

# **Chapitre 6**

**Gestion du clavier, de l'écran et de la souris**



## CONTENU DU CHAPITRE 6

<b>6.</b>	<b><i>Gestion du clavier, de l'écran et de la souris</i></b>	<b>6-1</b>
<b>6.1.</b>	<b>La classe MessageBox</b>	<b>6-1</b>
6.1.1.	MessageBox, Aide	6-1
6.1.2.	Simple message	6-2
6.1.3.	message avec titre	6-2
6.1.4.	message avec titre et boutons Oui/Non	6-3
6.1.5.	message avec titre et boutons Oui/Non/Annuler	6-3
6.1.6.	Message et boutons recommencer, annuler	6-4
<b>6.2.</b>	<b>La fonction InputBox</b>	<b>6-5</b>
<b>6.3.</b>	<b>Gestion du clavier</b>	<b>6-6</b>
6.3.1.	Les événements du clavier	6-6
6.3.1.1.	La propriété KeyPreview d'une feuille	6-7
6.3.2.	L'événement KeyPress	6-7
6.3.2.1.	Argument e de l'événement KeyPress	6-8
6.3.3.	Les événements KeyDown et KeyUp	6-10
6.3.3.1.	Aide événement KeyDown	6-10
6.3.3.2.	Exemple événement KeyDown	6-10
6.3.3.3.	Propriétés de KeyEventArgs	6-11
6.3.3.4.	Touches en relation avec KeyDown et KeyUp	6-12
6.3.3.5.	Exemple d'utilisation de KeyDown et KeyUp	6-13
<b>6.4.</b>	<b>Gestion de la souris</b>	<b>6-15</b>
6.4.1.	Réponses aux événements de la souris	6-15
6.4.1.1.	Appartenance des événements de la souris	6-15
6.4.1.2.	Arguments des événements de la souris	6-16
6.4.1.3.	Événements souris, exemple 1	6-17
6.4.1.1.	Événements souris, exemple 2	6-19
6.4.1.2.	Événements souris, exemple 3	6-20
6.4.2.	Informations sur la souris	6-22
6.4.2.1.	Événements souris, exemple 4	6-23
<b>6.5.</b>	<b>La classe Graphics</b>	<b>6-25</b>
6.5.1.	La méthode CreateGraphics des contrôles	6-25
6.5.2.	L'objet Pen	6-25
6.5.2.1.	Liste de surcharge de Pen	6-26
6.5.2.2.	Propriétés de Pen	6-26
6.5.2.3.	Exemple utilisation des propriétés de Pen	6-27
6.5.2.4.	Pen, propriété Color	6-28
6.5.2.5.	Pen, propriété Width	6-28
6.5.2.6.	Pen, propriété DashStyle	6-29
6.5.2.7.	Pen, exemple DashStyle, custom	6-29
6.5.3.	La classe ou structure Point	6-30
6.5.3.1.	Point, méthode ToString	6-30
6.5.4.	La classe ou structure PointF	6-31
6.5.4.1.	PointF, méthode ToString	6-31

<b>6.6.</b>	<b>les membres de la classe Graphics</b>	<b>6-32</b>
6.6.1.	Vue d'ensemble des méthodes	6-32
6.6.2.	Graphics, méthode DrawLine	6-33
6.6.2.1.	DrawLine, Forme (Pen, Point, Point)	6-33
6.6.2.2.	DrawLine, Forme (Pen, PointF, PointF)	6-33
6.6.2.3.	DrawLine, Forme (Pen, Int32, Int32, Int32, Int32)	6-34
6.6.2.4.	DrawLine, Forme (Pen, Single, Single, Single, Single, Single)	6-34
6.6.2.5.	DrawLine exemple	6-34
6.6.3.	Graphics, méthode DrawRectangle	6-36
6.6.3.1.	DrawRectangle, Forme (Pen, Rectangle)	6-36
6.6.3.2.	DrawRectangle, Forme (Pen, Int32, Int32, Int32, Int32)	6-38
6.6.3.3.	DrawRectangle, Forme (Pen, Single, Single, Single, Single, Single)	6-38
6.6.3.4.	DrawRectangle, exemple	6-38
6.6.3.5.	Utilisation de la structure Rectangle	6-40
6.6.4.	Graphics, méthode DrawEllipse	6-41
6.6.4.1.	DrawEllipse, Forme (Pen, Rectangle)	6-41
6.6.4.2.	DrawEllipse, Forme (Pen, RectangleF)	6-41
6.6.4.3.	DrawEllipse, Forme (Pen, Int32, Int32, Int32, Int32)	6-42
6.6.4.4.	DrawEllipse, Forme (Pen, Single, Single, Single, Single, Single)	6-42
6.6.4.5.	DrawEllipse, exemple	6-42
6.6.5.	Illustration du dessin d'une ellipse	6-43
<b>6.7.</b>	<b>Graphiques au niveau du pixel, classe BitMap</b>	<b>6-44</b>
6.7.1.	Méthode SetPixel	6-44
6.7.1.1.	SetPixel, exemple	6-44
6.7.1.2.	SetPixel, exemple de résultat	6-46
<b>6.8.</b>	<b>La methode DrawString</b>	<b>6-47</b>
6.8.1.	DrawString, paramètre String	6-47
6.8.2.	DrawString, paramètre Font	6-47
6.8.2.1.	Font (Nom, taille)	6-48
6.8.2.2.	Font (Nom, taille, style)	6-48
6.8.3.	DrawString, parametre Brush	6-48
6.8.4.	DrawString, paramètre StringFormat	6-49
6.8.5.	DrawString, positionnement du texte	6-49
6.8.5.1.	Positionnement du texte : Single, Single	6-49
6.8.5.2.	Positionnement du texte : PointF	6-49
6.8.5.3.	Positionnement du texte : RectangleF	6-49
6.8.6.	DrawString, exemple1	6-50
6.8.6.1.	DrawString, exemple1, résultat	6-51
6.8.7.	DrawString, exemple2	6-51
6.8.7.1.	DrawString, exemple2, résultat	6-52
<b>6.9.</b>	<b>Les boîtes de dialogue communes</b>	<b>6-53</b>
6.9.1.	Mise en place des contrôles	6-53
6.9.2.	Le contrôle FolderBrowserDialog	6-54
6.9.2.1.	Utilisations des résultats	6-54
6.9.3.	Le contrôle OpenFileDialog	6-56
6.9.3.1.	Filter, propriété (OpenFileDialog)	6-57
6.9.3.2.	Utilisations des résultats	6-58
6.9.3.3.	InitialDirectory, propriété	6-58
6.9.3.4.	OpenFile, méthode	6-59
6.9.4.	Le contrôle SaveFileDialog	6-60
6.9.5.	Le contrôle ColorDialog	6-61

6.9.5.1.	Utilisation du résultats	6-61
6.9.5.2.	Configuration de la boîte de dialogue couleur	6-62
6.9.6.	Le contrôle FontDialog	6-63
6.9.6.1.	Utilisation du résultats	6-63
<b>6.10.</b>	<b>Les boîtes d'impression</b>	<b>6-64</b>
6.10.1.	PrintDocument, vue d'ensemble	6-64
6.10.2.	PrintDocument, Aide	6-65
6.10.3.	Application pour les exemples	6-65
6.10.4.	Utilisation du composant PrintDocument	6-66
6.10.5.	Le contrôle PrintDialog	6-66
6.10.5.1.	La propriété PrinterSettings	6-67
6.10.6.	Le contrôle PrintPreviewDialog	6-67
6.10.7.	Le contrôle PageSetupDialog	6-68
6.10.7.1.	La propriété PageSettings	6-68
6.10.8.	Impression du document	6-69
<b>6.11.</b>	<b>Conclusion</b>	<b>6-70</b>
<b>6.12.</b>	<b>Historique</b>	<b>6-70</b>
6.12.1.	Version 1.0 juin 2016	6-70
6.12.2.	Version 1.0 B juin 2016	6-70



## 6. GESTION DU CLAVIER, DE L'ÉCRAN ET DE LA SOURIS

Dans ce chapitre nous allons étudier les méthodes permettant d'envoyer des messages à l'utilisateur (MessageBox). De plus nous étudierons la gestion du clavier et de la souris. Nous aborderons aussi les classes et méthodes permettant d'afficher et de dessiner. Et enfin nous étudierons quelques boîte de dialogue comme par exemple le ColorDialog.

### 6.1. LA CLASSE MESSAGEBOX

C'est une classe avec des méthodes statiques, en particulier la méthode Show.

#### 6.1.1. MESSAGEBOX, AIDE

Avec F1 en ayant sélectionné MessageBox, nous obtenons :

### MessageBox, classe

Cet article a été traduit automatiquement. Si vous disposez d'une connexion Internet, sélectionnez Afficher cette rubrique en ligne pour consulter cette page dans un mode d'édition présentant simultanément le contenu original en anglais.

[Envoyer des commentaires](#) à Microsoft sur cette rubrique.  Afficher cette rubrique [en ligne](#) dans le navigateur par défaut.

Affiche une fenêtre de message, également appelée une boîte de dialogue qui affiche un message à l'utilisateur. Il s'agit d'une fenêtre modale, bloquant les autres actions dans l'application jusqu'à ce que l'utilisateur la ferme. Une **MessageBox** peut contenir du texte, des boutons et des symboles donnant des informations et des instructions à l'utilisateur.

---







#### ▲ Hiérarchie d'héritage

[System.Object](#)  
**System.Windows.Forms.MessageBox**

**Espace de noms :** [System.Windows.Forms](#)  
**Assembly :** System.Windows.Forms (dans System.Windows.Forms.dll)

Pour obtenir l'affichage il faut utiliser la méthode **Show** qui possède une grande liste de surcharge, dont voici une partie :

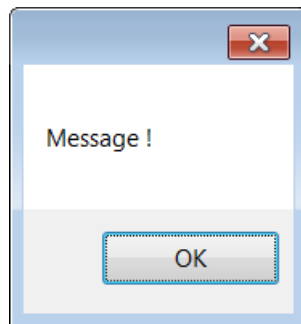
<b>S</b>	<a href="#">Show(String)</a>	Affiche une boîte de message avec le texte spécifié.
<b>S</b>	<a href="#">Show(IWin32Window, String)</a>	Affiche une boîte de message devant l'objet spécifié et avec le texte spécifié.
<b>S</b>	<a href="#">Show(String, String)</a>	Affiche une boîte de message avec le texte et la légende spécifiés.
<b>S</b>	<a href="#">Show(IWin32Window, String, String)</a>	Affiche une boîte de message devant l'objet spécifié, et avec le texte et la légende spécifiés.
<b>S</b>	<a href="#">Show(String, String, MessageBoxButtons)</a>	Affiche une boîte de message avec le texte, la légende et le texte spécifiés.

 	<code>Show(IWin32Window, String, String, MessageBoxButtons, MessageBoxIcon)</code>	Affiche une boîte de message devant l'objet spécifié, et avec le texte, la légende, les boutons et l'icône spécifiés.
 	<code>Show(String, String, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton)</code>	Affiche une boîte de message avec le texte, la légende, les boutons, l'icône et le bouton par défaut spécifiés.
 	<code>Show(IWin32Window, String, String, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton)</code>	Affiche une boîte de message devant l'objet spécifié, et avec le texte, la légende, les boutons, l'icône et le bouton par défaut spécifiés.

### 6.1.2. SIMPLE MESSAGE

Code :

```
string m_Message;
m_Message = "Message !";
MessageBox.Show(m_Message);
```



Résultat:

### 6.1.3. MESSAGE AVEC TITRE

Code :

```
string m_Title;
string m_Message;
m_Message = "Message !";
m_Title = "Chap6 MessageBox";
MessageBox.Show(m_Message, m_Title);
```



Résultat:

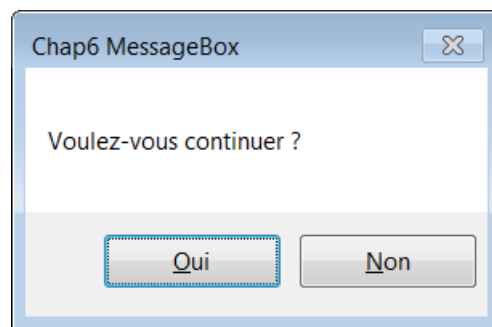


#### 6.1.4. MESSAGE AVEC TITRE ET BOUTONS OUI/NON

Code :

```
string m_Title;
string m_Message;
MessageBoxButtons m_Buttons;
DialogResult m_Reponse;
m_Buttons = MessageBoxButtons.YesNo;

m_Title = "Chap6 MessageBox";
m_Message = "Voulez-vous continuer ?";
m_Reponse = MessageBox.Show(m_Message, m_Title, m_Buttons);
if (m_Reponse == DialogResult.Yes) {
    MessageBox.Show("Oui");
}
if (m_Reponse == DialogResult.No)
{
    MessageBox.Show("Non");
}
```



Résultat:

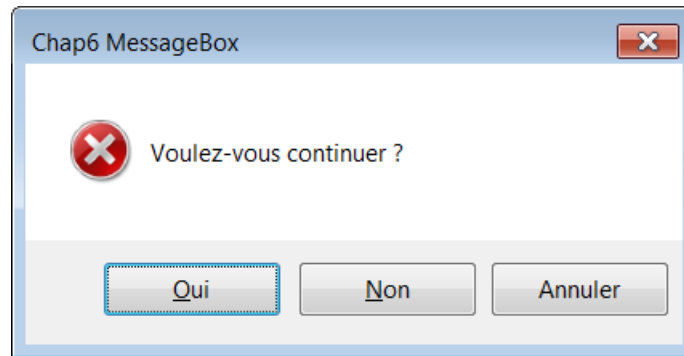
#### 6.1.5. MESSAGE AVEC TITRE ET BOUTONS OUI/NON/ANNULER

Code :

```
string m_Title;
string m_Message;
MessageBoxButtons m_Buttons;
DialogResult m_Reponse;
MessageBoxIcon m_Icône;

m_Title = "Chap6 MessageBox";
m_Buttons = MessageBoxButtons.YesNoCancel;
m_Icône = MessageBoxIcon.Stop;
m_Message = "Voulez-vous continuer ?";
m_Reponse = MessageBox.Show(m_Message, m_Title,
                             m_Buttons, m_Icône);

// Affiche le choix de l'utilisateur
MessageBox.Show(m_Reponse.ToString());
```

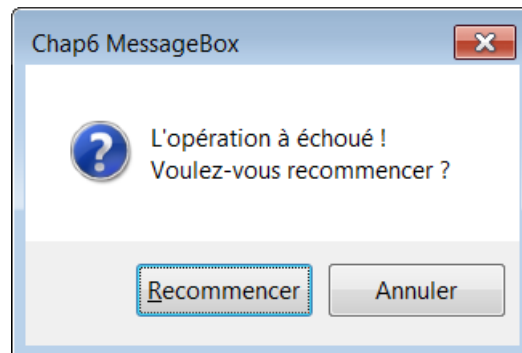


Résultat:

### 6.1.6. MESSAGE ET BOUTONS RECOMMENCER, ANNULER

Code :

```
string m_Title;  
string m_Message;  
MessageBoxButtons m_Buttons;  
DialogResult m_Reponse;  
MessageBoxIcon m_Icône;  
  
m_Title = "Chap6 MessageBox";  
m_Buttons = MessageBoxButtons.RetryCancel;  
m_Icône = MessageBoxIcon.Question;  
m_Message = "L'opération à échoué !" + "\n\r"  
            + "Voulez-vous recommencer ?";  
m_Reponse = MessageBox.Show(m_Message, m_Title,  
                             m_Buttons, m_Icône);  
  
// Affiche le choix de l'utilisateur  
MessageBox.Show(m_Reponse.ToString());
```



Résultat:

## 6.2. LA FONCTION INPUTBOX

Affiche une invite dans une boîte de dialogue, attend que l'utilisateur tape du texte ou clique sur un bouton, puis renvoie le contenu de la zone de texte sous la forme d'une valeur de type **String**. Cette fonction fait partie de l'espace de nom Microsoft.**VisualBasic**, section Interaction. Elle n'a pas de classe équivalente en .Net Framework.

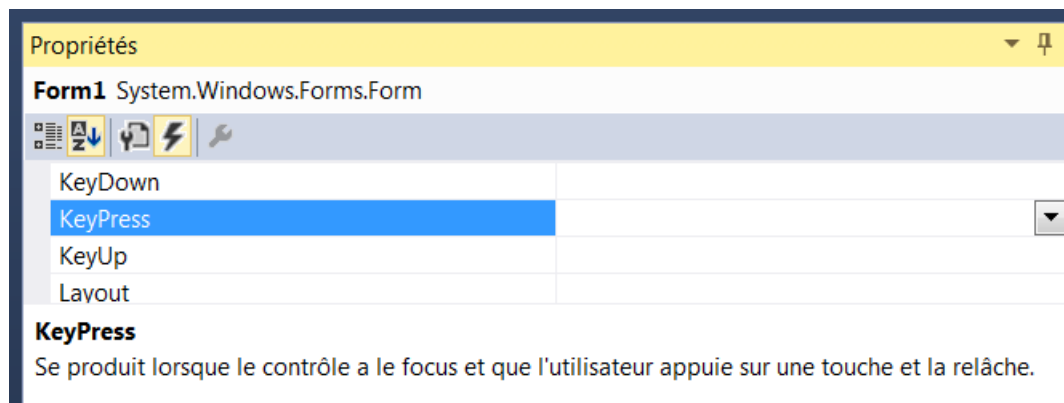
☹ par conséquent elle n'existe pas en C#

## 6.3. GESTION DU CLAVIER

Les contrôles et boîtes d'entrée ne peuvent suffire à traiter toutes les entrées clavier. Le programme doit être en mesure de répondre à des touches spécifiques au moment où l'utilisateur les frappe. Comme nous l'avons vu, Windows passe à votre programme les événements qui sont de son ressort afin qu'il puisse les traiter. Il s'agit des événements **KeyPress**, **KeyDown** et **KeyUp**.

