règles suivantes :

- retourner *void*

- prendre comme premier paramètre un type *object* que nous appellerons généralement *sender*

- prendre comme second paramètre un objet héritant de *EventArgs*, donc dans notre cas un objet *GenerateTextEventArgs*.

Note : il est possible de voir des *delegate* déclarer autrement. Ici je montre simplement la méthode généralement utilisée pour les événements.

Pour déclarer un *delegate*, nous utilisons la syntaxe suivante :

Langage C#

```
public delegate void TextGeneratedEventHandler (object sender, GenerateTextEventArgs e);
```

Langage VB.NET

```
Public Delegate Sub TextGeneratedEventHandler (ByVal sender As Object, ByVal e As GenerateTextEventArgs)
```

Vous pouvez prendre n'importe quel nom pour le *delegate*. Toutefois il est judicieux de prendre un nom parlant.

B - Le type event

Il nous faut ensuite déclarer un objet *event* du type du *delegate* déclaré plus haut. Le mot clé *event* vous permet de spécifier un délégué à appeler lors de l'occurrence d'un certain événement dans votre code.

Pour déclarer un *event*, nous utilisons la syntaxe suivante :

Langage C#

```
public event TextGeneratedEventHandler OnTextChanged;
```

Langage VB.NET

```
Public Event OnTextChanged As TextGeneratedEventHandler
```

Une fois de plus, vous pouvez prendre le nom que vous voulez pour votre événement. Le nom *OnTextChanged* est le nom qui va apparaître dans l'intellisense lorsque vous allez vous abonner à l'événement.

C - Générer l'événement

Pour générer un événement il suffit d'appeler son constructeur avec les paramètres éventuels comme ceci :

Langage C#

```
GenerateTextEventArgs e = new GenerateTextEventArgs("Compteur = " + i.ToString());
```

Langage VB.NET

```
Dim e As GenerateTextEventArgs = New GenerateTextEventArgs("Compteur = " & i.ToString())
```

Puis il nous reste à envoyer cet événement à tout le monde :

Langage C#

```
if (e != null) OnTextChanged(this,e);
```

Langage VB.NET

```
If Not (e Is Nothing) Then
    RaiseEvent OnTextChanged(Me,e)
End If
```

Voici la classe complète qui regroupe tout ce que nous venons de voir :

Langage C#

```
using System;
using System.Threading;

namespace TutoEvent
{
	/// <summary>
	/// A la responsabilité d'envoyer un evenement GenerateTextEvent toutes les secondes
	/// </summary>
	public class GenerateText
	{
		/// <summary>
		/// Declare un delegate
		/// </summary>
		public delegate void TextGeneratedEventHandler(object sender, GenerateTextEventArgs e);
		/// <summary>
		/// Declare un evenement qui va contenir les informations que nous souhaitons envoyer
		/// </summary>
		public event TextGeneratedEventHandler OnTextChanged;

		public GenerateText(){}

		public void Start(int theNumber)
		{
			int i = 0;
			while (i  < theNumber)
			{
				GenerateTextEventArgs e = new GenerateTextEventArgs("Compteur = " + i.ToString());
				if (e != null) OnTextChanged(this,e);
				Thread.Sleep(1000);
				i++;
			}
		}
	}
}
```

Langage VB.NET

```
Imports System
Imports System.Threading
''' <summary>
''' A la responsabilité d'envoyer un evenement GenerateTextEvent toutes les secondes
''' </summary>
Public Class GenerateText
	''' <summary>
	''' Declare un delegate
	''' </summary>
	Public Delegate Sub TextGeneratedEventHandler (ByVal sender As Object , ByVal e As GenerateTextEventArgs)
	''' <summary>
	''' Declare un evenement qui va contenir les informations que nous souhaitons envoyer
	''' </summary>
	Public Event OnTextChanged As TextGeneratedEventHandler
	Public Sub New()
```

```
        End Sub
        Public Sub Start(int theNumber)
            Dim i as Integer  = 0
            while i < theNumber
                Dim e As GenerateTextEventArgs = New GenerateTextEventArgs("Compteur = " & i.ToString())
                If Not(e Is Nothing) Then
                    RaiseEvent OnTextChanged(Me,e)
                End
                Thread.Sleep(1000)
                i = i + 1;
            End While
        End Sub
    End Class
```