👉 les événements du clavier sont lié à un contrôle en particulier ou au formulaire.

Événement clavier du formulaire :



Ils répondent à des combinaisons de touches du genre Alt-G ou Maj-P, ainsi qu'aux touches individuelles. Lorsqu'un événement clavier se produit, ces combinaisons sont testées. Une fois que l'application reçoit une entrée clavier, elle modifie cette entrée, ou bien l'ignore s'il ne s'agit pas de la frappe attendue. Le traitement des événements clavier permet de déclencher la fermeture d'un écran de démarrage, de valider une entrée, de jouer à des jeux, etc.

### 6.3.1. LES ÉVÉNEMENTS DU CLAVIER

Les événements de touche se produisent dans l'ordre suivant :

1. **KeyDown**            Enfoncement de la touche.
2. **KeyPress**          Utilisation de la touche.
3. **KeyUp**             Relâchement de la touche.

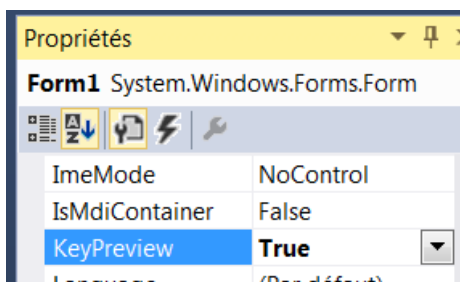
L'événement **KeyPress** n'est pas déclenché par les touches qui ne sont pas de type caractère ; cependant, ces touches déclenchent les événements **KeyDown** et **KeyUp**.

Utilisez la propriété **KeyPressEventArgs.KeyChar** pour échantillonner les séquences de touches au moment de l'exécution et utiliser ou modifier un sous-ensemble de séquences de touches courantes.

Pour gérer des événements du clavier uniquement au niveau du formulaire et ne pas autoriser d'autres contrôles à recevoir ces événements, affectez la valeur **true** à la propriété **KeyPressEventArgs.Handled** dans la méthode de gestion d'événements **KeyPress** du formulaire.

### 6.3.1.1. LA PROPRIÉTÉ KEYPREVIEW D'UNE FEUILLE

Si sa propriété **KeyPreview** a la valeur **True**, une feuille reçoit ces événements avant les contrôles qu'elle contient. Utilisez cette propriété pour créer des routines globales de gestion du clavier.



### 6.3.2. L'ÉVÉNEMENT KEYPRESS

Aide sous :

## Control.KeyPress, événement



Cet article a été traduit automatiquement. Si vous disposez d'une connexion Internet, sélectionnez Afficher cette rubrique en ligne pour consulter cette page dans un mode d'édition présentant simultanément le contenu original en anglais.

[Envoyer des commentaires](#) à Microsoft sur cette rubrique.

[Afficher cette rubrique en ligne](#) dans le navigateur par défaut.

Se produit en cas de pression sur une touche Espace ou Retour arrière alors que le contrôle a le focus.

**Espace de noms :** [System.Windows.Forms](#)

**Assembly :** System.Windows.Forms (dans System.Windows.Forms.dll)

## Syntaxe

JavaScript C# C++ F# JScript VB

```
public event KeyPressEventHandler KeyPress
```

[Copier dans le Presse-papiers](#)

L'événement **KeyPress** a lieu lorsque l'utilisateur appuie sur une touche parmi les suivantes:

- Lettres capitales et minuscules.
- Chiffres.
- Signes de ponctuation.
- Entrée, Tab et Retour arrière.

L'événement **KeyPress** reconnaît la plupart des caractères ASCII. Font toutefois exception la touche de tabulation, les touches flèches et autres caractères spéciaux dont le code ASCII est compris entre 0 et 31. **KeyPress** permet de déterminer avec exactitude la touche que l'utilisateur a frappée. Si l'utilisateur appuie sur la touche "A", **KeyPress** renvoie "A", etc.

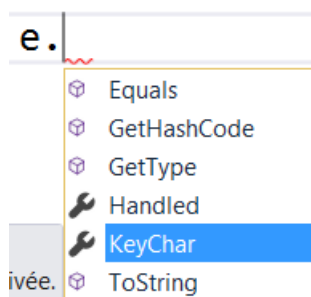
L'événement **KeyPress** a lieu lorsque la touche est enfoncée. Si l'utilisateur laisse son doigt dessus, l'événement ne se répète que si le clavier est paramétré en mode "refrappe".

Un événement, nous l'avons vu, est toujours associé à un objet tel qu'un bouton de commande ou une feuille. L'événement **KeyPress** est associé à l'objet qui, au moment de la frappe, a le *focus*. Si aucun objet n'a le *focus*, **KeyPress** s'applique à la feuille.

### 6.3.2.1. ARGUMENT E DE L'ÉVÉNEMENT KEYPRESS

L'argument **e** de l'événement KeyPress : **KeyPressEventArgs e**

Contient les champs suivants :



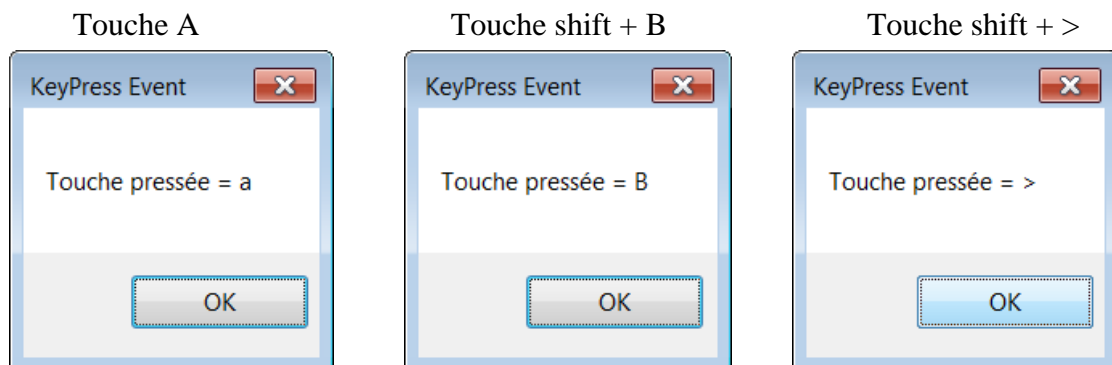
**e.KeyChar** correspond au code ASCII de la touche pressée.

**e.Handled** mis à **true** indique que l'on gère le caractère reçu sans l'attribuer au control.

#### 6.3.2.1.1. Exemple pour un Formulaire

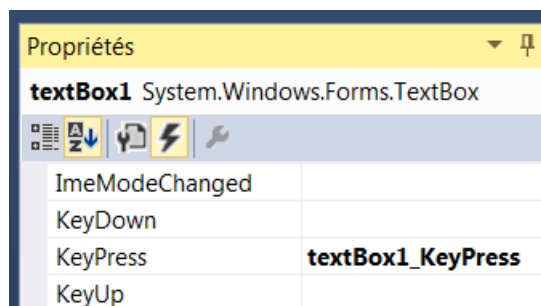
```
private void Form1_KeyPress(object sender, KeyPressEventArgs e)
{
    string Mess = "Touche pressée = " + e.KeyChar;
    MessageBox.Show(Mess, "KeyPress Event");
}
```

Si comme dans l'exemple ci-dessus on affiche la valeur de **e.KeyChar**, on obtient :



#### 6.3.2.1.2. Exemple pour un TextBox

Si on ajoute un contrôle TextBox à la feuille, il est possible d'associer l'événement KeyPress à ce contrôle :



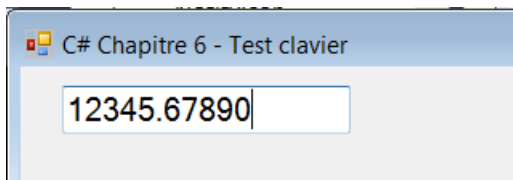
On obtient :

```
private void textBox1_KeyPress(object sender,
                               KeyPressEventArgs e)
{
}
}
```

Comme les frappes au clavier sont automatiquement affichée dans le control TextBox, le code suivant permet de filtrer les caractères affichés, en n'acceptant que les chiffres et le "." de séparation.

```
private void textBox1_KeyPress(object sender,
                               KeyPressEventArgs e)
{
    switch (e.KeyChar) {
        case '0': case '1': case '2': case '3':
        case '4': case '5': case '6': case '7':
        case '8': case '9': case '.':
            // ne fait rien
            break;
        default:
            // Élimine le caractère entré par le clavier
            e.Handled = true;
            break;
    }
}
```

☛ ne fonctionne pas si la propriété **KeyPreview** = **true** pour Form1 !




### 6.3.3. LES ÉVÉNEMENTS KEYDOWN ET KEYUP


Les événements claviers KeyDown et KeyUp se produisent lorsque l'utilisateur enfonce (KeyDown) ou relâche (KeyUp) une touche tandis qu'un objet a le focus.

#### 6.3.3.1. AIDE ÉVÉNEMENT KEYDOWN

## Control.KeyDown, événement Visual Studio

 Cet article a été traduit automatiquement. Si vous disposez d'une connexion Internet, sélectionnez Afficher cette rubrique en ligne pour consulter cette page dans un mode d'édition présentant simultanément le contenu original en anglais.

[Envoyer des commentaires](#) à Microsoft sur cette rubrique.

 Afficher cette rubrique [en ligne](#) dans le navigateur par défaut.

Se produit lorsqu'une touche est enfoncée alors que le contrôle a le focus.

**Espace de noms :** [System.Windows.Forms](#)

**Assembly :** System.Windows.Forms (dans System.Windows.Forms.dll)

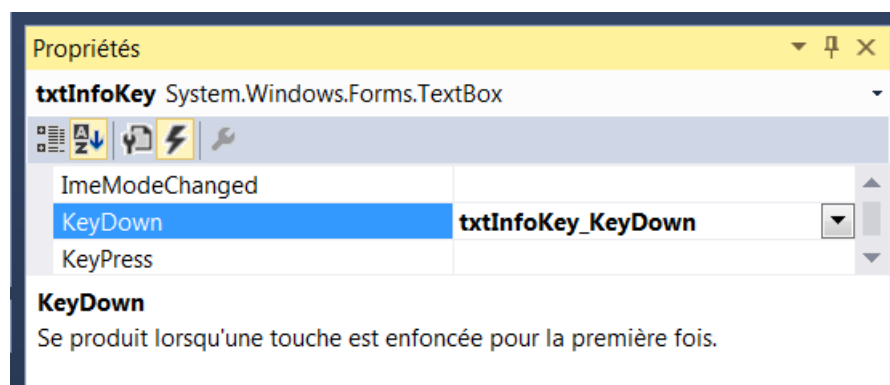
### 4 Syntaxe

JavaScript **C#** C++ F# JScript VB
[Copier dans le Presse-papiers](#)

```
public event KeyEventHandler KeyDown
```

#### 6.3.3.2. EXEMPLE ÉVÉNEMENT KEYDOWN

Voici un exemple pour un TextBox, l'événement est ajouté depuis les propriétés avec



Voici le code généré :










```
private void txtInfoKey_KeyDown(object sender, KeyEventArgs e)
{
}
}
```



L'argument **KeyEventArgs** possède des propriétés utiles à l'analyse de la touche frappée.

### 6.3.3.3. PROPRIÉTÉS DE KEYEVENTARGS

Voici les propriétés de l'argument KeyEventArgs.

	Nom	Description
	<b>Alt</b>	Obtient une valeur indiquant si la touche Alt a été activée.
	<b>Control</b>	Obtient une valeur indiquant si la touche Ctrl a été activée.
	<b>Handled</b>	Obtient ou définit une valeur indiquant si l'événement a été géré.
	<b>KeyCode</b>	Obtient le code de clavier d'un événement <b>KeyDown</b> ou <b>KeyUp</b> .
	<b>KeyData</b>	Obtient les données de touches d'un événement <b>KeyDown</b> ou <b>KeyUp</b> .
	<b>KeyValue</b>	Obtient la valeur de clavier d'un événement <b>KeyDown</b> ou <b>KeyUp</b> .
	<b>Modifiers</b>	Obtient les indicateurs de touches de modification d'un événement <b>KeyDown</b> ou <b>KeyUp</b> . Les indicateurs indiquent la combinaison de touches Ctrl, Maj et Alt activée.
	<b>Shift</b>	Obtient une valeur indiquant si la touche Maj a été activée.
	<b>SuppressKeyPress</b>	Obtient ou définit une valeur indiquant si l'événement de touche doit être transmis au contrôle sous-jacent.

#### 6.3.3.3.1. KeyCode et KeyData

Les propriétés KeyCode, KeyData et Modifiers sont du type **Keys**

#### 6.3.3.3.2. KeyValue


La propriété KeyValue est du type **int**.

#### 6.3.3.3.3. Le type énuméré Keys


Voici l'énumération Keys :

### Keys, énumération



 Cet article a été traduit automatiquement. Si vous disposez d'une connexion Internet, sélectionnez Afficher cette rubrique en ligne pour consulter cette page dans un mode d'édition présentant simultanément le contenu original en anglais.

[Envoyer des commentaires](#) à Microsoft sur cette rubrique.

 Afficher cette rubrique [en ligne](#) dans le navigateur par défaut.

Spécifie des codes et des modificateurs de touche.

Cette énumération possède un attribut **FlagsAttribute** qui permet la combinaison d'opérations de bits de ses valeurs de membres.

**Espace de noms :** [System.Windows.Forms](#)

**Assembly :** System.Windows.Forms (dans System.Windows.Forms.dll)

## 4 Syntaxe

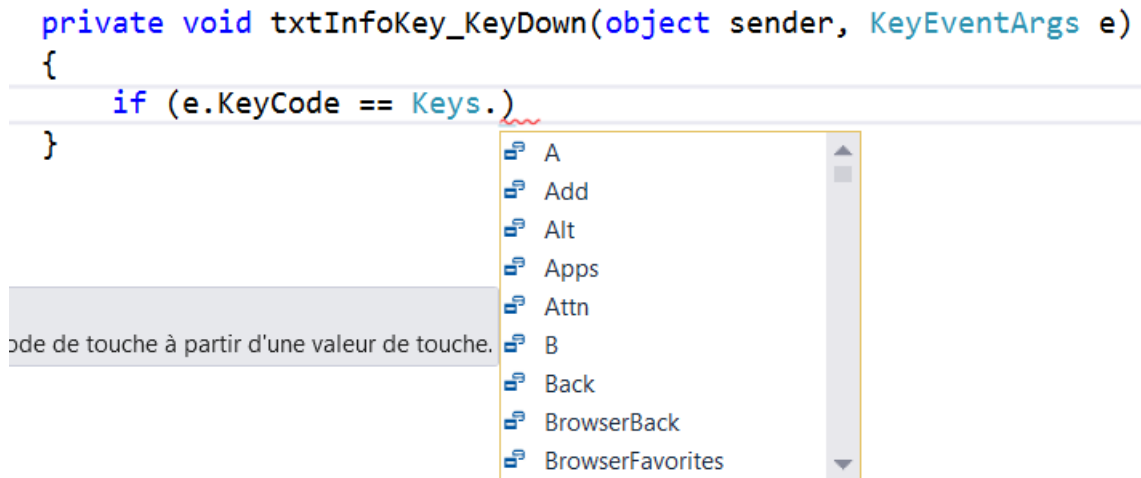
JavaScript C# C++ F# JScript VB

[Copier dans le Presse-papiers](#)

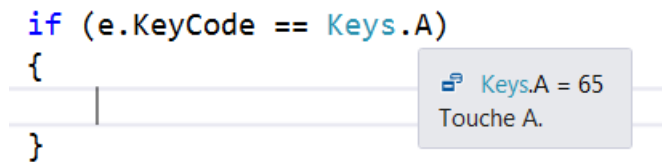
```
[FlagsAttribute]
[TypeConverterAttribute(typeof(KeysConverter))]
[ComVisibleAttribute(true)]
public enum Keys
```

Ce type énumérer définit toutes les touches du clavier.

Il est possible de tester si l'événement concerne une touche particulière :



Si on choisi une valeur on peut obtenir l'information correspondante :



Il est aussi possible de déclarer une variable du type Keys pour mémoriser ou manipuler la propriété e.KeyCode ou e.KeyData

```
Keys tmpKeyCode = e.KeyCode;
```

#### 6.3.3.4. TOUCHES EN RELATION AVEC KEYDOWN ET KEYUP

Que ce soit pour l'événement **KeyDown** ou **KeyUp**, l'objet qui a le *focus* reçoit toutes les frappes de touches. Une feuille ne peut avoir le *focus* que si elle ne contient aucun contrôle à la fois visible et activé. Les événements KeyDown et KeyUp s'appliquent à la plupart des touches, mais ils sont la plupart du temps utilisés pour :

- les touches associées à des caractères étendus, telles que les touches de fonctions;
- les touches de navigation ;
- les combinaisons de touches utilisant les modificateurs du clavier standard ;
- faire la distinction entre le pavé numérique et les touches numériques ordinaires.

Utilisez les procédures d'événement KeyDown et KeyUp si vous devez répondre à la fois à la pression et à au relâchement d'une touche.

Les événements KeyDown et KeyUp ne sont pas invoqués pour :

- la touche ENTRÉE si la feuille contient un contrôle **CommandButton** dont la propriété **Default** a la valeur **True**.
- la touche ÉCHAP (Esc) si la feuille comprend un contrôle **CommandButton** dont la propriété **Cancel** a la valeur **True**.
- la touche TAB.

### 6.3.3.5. EXEMPLE D'UTILISATION DE KEYDOWN ET KEYUP

Cet exemple affiche les différentes valeurs de l'événement `KeyEventArgs` lors d'une frappe combinant une touche et les combinaisons des touches Alt, Shift et Ctrl

La situation de Alt, Shift et Ctrl est indiquée par 3 `checkBox`. En plus on affiche dans des labels la conversion en string de `KeyCode` et `KeyData`.

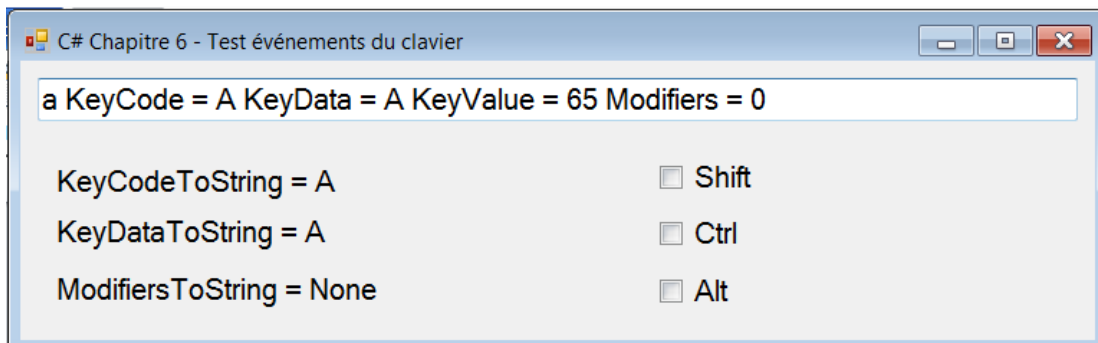
```
private void txtInfoKey_KeyDown(object sender, KeyEventArgs e)
{
    label1.Text = "KeyCodeToString = " + e.KeyCode.ToString();
    label2.Text = "KeyDataToString = " + e.KeyData.ToString();
    label3.Text = "ModifiersToString = " +
                  e.Modifiers.ToString();

    int modifiers = (int)e.Modifiers;

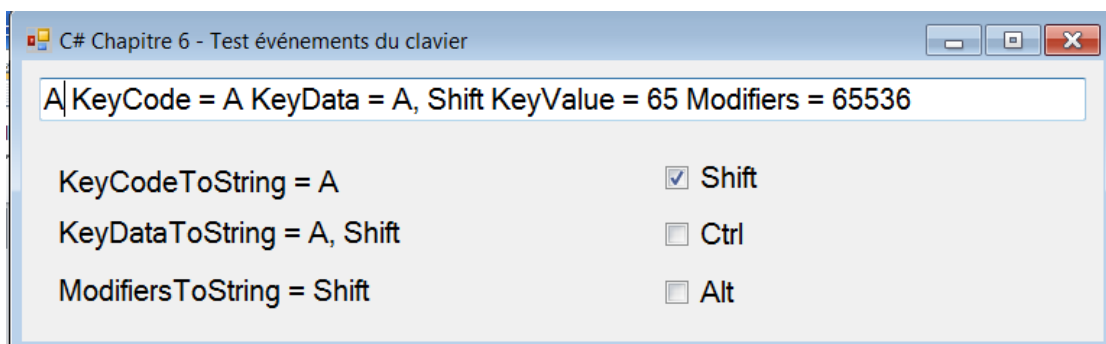
    txtInfoKey.Text = " KeyCode = " + e.KeyCode +
                     " KeyData = " + e.KeyData +
                     " KeyValue = " + e.KeyValue +
                     " Modifiers = " + modifiers;

    chkShift.Checked = e.Shift;
    chkCtrl.Checked = e.Control;
    chkAlt.Checked = e.Alt;
}
```

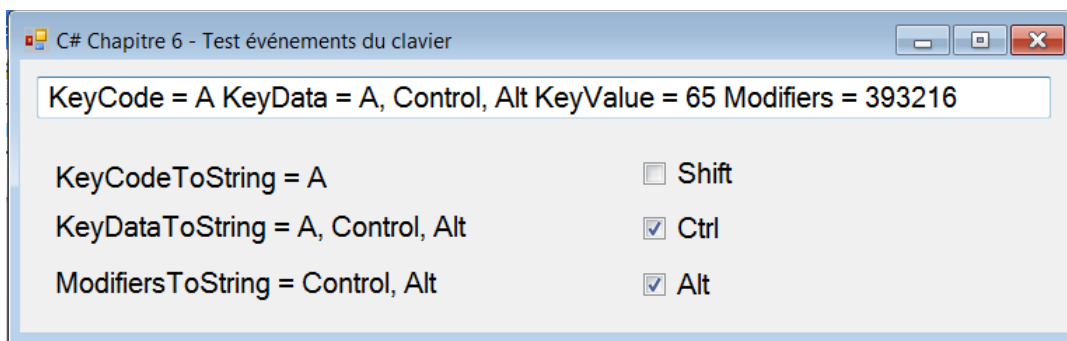
Résultat avec touche A seule :



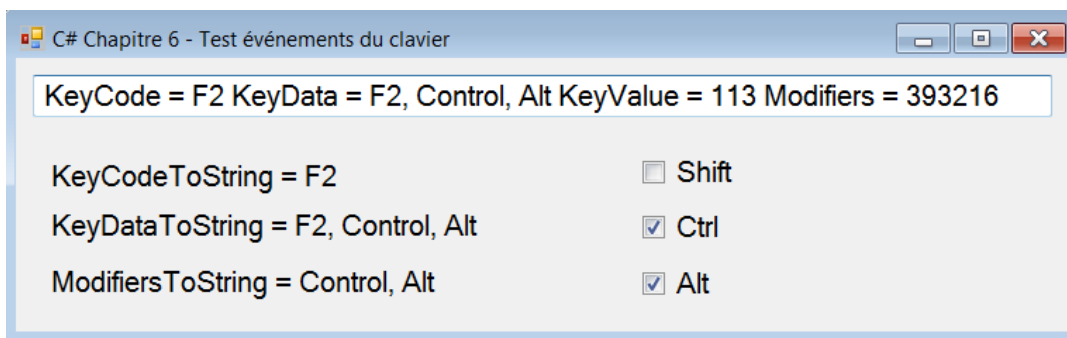
Résultat avec touche Shift + A:



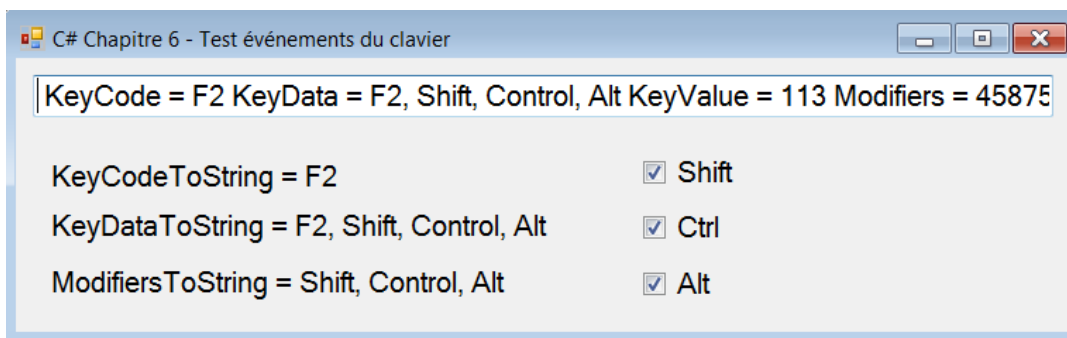
Résultat avec touche Alt + Ctrl + A :



Résultat avec touche Alt + Ctrl + F2 :



Résultat avec touche Alt + Ctrl + Shift + F2 :



La valeur du Modifiers exprimée en entier se comprend mieux si on l'exprime en hexadécimal:

Avec shift :	065536	=> 0x10000
Avec Ctrl	131072	=> 0x20000
Avec Alt	262144	=> 0x40000
Avec Alt+ Ctrl	393216	=> 0x60000 = combinaison
Avec Alt+ Ctrl+Shift	458752	=> 0x70000 = combinaison

La valeur de KeyData est réalisée de la manière suivante :  
KeyData = Modifiers + KeyValue.

On observe encore l'information fournie par le **e.KeyData.ToString()**, qui indique aussi la situation de Shift, Control et Alt. De même **e.Modifiers.ToString()** nous fournis la situation de Shift, Control et Alt.

## 6.4. GESTION DE LA SOURIS

Les actions de la souris tels que les déplacements, clicks, double-clic, glisser-déposer (*drag and drop*), et autres sont les fondements des applications Windows. Si l'utilisateur se sert de la souris lors de l'exécution, Windows passe les événements souris à votre programme. Les programmes que vous écrivez doivent interroger les événements souris et y répondre au moment et de la façon nécessaires. Si l'utilisateur se contente de cliquer sur un bouton d'option ou sur une case à cocher, naturellement, le programme n'a pas à répondre, car le contrôle déclenche un événement Click. Mais Visual Basic suit également les mouvements de la souris lorsque l'utilisateur effectue des glisser-déposer ou des copier-coller (*copy and paste*).

### 6.4.1. RÉPONSES AUX ÉVÉNEMENTS DE LA SOURIS

Vous pouvez utiliser les événements **MouseDown**, **MouseUp** et **MouseMove** pour activer vos applications afin qu'elles réagissent à la fois à la position et à l'état de la souris. Les événements **MouseEnter** et **MouseLeave** indiquent l'arrivée et le départ d'un contrôle du pointeur de la souris mais ne donnent pas d'information de position. L'événement **MouseWheel** est lié à la roulette de la souris.

Les événements suivants sont reconnus par la plupart des contrôles.

Événement	Description
MouseDown	Se produit lorsque l'utilisateur appuie sur un bouton quelconque de la souris.
MouseUp	Se produit lorsque l'utilisateur relâche un bouton quelconque de la souris.
MouseMove	Se produit chaque fois que le pointeur de la souris est déplacé vers un nouvel emplacement de l'écran.
MouseEnter	Se produit lorsque le pointeur de la souris se place dans le contrôle.
MouseLeave	Se produit lorsque le pointeur de la souris s'écarte du contrôle.
MouseWheel	Se produit lorsque la roulette de la souris bouge alors que le contrôle a le focus.

#### 6.4.1.1. APPARTENANCE DES ÉVÉNEMENTS DE LA SOURIS







Une feuille peut reconnaître un événement souris lorsque le pointeur se trouve à un endroit de la feuille où il n'y a pas de contrôle.

Un contrôle reconnaît l'événement lorsque le pointeur se trouve dessus.

Lorsque l'utilisateur maintient un bouton de la souris enfoncé, le même objet continue à reconnaître les événements souris jusqu'à ce que le bouton soit relâché, même si le pointeur est déplacé hors des limites de l'objet.

#### 6.4.1.2. ARGUMENTS DES ÉVÉNEMENTS DE LA SOURIS

Les quatre événements souris (MouseMove,MouseDown, MouseUp et MouseWheel) possèdent un argument e du type **System.Windows.Forms.MouseEventArgs** dont les propriétés sont les suivantes : (Pour les événements MouseEnter et MouseLeave, l'argument est du type **System.EventArgs**) .

	Nom	Description
	Button	Obtient le bouton de la souris sur lequel l'utilisateur a appuyé.
	Clicks	Obtient le nombre de fois où l'utilisateur a cliqué sur le bouton de la souris et l'a relâché.
	Delta	Obtient un décompte signé du nombre de détentes de rotation de la roulette de la souris, multiplié par la constante WHEEL_DELTA. Une détente représente un cran de la roulette de la souris.
	Location	Obtient l'emplacement de la souris pendant la génération d'événement de souris.
	X	Obtient la coordonnée x de la souris pendant la génération de l'événement de souris.
	Y	Obtient la coordonnée y de la souris pendant la génération de l'événement de souris.

##### 6.4.1.2.1. La propriété Button

La propriété *Button* est un type énuméré **MouseButtons** qui est décrit dans le tableau suivant :

Nom de membre	Description
<b>Left</b>	Le bouton gauche de la souris a été enfoncé.
<b>Middle</b>	Le bouton central de la souris a été enfoncé.
<b>None</b>	Aucun bouton de la souris n'a été enfoncé.
<b>Right</b>	Le bouton droit de la souris a été enfoncé.
<b>XButton1</b>	Le premier XButton a été enfoncé. Avec Windows 2000, Microsoft ajoute la prise en charge de la souris Microsoft IntelliMouse Explorer dotée de cinq boutons. Les deux nouveaux boutons de la souris (XBUTTON1 et XBUTTON2) sont utilisés pour la navigation avant/arrière.
<b>XButton2</b>	Le second XButton a été enfoncé. Avec Windows 2000, Microsoft ajoute la prise en charge de la souris Microsoft IntelliMouse Explorer dotée de cinq boutons. Les deux nouveaux boutons de la souris (XBUTTON1 et XBUTTON2) sont utilisés pour la navigation avant/arrière.

Le bouton Middle est obtenu en appuyant sur la roulette. Les XButton1 et Xbutton2 ont un sens pour les souris qui en possèdent, comme par exemple la souris Microsoft IntelliMouse.

##### 6.4.1.2.1. La propriété Delta

La valeur de la propriété Delta quantifie le déplacement de la roulette de la souris. C'est en particulier avec l'événement MouseWheel que cette propriété prend sens.

#### 6.4.1.2.2. La propriété Location

La propriété **Location** est du type **Point**, elle fournit la paire de coordonnée X et Y en une seule fois.

Avec

**Point** DownPos;

Il est possible lors de l'événement **MouseDown** d'écrire pour mémoriser et afficher :

```
private void Panel1_MouseDown(object sender, MouseEventArgs e)
{
    DownPos = e.Location;
    label1.Text = "DownPos = " + DownPos.ToString();
}
```

Résultat au niveau du label :

DownPos = {X=147,Y=55}

Au lieu de :

```
private void Panel1_MouseDown(object sender, MouseEventArgs e)
{
    DownPos.X = e.X;
    DownPos.Y = e.Y;
    label1.Text = "DownPos : X= " + e.X + " Y= " + e.Y;
}
```

Résultat au niveau du label :

DownPos : X= 170 Y= 73

#### 6.4.1.3. ÉVÉNEMENTS SOURIS, EXEMPLE 1

Soit un contrôle de type **Panel**, lors du déplacement de la souris sur la zone du panel la couleur de fond devient bleu, lorsque le curseur revient sur la feuille la couleur de fond du Panel devient beige. A la construction on établit la couleur est rouge.