IV - Récupérer un événement dans un gestionnaire d'évenéments

Un *gestionnaire d'évenéments ("Event Handler")* est la méthode qui va s'exécuter en réponse à l'événement. Un *Event hanlder* retourne normalement *void* et accepte 2 paramètres qui sont :

- le *sender* : l'objet dans lequel l'événement s'est produit.

- Un argument de type *EventArgs* qui contient les informations relatives à l'événement.

Pour récupérer un événement, la première chose à faire est de se placer à l'écoute de cet événement. C'est là que le *delegate* que nous avons déclaré plus haut trouve toute son utilité.

La syntaxe pour ajouter un *gestionnaire d'évenéments ("Event handler")* à l'écoute d'un événement est la suivante :

Langage C#

```
private GenerateText myTextGenerator = new GenerateText();
myTextGenerator.OnTextChanged+=new TutoEvent.GenerateText.TextGeneratedEventHandler(myTextGenerator_MonEvenement);
```

Note : Pour aller plus vite sous visual studio, apres les signes += vous pouvez utiliser la touche tabulation 2 fois de suite. Celle-ci générera tout ce qui est nécessaire après les signes +=.

Langage VB.NET

```
AddHandler myTextGenerator.OnTextChanged, AddressOf myTextGenerator_MonEvenement
```

Note : En VB.Net, vous pouvez associer directement une procédure à un ou plusieurs évènements comme suit, a condition de déclarer le fournisseur d'évènements à l'aide de WithEvents :

```
Private WithEvents myTextGenerator As GenerateText = New GenerateText()
Private myTextGenerator_TextGenerated (ByVal sender As Object , ByVal e As GenerateTextEventArgs) Handles
myTextGenerator.OnTextChanged
```

La premiere ligne sert à déclarer un objet *GenerateText*. Comme nous l'avons dit plus haut, le *delegate* stock des références sur des méthodes. La seconde ligne ajoute une référence sur la méthode (donc le *gestionnaire d'évenéments ("Event Handler")*) *myTextGenerator_MonEvenement*. Cela signifie que lorsque l'événement *OnTextChanged* va être envoyé, toutes les méthodes référencées dans le *delegate* vont en être averties. **Donc le code qui doit être exécuté en réponse à un événement doit se trouver dans cette méthode.**

Note : pour qu'une méthode ne soit plus à l'écoute d'un événement, il faut employer la syntaxe suivante :

Langage C#

```
myTextGenerator.OnTextChanged-=new TutoEvent.GenerateText.TextGeneratedEventHandler(myTextGenerator_MonEvenement);
```

Langage VB.NET

```
RemoveHandler myTextGenerator.OnTextChanged, AddressOf myTextGenerator_MonEvenement
```

V - Conclusion

Pour conclure nous allons simplement récapituler les étapes à suivre :

1. Définir la classe qui va hériter de *EventArgs* et qui va contenir les informations à envoyer.

2. Créer un *delegate* qui va stocker une référence sur le *gestionnaire d'évenéments ("Event Handler")* c'est à dire la méthode qui va recevoir et traiter l' événement.

3. Définir l'événement comme un objet de type *delegate*

4. Définir une méthode qui va signaler aux objets enregistrés qu'un événement s'est produit.

5. Appeler la méthode définie en 4 lorsque l'événement se produit.

VI - Téléchargement

Source de l'article en C# et VB.NET

Article au format PDF

VII - Liens

MSDN - Didacticiel sur les délégués

MSDN - Didacticiel sur les événements

VIII - Remerciements

Merci à Olivier Delmotte pour la traduction des sources en VB.NET ainsi qu'à Jean-Alain Baeyens et Laurent Dardenne pour la relecture de l'article.