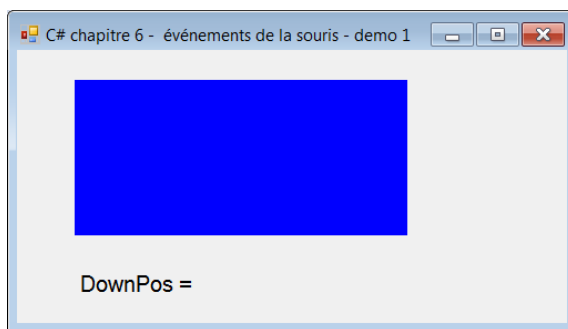
```
public partial class Form1 : Form
{
    Point DownPos;
    public Form1()
    {
        InitializeComponent();
        panel1.BackColor = Color.Red;
    }

    private void Form1_MouseMove(object sender,
                                MouseEventArgs e)
    {
        panel1.BackColor = Color.Yellow;
    }
}
```

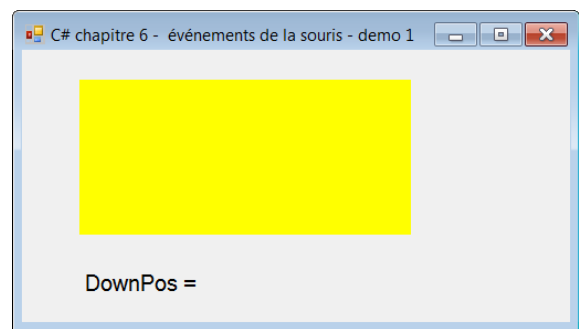
```
private void Panel1_MouseMove(object sender,
                               MouseEventArgs e)
{
    panel1.BackColor = Color.Blue;
}

private void Panel1_MouseDown(object sender,
                               MouseEventArgs e)
{
    DownPos = e.Location;
    label1.Text = "DownPos = " + DownPos.ToString();
}
}
```

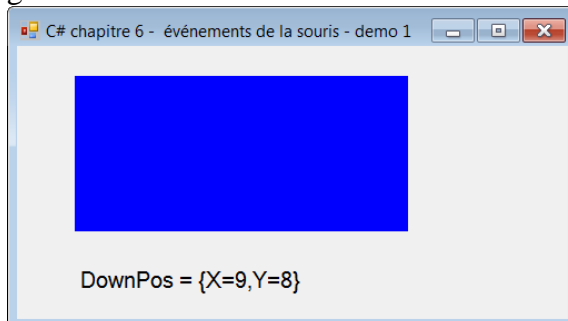
Lorsque la souris passe sur le Panel



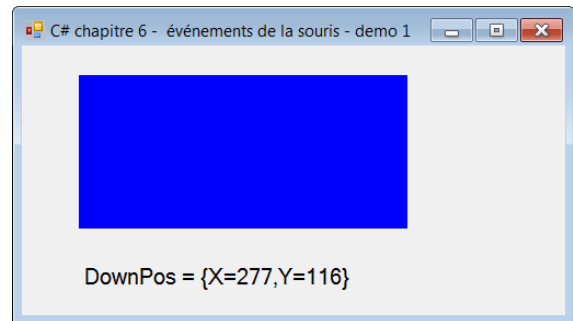
Lorsque la souris n'est plus sur le Panel



Lors d'un click droit près du coin en haut à gauche du Panel



Lors d'un click droit près du coin en bas à droite du Panel

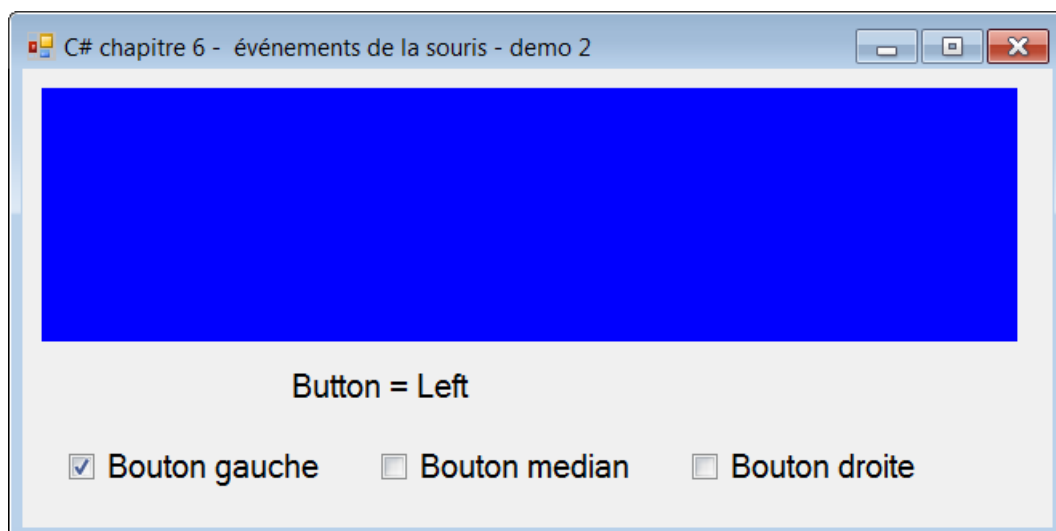




#### 6.4.1.1. ÉVÉNEMENTS SOURIS, EXEMPLE 2

Cet exemple illustre le test des boutons. Lors du déplacement de la souris sur le Panel, la situation des boutons est indiquée par des CheckBox ainsi que par un label en relation avec la méthode **.ToString**. Remarque : il faut se déplacer avec la souris dans le panel pour avoir l'information sur la situation des boutons.

```
private void Panel1_MouseMove(object sender,
                               MouseEventArgs e)
{
    panel1.BackColor = Color.Blue;
    lblButtons.Text = "Button = " + e.Button.ToString();
    switch (e.Button)
    {
        case MouseButton.None:
            chkLeft.Checked = false;
            chkRight.Checked = false;
            chkMiddle.Checked = false;
            break;
        case MouseButton.Left:
            chkLeft.Checked = true;
            break;
        case MouseButton.Right:
            chkRight.Checked = true;
            break;
        case MouseButton.Middle:
            chkMiddle.Checked = true;
            break;
    } // end swicth
}
```



#### 6.4.1.2. EVÉNEMENTS SOURIS, EXEMPLE 3

Cet exemple montre l'exploitation de la position X et Y lors de l'événement **MouseDown** et **MouseUp**. En plus une ligne est dessinée entre les deux positions. L'utilisation de la méthode **DrawLine** implique l'usage d'un objet **Graphics** et d'un objet **Pen** qui seront étudié plus loin dans le chapitre.

```
public partial class Form1 : Form
{
    Point m_DownPos;
    Point m_UpPos;
    Graphics m_Graphic;

    public Form1()
    {
        InitializeComponent();
        panel1.BackColor = Color.Blue;
        m_Graphic = panel1.CreateGraphics();
    }

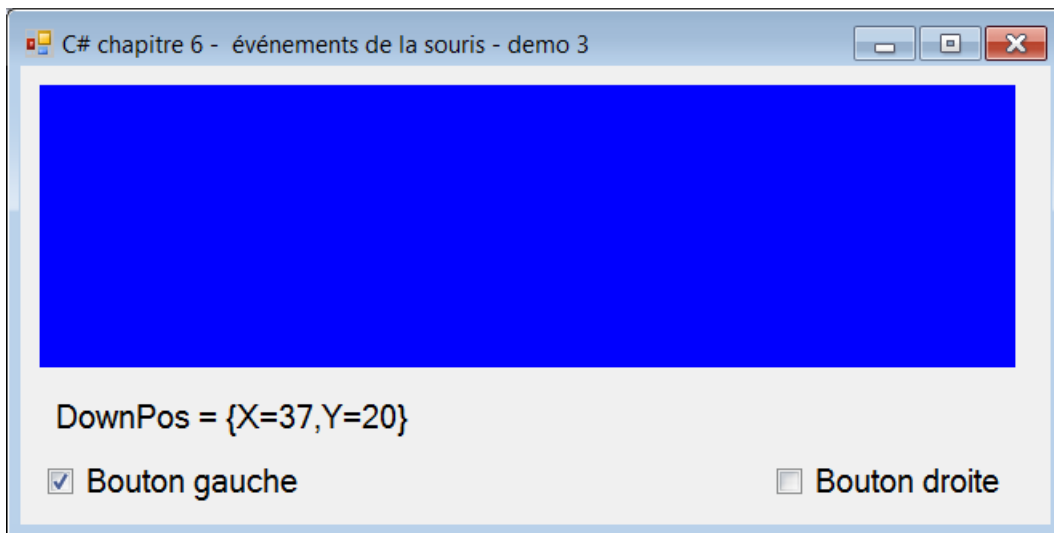
    private void ShowButtons (MouseButtons SituationBouton)
    {
        switch (SituationBouton)
        {
            case MouseButtons.None:
                chkLeft.Checked = false;
                chkRight.Checked = false;
                break;
            case MouseButtons.Left:
                chkLeft.Checked = true;
                break;
            case MouseButtons.Right:
                chkRight.Checked = true;
                break;
        } // end swicth
    }

    private void Panel1_MouseMove(object sender,
                                   MouseEventArgs e)
    {
        ShowButtons(e.Button);
    }

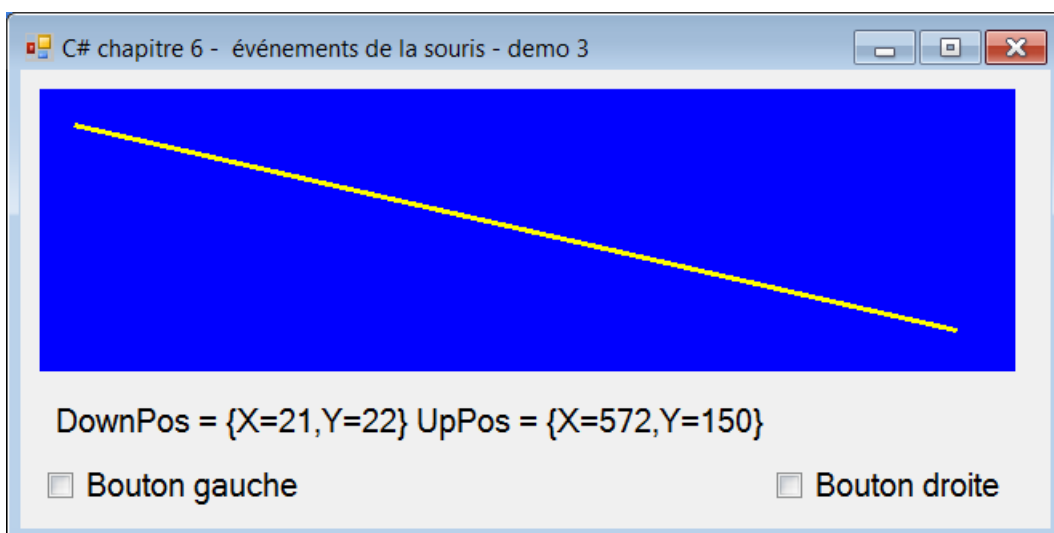
    private void Panel1_MouseDown(object sender,
                                    MouseEventArgs e)
    {
        // Mémoire les coordonnées et affiche
        m_DownPos = e.Location;
        lblPos.Text = " DownPos = " + m_DownPos.ToString();
        ShowButtons(e.Button);
    }
}
```

```
private void Panel1_MouseUp(object sender,
                             MouseEventArgs e)
{
    Pen ThePen;
    m_UpPos = e.Location;
    // Affiche les 2 coordonnées
    lblPos.Text = " DownPos = " + m_DownPos.ToString() +
                  " UpPos = " + m_UpPos.ToString();
    ShowButtons(e.Button);
    // Relie les 2 points par une ligne jaune width de 3
    ThePen = new Pen(Color.Yellow, 3);
    m_Graphic.DrawLine(ThePen, m_DownPos, m_UpPos);
}
}
```

Résultat lors du **MouseDown** ( bouton gauche enfoncé) :



Résultat lors du **MouseUp** après un déplacement avec le bouton gauche enfoncé :



### 6.4.2. INFORMATIONS SUR LA SOURIS

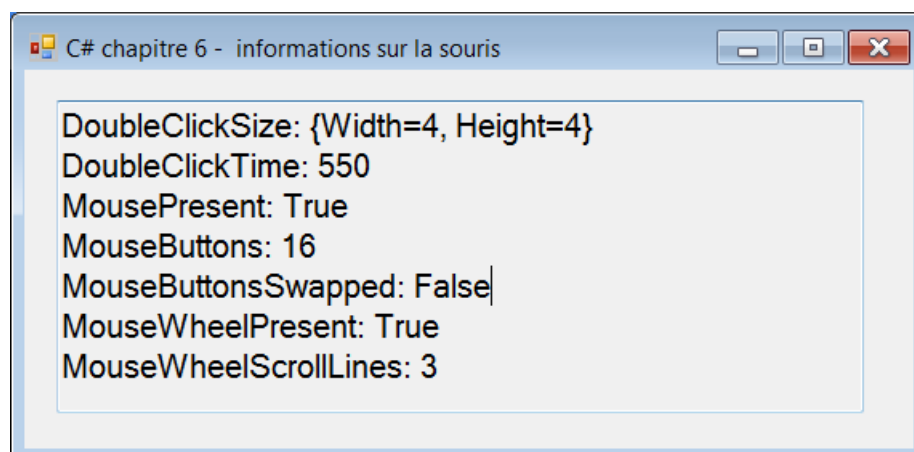
La classe **SystemInformation** nous fournit les indications systèmes à propos de la souris et de bien d'autres éléments.

Voici un programme inspiré d'un exemple Microsoft, l'action est réalisée dans le constructeur.

```
public Form1()
{
    string [ ] MultiLignes = new string[8];
    InitializeComponent();
    // Affichage des Info sur la souris.
    MultiLignes[0] = "DoubleClickSize: " +
        SystemInformation.DoubleClickSize.ToString();
    MultiLignes[1] = "DoubleClickTime: " +
        SystemInformation.DoubleClickTime.ToString();
    MultiLignes[2] = "MousePresent: " +
        SystemInformation.MousePresent.ToString();
    MultiLignes[3] = "MouseButtons: " +
        SystemInformation.MouseButtons.ToString();
    MultiLignes[4] = "MouseButtonsSwapped: " +
        SystemInformation.MouseButtonsSwapped.ToString();
    MultiLignes[5] = "MouseWheelPresent: " +
        SystemInformation.MouseWheelPresent.ToString();
    MultiLignes[6] = "MouseWheelScrollLines: " +
        SystemInformation.MouseWheelScrollLines.ToString();
    MultiLignes[7] = "NativeMouseWheelSupport: " +
        SystemInformation.NativeMouseWheelSupport.ToString();

    txtInfo.Lines = MultiLignes;
}
```

Résultat :



Il est surprenant d'obtenir l'information `MouseButtons` 16, s'agit-il d'un code décrivant la configuration des boutons.

Remarque : la classe `SystemInformation` contient beaucoup d'autres informations systèmes.

#### 6.4.2.1. EVÉNEMENTS SOURIS, EXEMPLE 4

Cet exemple montre l'exploitation de l'événement **MouseWheel** pour modifier la position d'une ligne dessinée. L'utilisation de la méthode **DrawLine** implique l'usage d'un objet **Graphics** et d'un objet **Pen** qui seront étudié plus loin dans le chapitre.

☹ mystérieusement l'événement **MouseWheel** n'apparaît pas dans la liste des événement du **Panel**. Cela nous permettra de créer l'événement au niveau du code !

L'événement **MouseEnter** est utilisé pour donner le **focus** au **Panel**.

👉 Il est nécessaire de donner le focus au **Panel** pour que l'événement **MouseWheel** se produise !

```
public partial class Form1 : Form
{
    Point m_DownPos;
    Point m_UpPos;
    Graphics m_Graphic;
    int m_ValDelta;

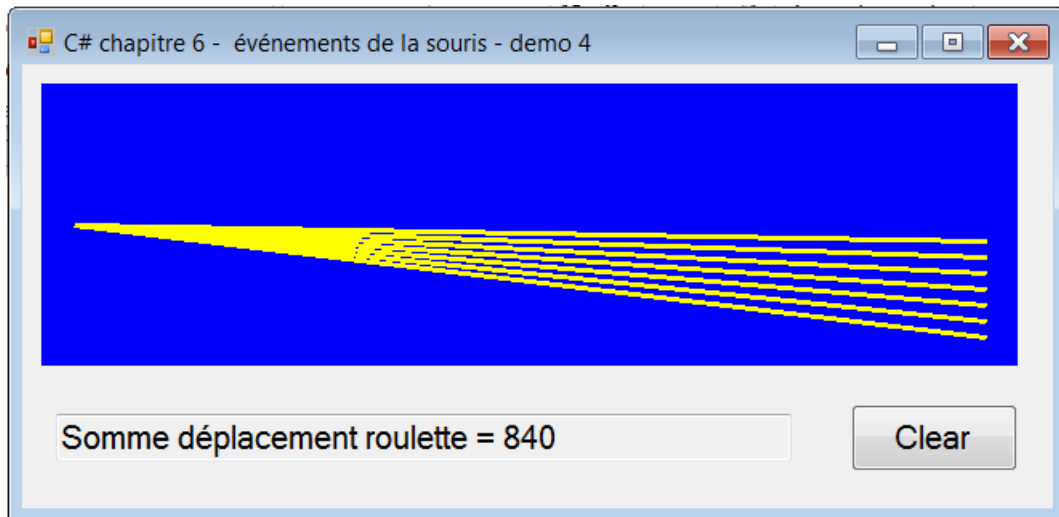
    public Form1()
    {
        InitializeComponent();
        // Ajout de l'événement
        this.panel1.MouseWheel += new
System.Windows.Forms.MouseEventHandler(this.panel1_MouseWheel);
        panel1.BackColor = Color.White;
        m_Graphic = panel1.CreateGraphics();
        m_ValDelta = 0;
        m_DownPos = new Point(20, panel1.Height / 2);
        m_UpPos = new Point(panel1.Width - 20,
                                panel1.Height / 2);
    }

    // Transformation d'un autre événement
    private void panel1_MouseWheel(object sender,
                                    MouseEventArgs e)
    {
        Pen ThePen;
        m_ValDelta += e.Delta;
        txtRes.Text = "Somme déplacement roulette = " +
                                m_ValDelta;
        // Relie les 2 points par une ligne jaune épaisse de 3
        // en variant en Y l'extrémité de la ligne
        ThePen = new Pen(Color.Yellow);
        ThePen.Width = 3;
        m_UpPos.Y = m_UpPos.Y + (e.Delta / 12);
        m_Graphic.DrawLine(ThePen, m_DownPos, m_UpPos);
    }
}
```

```
private void Panel1_MouseEnter(object sender, EventArgs e)
{
    panel1.Focus();
    panel1.BackColor = Color.Blue;
}

private void btnClear_Click(object sender, EventArgs e)
{
    m_Graphic.Clear(panel1.BackColor);
    m_ValDelta = 0;
    m_UpPos = new Point(panel1.Width - 20,
                        panel1.Height / 2);
    txtRes.Text = "Somme déplacement roulette = " +
                m_ValDelta;
}
}
```

Exemple de résultat:



## 6.5. LA CLASSE GRAPHICS

Pour réaliser des dessins de lignes et courbes il est nécessaire d'utiliser la classe **Graphics**.

Cette Classe **Graphics** doit être associée à un contrôle permettant le graphisme, comme par exemple un Panel, il est aussi possible de l'associer à une feuille (Form) ou à un GroupBox.

Remarque : le contrôle PictureBox supporte aussi la méthode CreateGraphics.

Pour réaliser un dessin sur un contrôle il est nécessaire de mettre en œuvre les éléments suivants :

- Création et liens d'un objet de type **Graphics** à un contrôle en utilisant la méthode **CreateGraphics**
- Création et configuration d'un objet de type Pen pour déterminer l'apparence du trait que l'on utilise avec les méthodes de dessin.
- Gestion des coordonnées du dessin à l'aide d'objet de type **Point** ou **PointF**

### 6.5.1. LA MÉTHODE CREATEGRAPHICS DES CONTRÔLES

Cette méthode crée l'objet Graphics pour le contrôle. La méthode CreateGraphics n'existe que pour les contrôles qui supportent du graphisme.

Exemple pour un Panel :

```
Dim MyGraph As Graphics 'variable globale
```

Puis par exemple dans l'événement Form\_Load:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal  
e As System.EventArgs) Handles MyBase.Load  
    MyGraph = Panel1.CreateGraphics  
End Sub
```

Exemple pour une Feuille :

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal  
e As System.EventArgs) Handles MyBase.Load  
    MyGraph = Me.CreateGraphics  
End Sub
```





### 6.5.2. L'OBJET PEN

Un objet **Pen** dessine une ligne de largeur et de style spécifié. Utilise la propriété **DashStyle** pour dessiner plusieurs variétés de lignes discontinues. La ligne dessinée par un objet **Pen** peut être remplie avec une variété de styles de remplissage, y compris des couleurs unies et des textures. Le style de remplissage dépend de la brosse (Brush) ou de la texture utilisée comme objet de remplissage.

Il est nécessaire de créer un objet de la classe **Pen** en utilisant **New**.

### 6.5.2.1. LISTE DE SURCHARGE DE PEN







Il est possible de construire un objet Pen en précisant différent paramètres

	Nom	Description
	Pen(Brush)	Initialise une nouvelle instance de la classe Pen avec le Brush spécifié.
	Pen(Color)	Initialise une nouvelle instance de la classe Pen avec la couleur spécifiée.
	Pen(Brush, Single)	Initialise une nouvelle instance de la classe Pen avec les Brush et Width spécifiés.
	Pen(Color, Single)	Initialise une nouvelle instance de la classe Pen avec les propriétés Color et Width spécifiées.







Remarque : l'objet **Brush** (Pinceau) offre une autre possibilité pour établir la couleur du trait.

### 6.5.2.2. PROPRIÉTÉS DE PEN

Voici la liste des propriétés de la classe System.Drawin.Pen

	Nom	Description
	Alignment	Obtient ou définit l'alignement pour ce Pen.
	Brush	Obtient ou définit l'objet Brush qui détermine les attributs de ce Pen.
	Color	Obtient ou définit la couleur de Pen.
	CompoundArray	Obtient ou définit un tableau de valeurs qui spécifie un stylet composé. Un stylet composé dessine une ligne composée constituée de lignes parallèles et d'espaces.
	CustomEndCap	Obtient ou définit une extrémité personnalisée à utiliser à la fin des lignes dessinées avec ce Pen.
	CustomStartCap	Obtient ou définit une extrémité personnalisée à utiliser au début des lignes dessinées avec ce Pen.
	DashCap	Obtient ou définit le style d'extrémité utilisé à la fin des tirets qui constituent les lignes en pointillés dessinées avec ce Pen.
	DashOffset	Obtient ou définit la distance entre le début d'une ligne et le début d'un tiret.
	DashPattern	Obtient ou définit un tableau d'espaces et de tirets personnalisés.
	DashStyle	Obtient ou définit le style utilisé pour les lignes en pointillés dessinées à l'aide de ce Pen.
	EndCap	Obtient ou définit le style d'extrémité utilisé à la fin des lignes dessinées avec ce Pen.



	<b>LineJoin</b>	Obtient ou définit le style de jointure pour la fin de deux lignes consécutives dessinées avec ce <b>Pen</b> .
	<b>MiterLimit</b>	Obtient ou définit la limite d'épaisseur de la jointure sur un coin à onglet.
	<b>PenType</b>	Obtient le style des lignes dessinées à l'aide de ce <b>Pen</b> .
	<b>StartCap</b>	Obtient ou définit le style d'extrémité utilisé au début des lignes dessinées avec ce <b>Pen</b> .
	<b>Transform</b>	Obtient ou définit une copie de la transformation géométrique pour ce <b>Pen</b> .
	<b>Width</b>	Obtient ou définit la largeur de ce <b>Pen</b> , en unités de l'objet <b>Graphics</b> utilisé pour dessiner.

### 6.5.2.3. EXEMPLE UTILISATION DES PROPRIÉTÉS DE PEN

L'objet Pen possède un certain nombre de propriétés. Les plus importantes pour le dessin sont : **Color**, **Width** et **DashStyle**.

Voici un exemple comportant la déclaration et la modification d'un objet Pen, dans le but d'obtenir 3 styles de traits :

- un trait continu noir d'une épaisseur de 3.
- un trait fait de point de couleur verte et d'une épaisseur de 5.
- un trait composé de 2 points et d'un trait, de couleur rouge et d'une épaisseur de 8.

```
Dim MyGraph As Graphics
```

```
Dim ThePen As Pen
```

```
Dim PosDown, PosUp As Point
```

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
```

```
    MyGraph = Panel1.CreateGraphics
```

```
    ThePen = New Pen(Color.Black)
```

```
    ThePen.Width = 3
```

```
    PosDown.X = 20
```

```
    PosDown.Y = 20
```

```
    PosUp.X = Panel1.Width - 20
```

```
    PosUp.Y = 20
```

```
    OptStyle1.Checked = True
```

```
End Sub
```

```
Private Sub cmdDraw_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdDraw.Click
```

```
    If OptStyle1.Checked Then
```

```
        ' trait noir épaisseur 3
```

```
        ThePen = New Pen(Color.Black, 3)
```

```
    End If
```

```

    If OptStyle2.Checked Then
        ThePen = New Pen(Color.Green)
        ThePen.Width = 5
        ThePen.DashStyle = Drawing.Drawing2D.DashStyle.Dot
    End If
    If OptStyle3.Checked Then
        ThePen = New Pen(Color.Red, 8)
        ThePen.DashStyle = _
            Drawing.Drawing2D.DashStyle.DashDotDot
    End If
    MyGraph.DrawLine(ThePen, PosDown, PosUp)
    'Déplace la ligne
    PosDown.Y += 20
    PosUp.Y += 20
End Sub
End Class

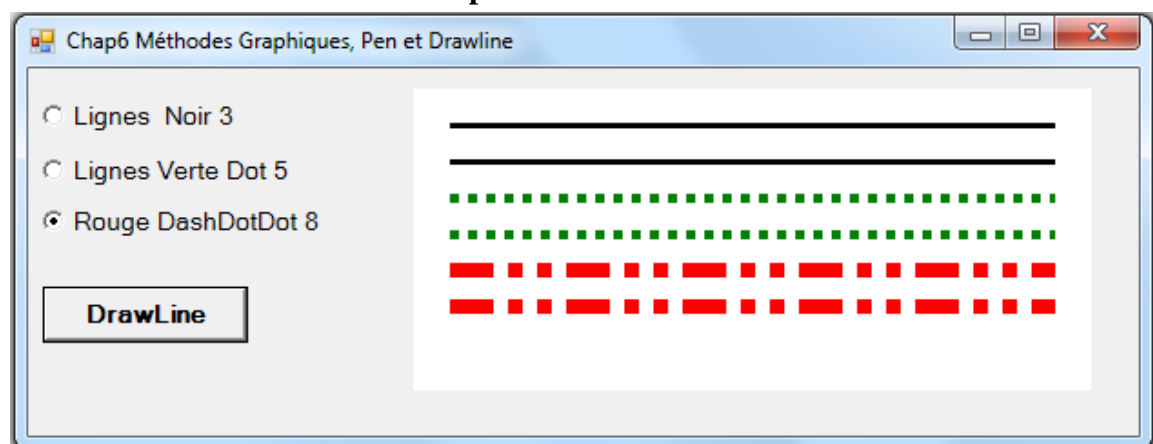
```

Remarque : certaines propriétés sont établies lors de la création avec **New**, d'autres peuvent être modifiées par la suite.



Il est indispensable d'effectuer une fois un **New** pour que l'objet existe. Ensuite il est possible de modifier les propriétés.

#### 6.5.2.3.1. Résultat de l'exemple Pen



#### 6.5.2.4. PEN, PROPRIÉTÉ COLOR

La propriété **Color** permet de définir la couleur. La classe **Color** possède une liste de membres représentant les différentes couleurs. L'attribution de la propriété **Color** est réalisée lors du **New**. Ensuite il est possible de changer la couleur.

```

ThePen = New Pen(Color.Black)
ThePen.Color = Color.Green

```

#### 6.5.2.5. PEN, PROPRIÉTÉ WIDTH

La propriété **Width** permet avec un nombre de définir l'épaisseur du trait de 1 à n.

```

ThePen = New Pen(Color.Yellow)
ThePen.Width = 3

```

OU en une seule fois :

```

ThePen = New Pen(Color.Yellow, 3)

```

#### 6.5.2.6. PEN, PROPRIÉTÉ DASHSTYLE

La propriété **DashStyle** permet de définir le style du trait : plein, constitué de points ou de tirets.

Il s'agit d'un type énuméré pouvant prendre les valeurs suivantes :

Nom de membre	Description
<b>Solid</b>	Spécifie une ligne pleine.
<b>Dash</b>	Spécifie une ligne constituée de tirets.
<b>Dot</b>	Spécifie une ligne constituée de points.
<b>DashDot</b>	Spécifie une ligne constituée d'un motif tiret-point répété.
<b>DashDotDot</b>	Spécifie une ligne constituée d'un motif tiret-point-point répété.
<b>Custom</b>	Spécifie un style de ligne personnalisé défini par l'utilisateur.

Remarque : pour définir une propriété **DashStyle** personnalisée, définissez la propriété **DashPattern** de Pen.

#### 6.5.2.7. PEN, EXEMPLE DASHSTYLE, CUSTOM

La propriété **DashPattern** d'une Pen, accepte un tableau de Single, donnant successivement la longueur du trait et la longueur de l'espace.

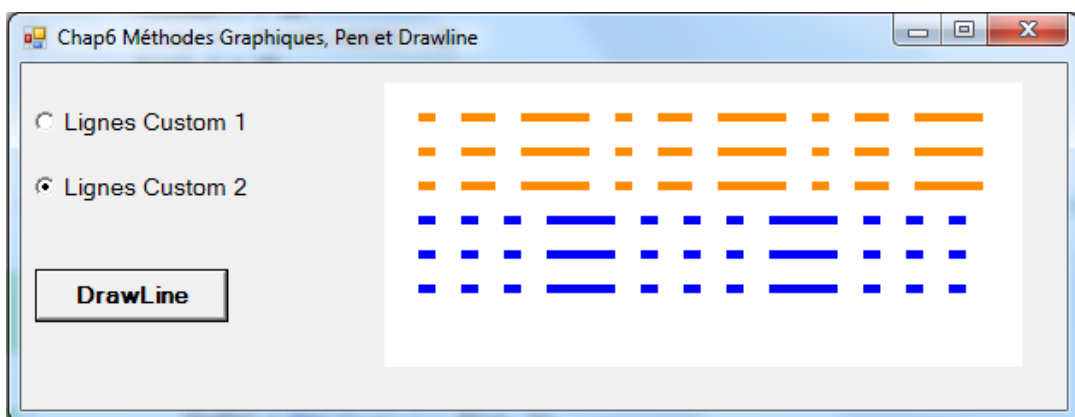
Voici en reprenant le même principe que l'exemple précédant la configuration de 2 styles personnalisés.

```

If OptStyle1.Checked Then
    ThePen = New Pen(Color.DarkOrange, 5)
    ThePen.DashPattern = New Single() {2, 3, 4, 3, 8, 3}
    ThePen.DashStyle = Drawing.Drawing2D.DashStyle.Custom
End If

If OptStyle2.Checked Then
    ThePen = New Pen(Color.Blue, 5)
    ThePen.DashPattern = _
        New Single() {2, 3, 2, 3, 2, 3, 8, 3}
    ThePen.DashStyle = Drawing.Drawing2D.DashStyle.Custom
End If




```



### 6.5.3. LA CLASSE OU STRUCTURE POINT

Représente une paire ordonnée de coordonnées X et Y entières qui définit un point dans un plan à deux dimensions. X et Y sont du type Integer.

On utilise principalement ces propriétés publiques :

	Nom	Description
	IsEmpty	Obtient une valeur indiquant si ce Point est vide.
	X	Obtient ou définit la coordonnée x de ce Point.
	Y	Obtient ou définit la coordonnée y de ce Point.

Lorsque l'on veut attribuer des valeurs aux propriétés d'un objet **Point** on utilise à nouveau le **New**. Il est aussi possible de l'utiliser comme une structure.

```
Dim downPos As Point
downPos = New Point(20, 200)
```

Il est aussi possible de considérer Point comme une structure D'ou :

```
Dim TestPoint As Point
TestPoint.X = 13
TestPoint.Y = 83
```

#### 6.5.3.1. POINT, MÉTHODE ToString

La méthode **ToString** permet d'afficher les coordonnées en une fois. Voici un exemple avec l'événement MouseDown de la souris.

```
Dim downPos As Point




Private Sub Panell1_MouseDown(ByVal sender As Object, ByVal
e As System.Windows.Forms.MouseEventArgs) Handles
Panell1.MouseDown
    ' Mémoire les coordonnées
    downPos = e.Location
    TxtPoint.Text = "Point.ToString = " & downPos.ToString
End Sub
```

**Point.ToString = {X=64,Y=55}**

#### 6.5.4. LA CLASSE OU STRUCTURE POINTF

Représente une paire ordonnée de coordonnées X et Y entières qui définit un point dans un plan à deux dimensions. X et Y sont du type **Single**.

On utilise principalement ces propriétés publiques :

	Nom	Description
	<b>IsEmpty</b>	Obtient une valeur indiquant si ce <b>PointF</b> est vide.
	<b>X</b>	Obtient ou définit la coordonnée x de ce <b>PointF</b> .
	<b>Y</b>	Obtient ou définit la coordonnée y de ce <b>PointF</b> .

Lorsque l'on veut attribuer des valeurs aux propriétés d'un objet **PointF** on utilise à nouveau le **New**. Il est aussi possible de l'utiliser comme une structure.

```
Dim MyPos As PointF
```

```
MyPos = New PointF(57.5, 33.5)
```

Il est aussi possible de considérer **PointF** comme une structure D'ou :

```
Dim TestPointF As PointF
TestPointF.X = 13.5
TestPointF.Y = 83.5
```

##### 6.5.4.1. POINTF, MÉTHODE ToString

La méthode **ToString** permet d'afficher les coordonnées en une fois. Voici un exemple avec l'événement **MouseDown** de la souris.

```
Dim downPosF As PointF
```

```
Private Sub Panel2_MouseDown(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles Panel2.MouseDown
```

```
    Dim DeltaF As New PointF(0.25, 0.75)
```

```
    ' Memorise les coordonnées
```

```
    downPosF = e.Location 'valeurs entières
```

```
    downPosF.X = downPosF.X + DeltaF.X
```

```
    downPosF.Y = downPosF.Y + DeltaF.Y
```

```
    TxtPoint.Text = "PointF.ToString = _  
                        " & downPosF.ToString
```

```
End Sub
```

**PointF.ToString = {X=105.25, Y=52.75}**

Remarque : ajout d'un Delta représentant un fragment de pixel pour mettre en évidence le type **Single**.

## 6.6. LES MEMBRES DE LA CLASSE GRAPHICS

La classe **Graphics** possède de nombreux membres, nous nous limiterons aux méthodes permettant de dessiner des lignes et des courbes. Il existe aussi des méthodes permettant de remplir des formes.

### 6.6.1. VUE D'ENSEMBLE DES MÉTHODES

Il faut se référer à la section de l'aide ci-dessous pour découvrir les nombreuses méthodes de la classe Graphics. Comme l'aide détail la liste de surcharge de chaque méthode la liste est interminable. Voici le début:

#### Membres Graphics



[Envoyer des commentaires](#)

Encapsule une surface de dessin GDI+. Cette classe ne peut pas être héritée.

Le type [Graphics](#) expose les membres suivants.

#### Méthodes

	Nom	Description
⇒	<a href="#">AddMetafileComment</a>	Ajoute un commentaire au <a href="#">Metafile</a> en cours.
⇒	<a href="#">BeginContainer</a>	Enregistre un conteneur graphique avec l'état actuel de ce <a href="#">Graphics</a> et ouvre et utilise un nouveau conteneur graphique.
⇒	<a href="#">BeginContainer(Rectangle, Rectangle, GraphicsUnit)</a>	Enregistre un conteneur graphique avec l'état actuel de ce <a href="#">Graphics</a> et ouvre et utilise un nouveau conteneur graphique avec la transformation d'échelle spécifiée.
⇒	<a href="#">BeginContainer(RectangleF, RectangleF, GraphicsUnit)</a>	Enregistre un conteneur graphique avec l'état actuel de ce <a href="#">Graphics</a> et ouvre et utilise un nouveau conteneur graphique avec la transformation d'échelle spécifiée.
⇒	<a href="#">Clear</a>	Efface l'intégralité de la surface de dessin et la remplit avec la couleur d'arrière-plan spécifiée.
⇒	<a href="#">CopyFromScreen(Point, Point, Size)</a>	Effectue le transfert par bloc de bits des données de couleur correspondant à un rectangle de pixels de l'écran à la surface de dessin de <a href="#">Graphics</a> .
⇒	<a href="#">CopyFromScreen(Point, Point, Size, CopyPixelOperation)</a>	Effectue le transfert par bloc de bits des données de couleur correspondant à un rectangle de pixels de l'écran à la surface de dessin de <a href="#">Graphics</a> .

### 6.6.2. GRAPHICS, MÉTHODE DRAWLINE

Dessine une ligne reliant les deux points spécifiés par deux paires de coordonnées. Il y a plusieurs façons d'exprimer les coordonnées. Pour l'aide recherchez sous **Graphics.DrawLine**

#### Graphics .DrawLine, méthode



[Envoyer des commentaires](#)

Dessine une ligne reliant les deux points spécifiés par les paires de coordonnées.

#### Liste de surcharge

	Nom	Description
	<code>DrawLine(Pen, Point, Point)</code>	Dessine une ligne reliant deux structures <code>Point</code> .
	<code>DrawLine(Pen, PointF, PointF)</code>	Dessine une ligne reliant deux structures <code>PointF</code> .
	<code>DrawLine(Pen, Int32, Int32, Int32, Int32)</code>	Dessine une ligne reliant les deux points spécifiés par les paires de coordonnées.
	<code>DrawLine(Pen, Single, Single, Single, Single)</code>	Dessine une ligne reliant les deux points spécifiés par les paires de coordonnées.

#### 6.6.2.1. DRAWLINE, FORME (PEN, POINT, POINT)

Le prototype de la méthode est le suivant :

```
Public Sub DrawLine ( _
    pen As Pen, _
    pt1 As Point, _
    pt2 As Point _
)
```

##### 6.6.2.1.1. Paramètres, DrawnLine(Pen, Point, Point)

*pen* du type `System.Drawing.Pen` détermine la couleur, la largeur et le style de la ligne.

*pt1* du type `System.Drawing.Point` représente le premier point à relier.

*pt2* du type `System.Drawing.Point` représente le deuxième point à relier.

##### 6.6.2.2. DRAWLINE, FORME (PEN, POINTF, POINTF)

Même principe que la forme `DrawLine(Pen, Point, Point)` mais en utilisant la structure `PointF` à la place de `Point`. `PointF` utilise le type `Single` pour X et Y.

**6.6.2.3. DRAWLINE, FORME (PEN, INT32, INT32, INT32, INT32)**

Le prototype de la méthode est le suivant :

```
Public Sub DrawLine ( _
    pen As Pen, _
    x1 As Integer, _
    y1 As Integer, _
    x2 As Integer, _
    y2 As Integer _
)
```

**6.6.2.3.1. Paramètres, DrawnLine(Pen, Integer, Integer, Integer, Integer)**

*pen* du type System.Drawing.Pen détermine la couleur, la largeur et le style de la ligne.

*x1* du type Integer représente la coordonnée X du premier point.

*y1* du type Integer représente la coordonnée Y du premier point.

*x2* du type Integer représente la coordonnée X du deuxième point.

*y2* du type Integer représente la coordonnée Y du deuxième point.

**6.6.2.4. DRAWLINE, FORME (PEN, SINGLE, SINGLE, SINGLE, SINGLE)**

Même principe que la forme DrawnLine(Pen, Integer, Integer, Integer, Integer), mais les coordonnées sont exprimées avec des types **Single**. Le type Single est un type flottant simple précision équivalent au float du C.

**6.6.2.5. DRAWLINE EXEMPLE**

Dans cet exemple, c'est la forme DrawLine (Pen, Point, Point) qui est utilisée.

En relation avec les événements de la souris, une droite est tracée entre la position donnée par le MouseDown et celle fournie par le MouseUp.

Voici le code complet du programme permettant de tester les différentes fonctions graphiques:

```
Dim MyGraph As Graphics
Dim ThePen As Pen
Dim PosDown, PosUp As Point

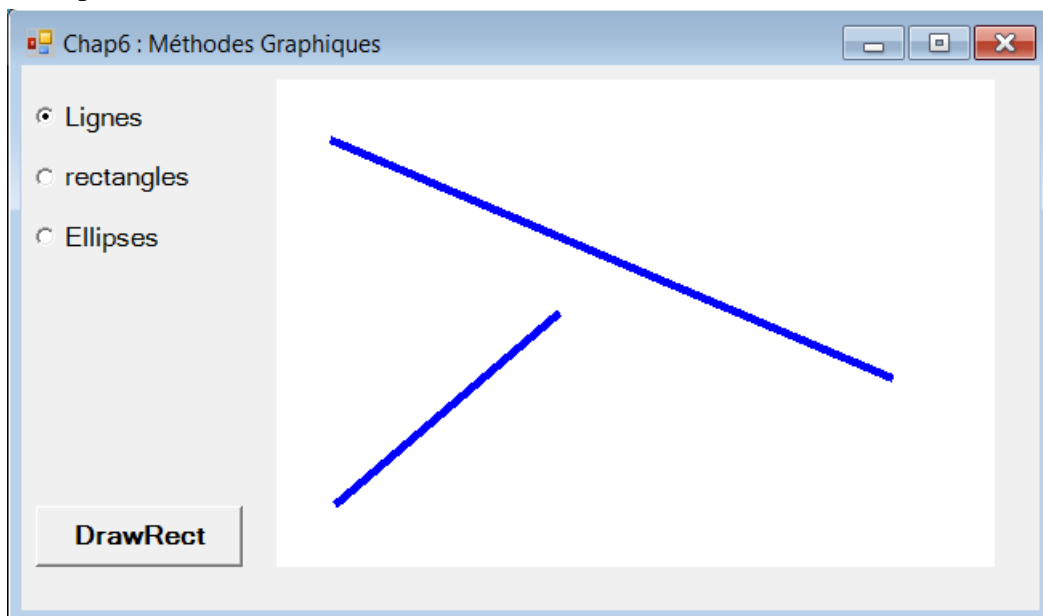
Private Sub Form1_Load(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles MyBase.Load
    MyGraph = Panel1.CreateGraphics
    ThePen = New Pen(Color.blue)
    ThePen.Width = 5
End Sub
```



```
Private Sub Panell1_MouseDown(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles Panell1.MouseDown
    'Memo position lors du MouseDown
    PosDown = New Point(e.X, e.Y)
End Sub
```

```
Private Sub Panell1_MouseUp(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles Panell1.MouseUp
    'Memo position lors du mouseUp
    PosUp = e.Location
    'Dessine en fonction du choix
    If OptLines.Checked Then
        MyGraph.DrawLine(ThePen, PosDown, PosUp)
    End If
End Sub
```

Exemple de résultat:



### 6.6.3. GRAPHICS, MÉTHODE DRAWRECTANGLE




Dessine un rectangle spécifié par une paire de coordonnées, une largeur et une hauteur. Il y a plusieurs façons d'exprimer les coordonnées. Pour l'aide recherchez sous **Graphics.DrawRectangle**.

## Graphics .DrawRectangle, méthode Visual Studio

[Envoyer des commentaires](#)

Dessine un rectangle spécifié par une paire de coordonnées, une largeur et une hauteur.

### Liste de surcharge

	Nom	Description
	<a href="#">DrawRectangle(Pen, Rectangle)</a>	Dessine un rectangle spécifié par une structure <a href="#">Rectangle</a> .
	<a href="#">DrawRectangle(Pen, Int32, Int32, Int32, Int32)</a>	Dessine un rectangle spécifié par une paire de coordonnées, une largeur et une hauteur.
	<a href="#">DrawRectangle(Pen, Single, Single, Single, Single)</a>	Dessine un rectangle spécifié par une paire de coordonnées, une largeur et une hauteur.

#### 6.6.3.1. DRAWRECTANGLE, FORME (PEN, RECTANGLE)

Le prototype de la méthode est le suivant :

```
Public Sub DrawRectangle ( _
    pen As Pen, _
    rect As Rectangle _
)
```

##### 6.6.3.1.1. Paramètres, DrawRectangle(Pen, Rectangle)

**pen** du type System.Drawing.Pen détermine la couleur, la largeur et le style de la ligne.  
**rect** du type System.Drawing.Rectangle représente le rectangle à dessiner.

##### 6.6.3.1.2. La Classe ou Structure rectangle

Cette structure permet de définir le rectangle par les 4 propriétés suivantes :

**X** du type Integer détermine la Coordonnée X de l'angle supérieur gauche du rectangle à dessiner.

**Y** du type Integer détermine la Coordonnée Y de l'angle supérieur gauche du rectangle à dessiner.

**Width** du type Integer détermine la largeur du rectangle à dessiner.

**Height** du type Integer détermine la hauteur du rectangle à dessiner.

Voici la liste des propriétés :

	Nom	Description
	Bottom	Obtient la coordonnée y qui est la somme des valeurs de propriété Y et Height de cette structure Rectangle.
	Height	Obtient ou définit la hauteur de cette structure Rectangle.
	IsEmpty	Teste si toutes les propriétés numériques de ce Rectangle ont des valeurs de zéro.
	Left	Obtient la coordonnée x du bord gauche de cette structure Rectangle.
	Location	Obtient ou définit les coordonnées de l'angle supérieur gauche de cette structure Rectangle.
	Right	Obtient la coordonnée x qui est la somme des valeurs de propriété X et Width de cette structure Rectangle.
	Size	Obtient ou définit la taille de ce Rectangle.
	Top	Obtient la coordonnée y du bord supérieur de cette structure Rectangle.
	Width	Obtient ou définit la largeur de cette structure Rectangle.
	X	Obtient ou définit la coordonnée x de l'angle supérieur gauche de cette structure Rectangle.
	Y	Obtient ou définit la coordonnée y de l'angle supérieur gauche de cette structure Rectangle.

Les propriétés autres que X, Y, Width et Height permettent d'obtenir des informations sur la structure.

### Exemple de déclaration :

```
Dim rect As New Rectangle(10, 20, 300, 200)
```

OU

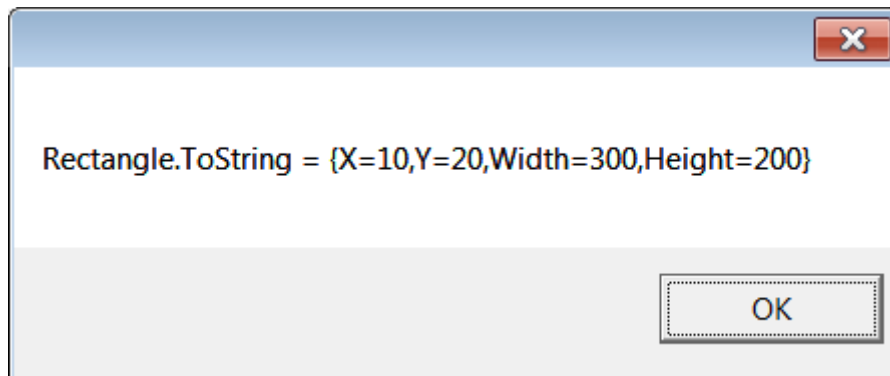
```
Dim rect As Rectangle
```

```
rect.X = 10
rect.Y = 20
rect.Width = 300
rect.Height = 200
```

Affichage des valeurs (méthode **ToString**)

```
MessageBox.Show("Rectangle.ToString = " & rect.ToString)
```

Avec pour résultat :



#### 6.6.3.2. DRAWRECTANGLE, FORME (PEN, INT32, INT32, INT32, INT32)

Le prototype de la méthode est le suivant :

```
Public Sub DrawRectangle ( _
    pen As Pen, _
    x As Integer, _
    y As Integer, _
    width As Integer, _
    height As Integer _
)
```

##### 6.6.3.2.1. Paramètres, DrawRectangle(Pen, Integer, Integer, Integer, Integer)

*pen* du type System.Drawing.Pen détermine la couleur, la largeur et le style de la ligne.

*x* du type Integer détermine la Coordonnée X de l'angle supérieur gauche du rectangle à dessiner.

*y* du type Integer détermine la Coordonnée Y de l'angle supérieur gauche du rectangle à dessiner.

*width* du type Integer détermine la largeur du rectangle à dessiner.

*height* du type Integer détermine la hauteur du rectangle à dessiner.

##### 6.6.3.3. DRAWRECTANGLE, FORME (PEN, SINGLE, SINGLE, SINGLE, SINGLE)

Même principe que la forme DrawRectangle(Pen, Integer, Integer, Integer, Integer), mais avec des valeur du type **Single**.

##### 6.6.3.4. DRAWRECTANGLE, EXEMPLE

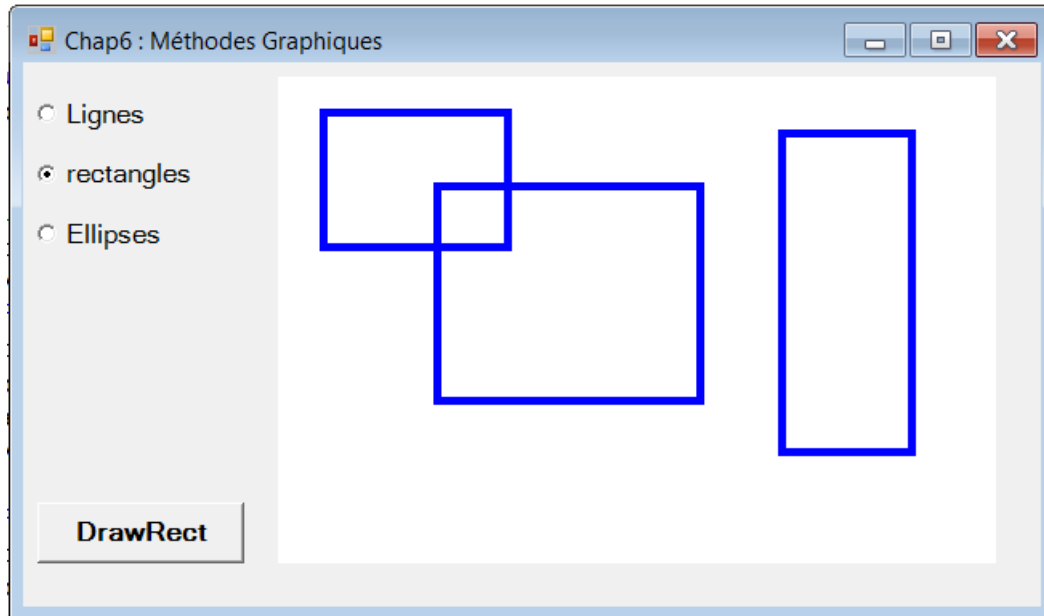
Dans cet exemple, c'est la forme DrawRectangle (Pen, Integer, Integer, Integer, Integer) qui est utilisée.

N'est fourni que le complément de l'exemple fourni sous DrawLine. Le code est placé dans la procédure Sub Panel1\_MouseUp.

```
Dim Longueur, Hauteur As Integer
```

```
If OptRectangle.Checked Then
    Longueur = PosUp.X - PosDown.X
    Hauteur = PosUp.Y - PosDown.Y
    MyGraph.DrawRectangle(ThePen, PosDown.X, PosDown.Y, _
                          Longueur, Hauteur)
End If
```

#### 6.6.3.4.1. DrawRectangle, exemple de résultat



☹ Remarque : si avec la souris, le rectangle est sélectionné de manière à avoir une largeur ou une hauteur négative, on n'obtient pas de dessin.

### 6.6.3.5. UTILISATION DE LA STRUCTURE RECTANGLE

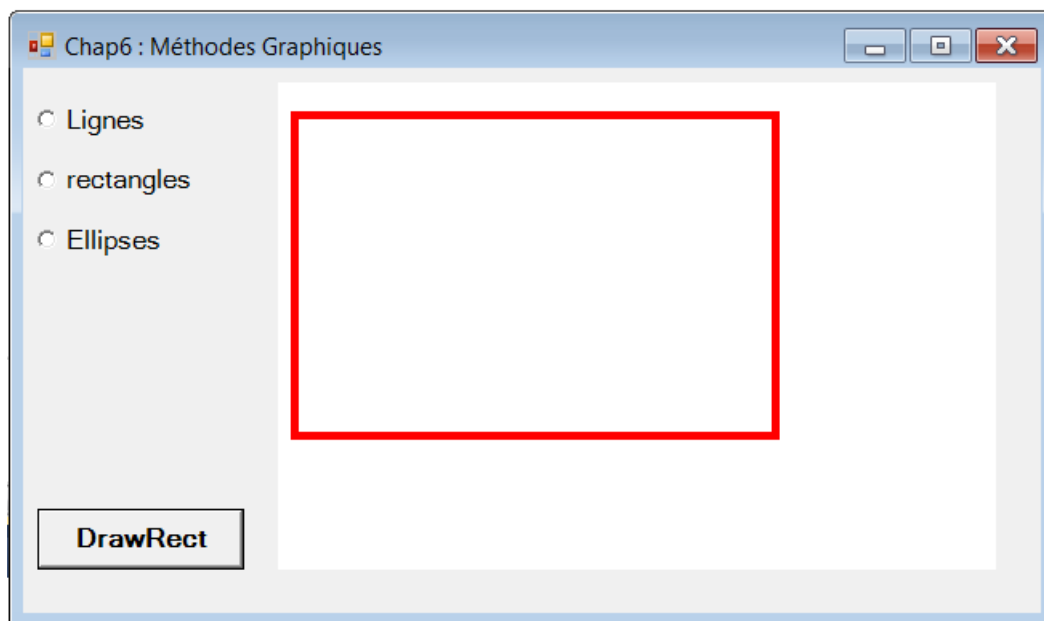
Voici un exemple traçant un rectangle dont la taille est fixe. Ces actions sont liées au bouton DrawRect.

```
Private Sub cmdDrawRect_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
cmdDrawRect.Click
    Dim rect As New Rectangle(10, 20, 300, 200)
    ' Affiche les infos du rectangles
    MessageBox.Show("Rectangle.ToString = " & _
                    rect.ToString)

    ' Dessine le rectangle en rouge dans le Panel.
    ThePen.Color = Color.Red
    MyGraph.DrawRectangle(ThePen, rect)
End Sub
```

Les coordonnées sont relatives à la zone du Panel. Pour le coin supérieur gauche X=10 et Y=20, une largeur de 300 et une hauteur de 200.

Résultat :



### 6.6.4. GRAPHICS, MÉTHODE DRAWELLIPSE

Dessine une ellipse définie par un rectangle englobant, spécifié par une paire de coordonnées, une hauteur et une largeur. Plusieurs formes sont disponible pour la méthode. Aide: utilisez **Graphics.DrawEllipse**.





#### Graphics .DrawEllipse, méthode



[Envoyer des commentaires](#)

Dessine une ellipse définie par un rectangle englobant spécifié par une paire de coordonnées, une hauteur et une largeur.

#### Liste de surcharge

	Nom	Description
	<a href="#">DrawEllipse(Pen, Rectangle)</a>	Dessine une ellipse spécifiée par une structure de délimitation <a href="#">Rectangle</a> .
	<a href="#">DrawEllipse(Pen, RectangleF)</a>	Dessine une ellipse définie par un <a href="#">RectangleF</a> de délimitation.
	<a href="#">DrawEllipse(Pen, Int32, Int32, Int32, Int32)</a>	Dessine une ellipse définie par un rectangle englobant spécifié par les coordonnées du coin supérieur gauche du rectangle, une largeur et une hauteur.
	<a href="#">DrawEllipse(Pen, Single, Single, Single, Single)</a>	Dessine une ellipse définie par un rectangle englobant spécifié par une paire de coordonnées, une hauteur et une largeur.

#### 6.6.4.1. DRAWELLIPSE, FORME (PEN, RECTANGLE)

Le prototype de la méthode est le suivant :

```
Public Sub DrawEllipse ( _  
    pen As Pen, _  
    rect As Rectangle _  
)
```

##### 6.6.4.1.1. Paramètres, DrawEllipse(Pen, Rectangle)

*pen* du type System.Drawing.Pen détermine la couleur, la largeur et le style de la ligne.  
*rect* du type System.Drawing.Rectangle qui définit les limites de l'ellipse à dessiner.

##### 6.6.4.2. DRAWELLIPSE, FORME (PEN, RECTANGLEF)

Même principe que la forme DrawEllipse(Pen, Rectangle) mais utilisation de la structure RectangleF qui utilise des Single pour les coordonnées.

**6.6.4.3. DRAWELLIPSE, FORME (PEN, INT32, INT32, INT32, INT32)**

Le prototype de la méthode est le suivant :

```
Public Sub DrawEllipse ( _
    pen As Pen, _
    x As Integer, _
    y As Integer, _
    width As Integer, _
    height As Integer _
)
```

**6.6.4.3.1. Paramètres, DrawEllipse(Pen, Integer, Integer, Integer, Integer)**

*pen* du type System.Drawing.Pen détermine la couleur, la largeur et le style de la ligne.

*x* du type Integer détermine la Coordonnée X de l'angle supérieur gauche du rectangle enveloppant l'ellipse à dessiner.

*y* du type Integer détermine la Coordonnée Y de l'angle supérieur gauche du rectangle enveloppant l'ellipse à dessiner.

*width* du type Integer détermine la largeur du rectangle enveloppant l'ellipse à dessiner.

*height* du type Integer détermine la hauteur du rectangle enveloppant l'ellipse à dessiner.

**6.6.4.4. DRAWELLIPSE, FORME (PEN, SINGLE, SINGLE, SINGLE, SINGLE)**

Même principe que la forme DrawEllipse (Pen, Integer, Integer, Integer, Integer), mais avec des valeurs du type **Single**.

**6.6.4.5. DRAWELLIPSE, EXEMPLE**

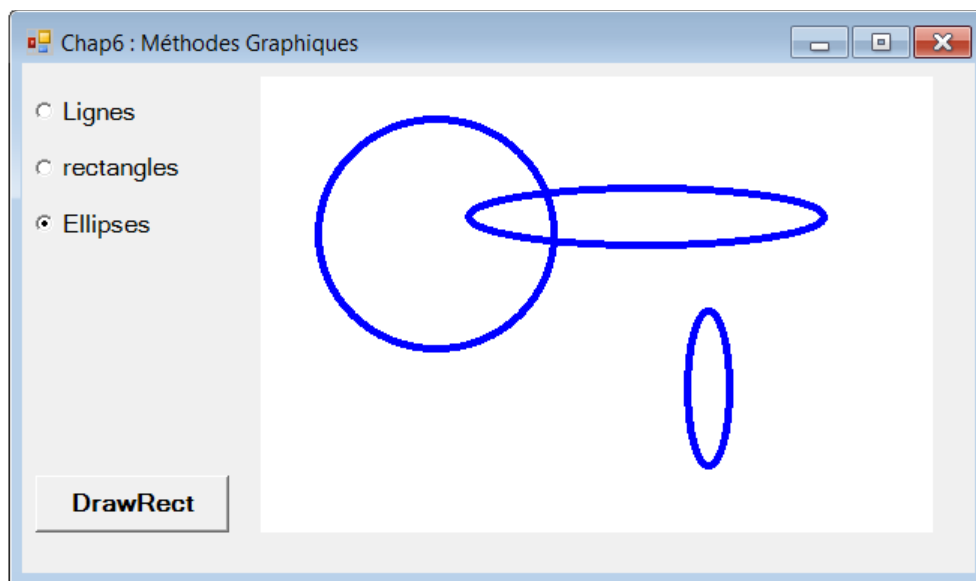
Dans cet exemple, c'est la forme **DrawElippse** (Pen, Integer, Integer, Integer, Integer) qui est utilisée.

```
Dim Longueur, Hauteur As Integer
```

```
If OptEllipse.Checked Then
    Longueur = PosUp.X - PosDown.X
    Hauteur = PosUp.Y - PosDown.Y
    MyGraph.DrawEllipse(ThePen, PosDown.X, PosDown.Y,
                        Longueur, Hauteur)
End If
```



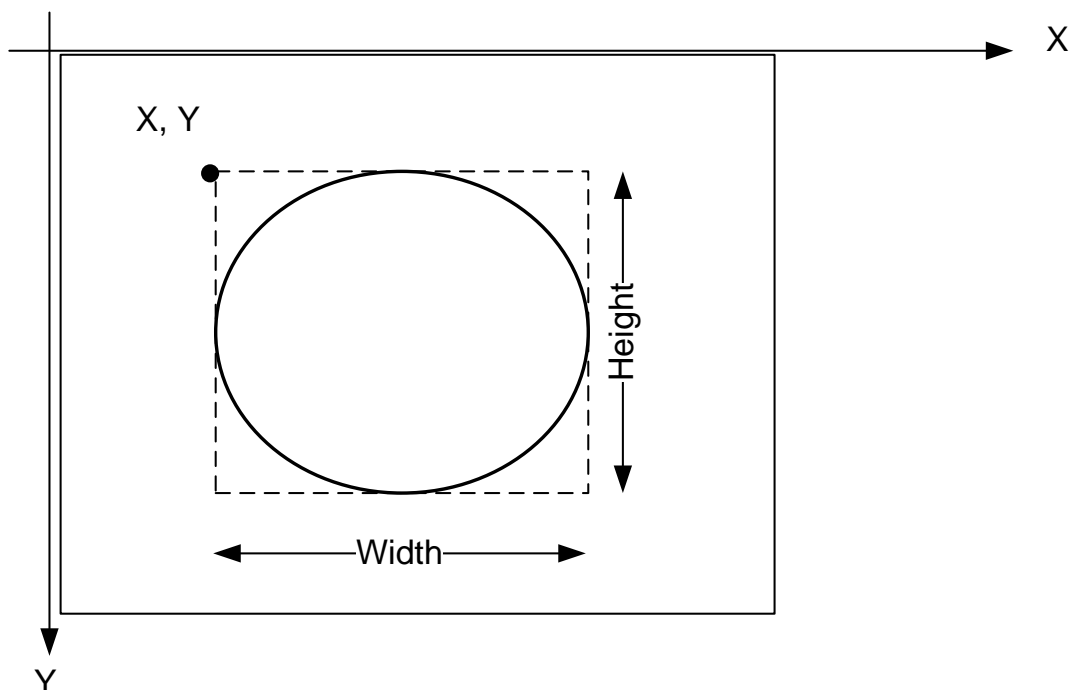
Résultat :



😊 Remarque : si avec la souris, le rectangle est sélectionné de manière à avoir une largeur ou une hauteur négative, on obtient l'ellipse contrairement à la fonction DrawRectangle.

#### 6.6.5. ILLUSTRATION DU DESSIN D'UNE ELLIPSE

La figure ci-dessous montre comment s'organise les axes X et Y d'un graphique et comment est spécifiée une ellipse.



On constate que Y croît vers le bas, le coin en haut à gauche d'une zone de dessin a pour valeur  $X = 0$  et  $Y = 0$ .

Remarque : si on souhaite placer le centre de l'ellipse à une position donnée, il est nécessaire par calcul de obtenir la position X, Y du coin en haut à gauche.

## 6.7. GRAPHIQUES AU NIVEAU DU PIXEL, CLASSE BITMAP

La classe `System.Drawing.Bitmap` encapsule une bitmap GDI+ qui est composée des données de pixels d'une image graphique et de ses attributs. Un objet **Bitmap** est l'objet utilisé pour manipuler des images définies par des données de pixels.

Pour manipuler une image avec GDI+, faut :

- Créez un objet représentant l'image à afficher. Cet objet doit être un membre d'une classe héritant de la classe **Image**, comme **Bitmap** ou **MetaFile**. Voici un exemple :  
`Dim myBitmap as New Bitmap(My.Resources.drapeau_suisse2)`
- Créez un objet **Graphics** représentant la surface de dessin à utiliser. Par exemple :  
`MyGraph = PictureBox1.CreateGraphics`
- Appelez la méthode `Graphics.DrawImage` de l'objet **Graphics** pour rendre l'image. Vous devez spécifier à la fois l'image à dessiner et les coordonnées indiquant sa position ainsi que sa dimension.

```
MyGraph.DrawImage(MyBitmap, 0, 0, PictureBox1.Width,
                  PictureBox1.Height)
```

Ensuite il est possible de modifier l'image en utilisant par exemple la méthode `SetPixel`

### 6.7.1. MÉTHODE SETPIXEL

Cette méthode de la classe **Bitmap** permet de modifier la couleur d'un pixel dont on fournit les coordonnées. Le prototype de la fonction est le suivant :

```
Public Sub SetPixel ( _
    x As Integer, _
    y As Integer, _
    color As Color _
)
```

**x** du type `Integer` : coordonnée en x du pixel à définir.

**y** du type `Integer` : coordonnée en y du pixel à définir.

**color** du type `Color` : représente la couleur à assigner au pixel spécifié.

#### 6.7.1.1. SETPIXEL, EXEMPLE

Voici un exemple complet permettant de réaliser du dessin à main levée sur une image.

##### 6.7.1.1.1. SetPixel, exemple, éléments globaux

Introduction d'un objet de type `Bitmap`.

```
Dim MyBitmap As Bitmap
Dim MyGraph As Graphics
Dim PaintOk As Boolean = False
```

**6.7.1.1.2. SetPixel, exemple, action form\_load**

Il est nécessaire d'associer une image à l'objet BitMap. De même qu'il est nécessaire d'utiliser l'objet BitMap avec la méthode DrawImage de l'objet Graphics. Il faut adapter la dimension de la PictureBox à celle de l'image pour que la position de la souris corresponde au pixel visé.

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    MyBitmap = New Bitmap(My.Resources.drapeau_suisse2)
    'Dimensionnement de la PictureBox en fonction de la
    'taille de l'image
    PictureBox1.Width = MyBitmap.Width
    PictureBox1.Height = MyBitmap.Height
    MyGraph = PictureBox1.CreateGraphics
    MyGraph.DrawImage(MyBitmap, 0, 0, PictureBox1.Width, _
PictureBox1.Height)
End Sub
```

**6.7.1.1.3. SetPixel, exemple, action de dessin**

Combinaison du SetPixel et du DrawImage qui sert de rafraichissement de l'image.

Il faut éviter d'appeler la fonction SetPixel en lui donnant des coordonnées hors de la zone image, c'est la raison des tests effectués ceci malgré l'adaptation de la taille du PictureBox à celle de l'image.

```
Private Sub PictureBox1_MouseMove(ByVal sender As Object,
ByVal e As System.Windows.Forms.MouseEventArgs) Handles
PictureBox1.MouseMove
    If PaintOk = True Then
        If (e.X < MyBitmap.Width) And (e.X > 0) And _
            (e.Y < MyBitmap.Height) And (e.Y > 0)
        Then
            MyBitmap.SetPixel(e.X, e.Y, Color.Black)
            MyGraph.DrawImage(MyBitmap, 0, 0,
                PictureBox1.Width, PictureBox1.Height)
        End If
    End If
End Sub

Private Sub PictureBox1_MouseDown(ByVal sender As Object,
ByVal e As System.Windows.Forms.MouseEventArgs) Handles
PictureBox1.MouseDown
    PaintOk = True
    MyGraph.DrawImage(MyBitmap, 0, 0, PictureBox1.Width, _
        PictureBox1.Height)
End Sub
```

```
Private Sub PictureBox1_MouseUp(ByVal sender As Object,  
ByVal e As System.Windows.Forms.MouseEventArgs) Handles  
PictureBox1.MouseUp  
    PaintOk = False  
End Sub
```

#### 6.7.1.2. SETPIXEL, EXEMPLE DE RÉSULTAT

C'est lors du MouseDown que l'image du drapeau apparaît.

Dès que l'on se déplace avec la souris, la trajectoire est marquée au niveau du pixel.



Cet exemple montre la possibilité de dessiner à main levée avec la souris sur une image.

## 6.8. LA METHODE DRAWSTRING

La méthode **DrawString** de l'objet Graphics remplace la méthode Print de Visual Basic 6.0. Alors que la méthode **Print** utilisait un seul argument Text, DrawString utilise en plus des arguments qui spécifient la Police, un objet **Brush** qui spécifie un objet Color et les coordonnées de départ pour dessiner le texte.

La méthode **DrawString** possède plusieurs formes d'appel :




### Graphics .DrawString, méthode



[Envoyer des commentaires](#)

Dessine la chaîne de texte précisée à l'emplacement indiqué avec les objets **Brush** et **Font** spécifiés.

#### Liste de surcharge

	Nom	Description
	<code>DrawString(String, Font, Brush, PointF)</code>	Dessine la chaîne de texte précisée à l'emplacement indiqué avec les objets <b>Brush</b> et <b>Font</b> spécifiés.
	<code>DrawString(String, Font, Brush, RectangleF)</code>	Dessine la chaîne de texte précisée dans le rectangle indiqué avec les objets <b>Brush</b> et <b>Font</b> spécifiés.
	<code>DrawString(String, Font, Brush, PointF, StringFormat)</code>	Dessine la chaîne de texte précisée à l'emplacement indiqué avec les objets <b>Brush</b> et <b>Font</b> spécifiés, à l'aide des attributs de mise en forme du <b>StringFormat</b> spécifié.
	<code>DrawString(String, Font, Brush, RectangleF, StringFormat)</code>	Dessine la chaîne de texte précisée dans le rectangle indiqué avec les objets <b>Brush</b> et <b>Font</b> spécifiés, à l'aide des attributs de mise en forme du <b>StringFormat</b> spécifié.
	<code>DrawString(String, Font, Brush, Single, Single)</code>	Dessine la chaîne de texte précisée à l'emplacement indiqué avec les objets <b>Brush</b> et <b>Font</b> spécifiés.
	<code>DrawString(String, Font, Brush, Single, Single, StringFormat)</code>	Dessine la chaîne de texte précisée à l'emplacement indiqué avec les objets <b>Brush</b> et <b>Font</b> spécifiés, à l'aide des attributs de mise en forme du <b>StringFormat</b> spécifié.

#### 6.8.1. DRAWSTRING, PARAMÈTRE STRING

C'est le texte que l'on souhaite afficher

#### 6.8.2. DRAWSTRING, PARAMÈTRE FONT

L'objet Font permet de définir le nom de la police, sa dimension et le style d'écriture.

Lors de la création de l'objet Font il est possible d'utiliser différente forme d'appel. Nous nous limiterons à deux formes d'appel :

**6.8.2.1. FONT (NOM, TAILLE)**

```
Public Sub New( _
    ByVal FamilyName As String, _
    ByVal EmSize As Single _
)
```

*FamilyName* représente de nom de la police

*EmSize* représente la taille de la police

Exemple (police Arial taille 16):

```
Dim drawFont As New Font("Arial", 16)
```

**6.8.2.2. FONT (NOM, TAILLE, STYLE)**

```
Public Sub New( _
    ByVal FamilyName As String, _
    ByVal EmSize As Single _
    ByVal Style As FontStyle _
)
```

*FamilyName* représente de nom de la police.

*EmSize* représente la taille de la police.

*Style* représente le style de la police. Le type énuméré `FontStyle` permet de spécifier les styles suivants :

Nom de membre	Description
<b>Bold</b>	Texte gras.
<b>Italic</b>	Texte italique.
<b>Regular</b>	Texte normal.
<b>Strikeout</b>	Texte barré d'une ligne à mi-hauteur.
<b>Underline</b>	Texte souligné.

Exemple (police Arial, taille 16, Italique):

```
drawFont = New Font("Arial", 16, FontStyle.Italic)
```

**6.8.3. DRAWSTRING, PARAMETRE BRUSH**

Cette classe est une classe de base abstraite qui ne peut pas être instanciée. Pour créer un objet brosse, utilisez les classes dérivées de **Brush**, comme **SolideBrush**, **TextureBrush** et **Drawing2D.HatchBrush**. Il est aussi possible d'utiliser directement la classe **Brushes** qui implémente un jeu d'objets **SolidColorBrush** prédéfinis.

Classe	Comportement
<b>Brushes</b>	Pinceaux prédéfinis avec une seule couleur
<b>SolidBrush</b>	Définit un remplissage avec une seule couleur
<b>TextureBrush</b>	Utilise une image pour remplir l'intérieur d'une forme.

Classe	Comportement
<b>Drawing2D.HatchBrush</b>	Définit un pinceau rectangulaire avec un style de hachurage, une couleur de premier plan et une couleur d'arrière-plan.  Le type énuméré <b>HatchStyle</b> , définit le style de hachurage
<b>Drawing2D.LinearGradientBrush</b>	Permet des dégradés bicolores linéaires.
<b>Drawing2D.PathGradientBrush</b>	Permet un ombrage de couleurs unies allant du point central du tracé à son bord extérieur.

Exemple :

```
Dim drawBrush As New SolidBrush(Color.Black)
```

Usage direct de Brushes :

```
MyGraph.DrawString(text2, drawFont, Brushes.Blue, 10, 30)
```

#### **6.8.4. DRAWSTRING, PARAMÈTRE STRINGFORMAT**

Le paramètre StringFormat permet d'agir sur la mise en page du texte (telles que l'alignement, l'orientation et les taquets de tabulation).

StringFormat est une classe dont nous utiliserons en particulier les propriétés **Alignement** et **LineAlignement**

#### **6.8.5. DRAWSTRING, POSITIONNEMENT DU TEXTE**

Dans les variantes d'appel de DrawString, on trouve trois façons d'indiquer la position du texte.

##### **6.8.5.1. POSITIONNEMENT DU TEXTE : SINGLE, SINGLE**

Une des variantes permet d'indiquer séparément la position X et Y du début du texte à affiché. Par exemple :

```
MyGraph.DrawString(text1, drawFont, drawBrush, 10, 5)
```

##### **6.8.5.2. POSITIONNEMENT DU TEXTE : POINTF**

Une des variantes permet d'indiquer la position X et Y du début du texte à afficher par un objet PointF . Par exemple :

```
Dim PosText as new PointF(10, 5)
```

```
MyGraph.DrawString(text1, drawFont, drawBrush, PosText)
```

##### **6.8.5.3. POSITIONNEMENT DU TEXTE : RECTANGLEF**

Une des variantes permet d'indiquer la position du texte à afficher en fournissant le rectangle qui entoure le texte par un objet RectangleF . Par exemple :

```
Dim ZoneTexte As New RectangleF(10, 120, 180, 50)
```

```
MyGraph.DrawString(text1, drawFont, drawBrush, ZoneTexte)
```

Remarque : si on souhaite dessiner le rectangle, il faut alors définir la zone par une structure `Rectangle`, car la méthode `DrawRectangle` ne supporte pas le `RectangleF`.

### 6.8.6. DRAWSTRING, EXEMPLE1

L'exemple suivant affiche quatre lignes de texte et un texte dans un rectangle, ceci sur le panel, lors de l'activation du bouton de commande.

Les 2 premières lignes sont affichées en Arial 16 de couleur noire. Les 2 lignes suivantes sont affichées en Verdana 16, italique et hachuré.

Le 3<sup>ème</sup> texte est affiché dans un rectangle avec une police Arial de 14. On constate dans ce cas que le texte est positionné en haut à gauche du rectangle et que le renvoi à la ligne est automatique pour ne pas sortir du rectangle.

```
Dim MyGraph As Graphics
```

```
Private Sub CmdDraw_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles CmdDraw.Click
    ' Création des lignes à afficher.
    Dim text1 As String = "Text affiché avec DrawString"
    Dim text2 As String = "2ème ligne à afficher"
    Dim text3 As String = "3ème ligne dans un rectangle"

    ' Creation font et brush.
    Dim drawFont As New Font("Arial", 16)
    Dim drawBrush As New SolidBrush(Color.Black)

    ' Positionnement du text avec un PointF
    Dim Postext As New PointF(10, 5)

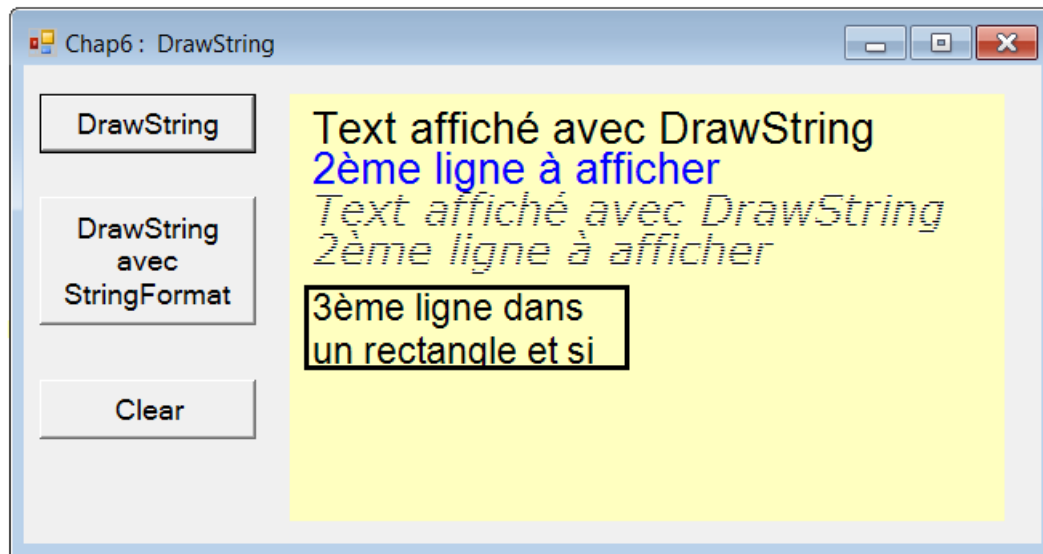
    ' Dessine les textes dans le panel.
    MyGraph.DrawString(text1, drawFont, drawBrush, Postext)
    MyGraph.DrawString(text2, drawFont, drawBrush, 10, 30)

    'Modification font et brush
    drawFont = New Font("Verdana", 16, FontStyle.Italic)
    Dim drawBrush2 As New Drawing2D.HatchBrush _
        (Drawing2D.HatchStyle.NarrowHorizontal, Color.White)
    ' Redessine les textes dans le panel.
    MyGraph.DrawString(text1, drawFont, drawBrush2, 10, 55)
    MyGraph.DrawString(text2, drawFont, drawBrush2, 10, 80)

    ' Cas du text dans le rectangle
    Dim ZoneTexte As New Rectangle(10, 120, 200, 50)
    drawFont = New Font("Arial", 14)
    Dim LaPlume As New Pen(Color.Black, 3)
    MyGraph.DrawRectangle(LaPlume, ZoneTexte)
    MyGraph.DrawString(text3, drawFont, _
        drawBrush, ZoneTexte)
End Sub
```



### 6.8.6.1. DRAWSTRING, EXEMPLE1, RÉSULTAT



👉 On remarque que pour le DrawString dans le rectangle, le texte est tronqué pour ne pas sortir du rectangle.

### 6.8.7. DRAWSTRING, EXEMPLE2

L'exemple suivant affiche un texte dans deux rectangles. Pour le premier, le texte est centré et pour le 2<sup>ème</sup>, le texte est aligné en bas à droite.

```
Private Sub CmdDraw2_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CmdDraw2.Click
    ' Construit le rectangle entourant le texte.
    Dim displayRectangle _
        As New Rectangle(10, 10, 220, 80)

    ' Déclaration de deux objets StringFormat
    Dim format1 As New _
        StringFormat(StringFormatFlags.NoClip)
    Dim format2 As New StringFormat(format1)

    ' Etablit les propriétés LineAlignment et Alignment
    ' pour les deux objets StringFormat
    format1.LineAlignment = StringAlignment.Center
    format1.Alignment = StringAlignment.Center
    format2.LineAlignment = StringAlignment.Far
    format2.Alignment = StringAlignment.Far

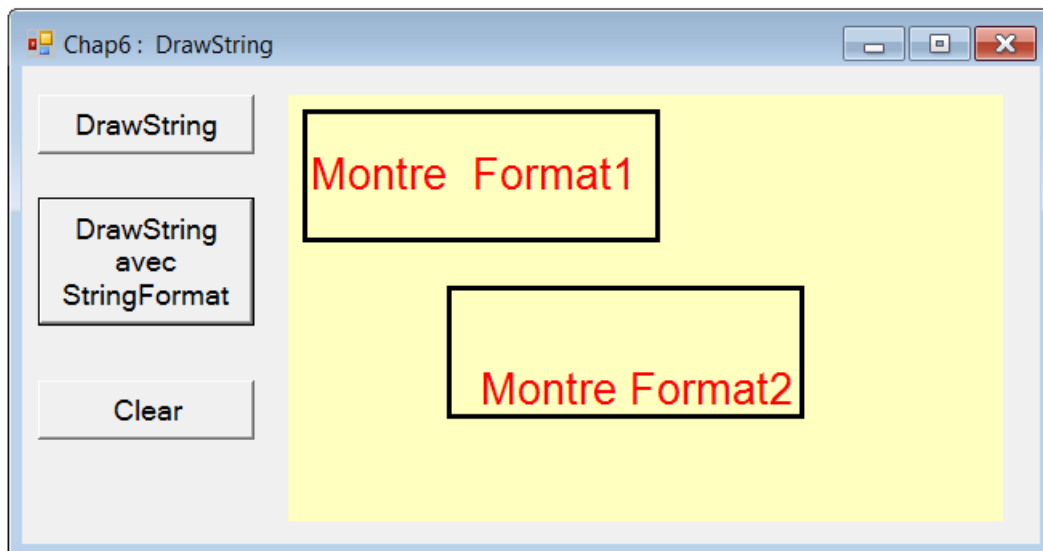
    ' Prépare une police
    Dim drawFont As New Font("Arial", 16)
    ' Prépare une plume
    Dim ThePen As New Pen(Color.Black, 3)

    ' Dessine le 1er rectangle et le texte avec le format1
    MyGraph.DrawRectangle(ThePen, DisplayRectangle)
    MyGraph.DrawString("Montre Format1", drawFont, _
        Brushes.Red, DisplayRectangle, format1)
```

```
' Dessine le 2ème rectangle et le texte avec le format2
displayRectangle.X = 100
displayRectangle.Y = 120
MyGraph.DrawRectangle(ThePen, DisplayRectangle)
MyGraph.DrawString("Montre Format2", drawFont, _
                   Brushes.Red, displayRectangle, format2)

End Sub
```

#### 6.8.7.1. DRAWSTRING, EXEMPLE2, RÉSULTAT








## 6.9. LES BOÎTES DE DIALOGUE COMMUNES

Dans la version VB6 Microsoft offrait, par le biais d'un composant appelé CommonDialog, un control comportant plusieurs fonctions permettant l'activation de boîtes de dialogues prédéfinies comme le choix d'une couleur, l'ouverture d'un fichier ou la sélection d'une police de caractère.

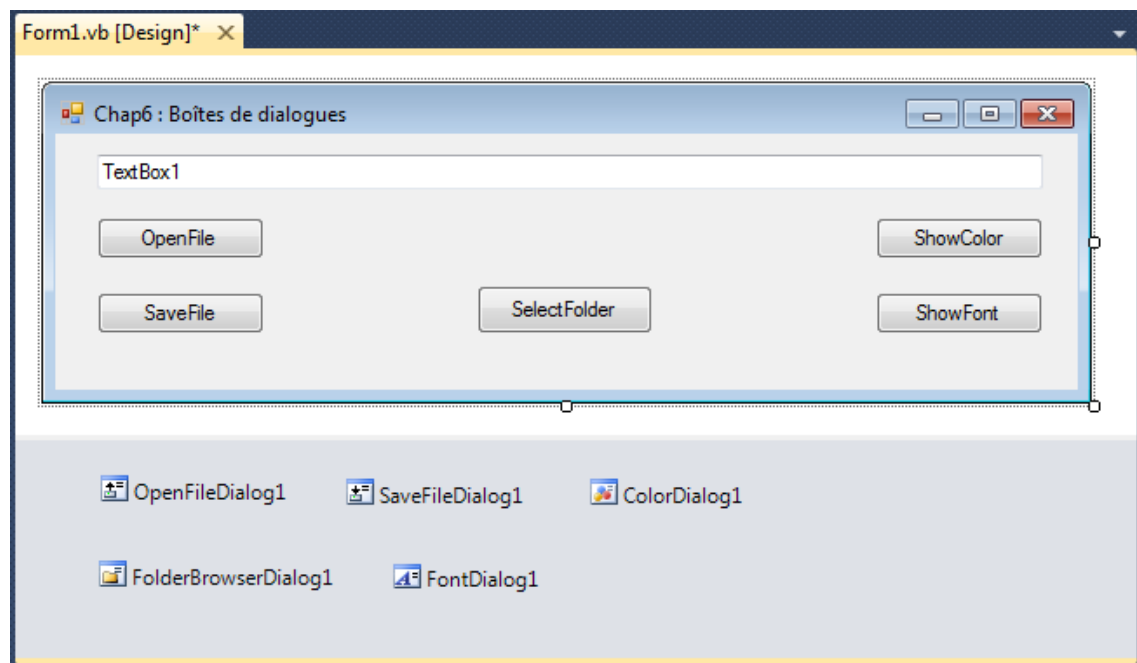
Au niveau du VB 2010, chacune des boîtes de dialogue est devenue un contrôle indépendant faisant partie de la boîte à outils de base.

Voici ce que l'on trouve dans la boîte à outils dans la section Boîtes de dialogue.

Boîtes de dialogue	Contrôle	Boîte de dialogue affichée
 ColorDialog	ColorDialog	Couleur
 FolderBrowserDialog	FolderBrowserDialog	Rechercher un dossier
 FontDialog	FontDialog	Police
 OpenFileDialog	OpenFileDialog	Ouvrir
 SaveFileDialog	SaveFileDialog	Enregistrer sous

### 6.9.1. MISE EN PLACE DES CONTRÔLES

Il faut placer le contrôle sur la feuille, mais celui-ci apparaît dans une zone placée sous la feuille :



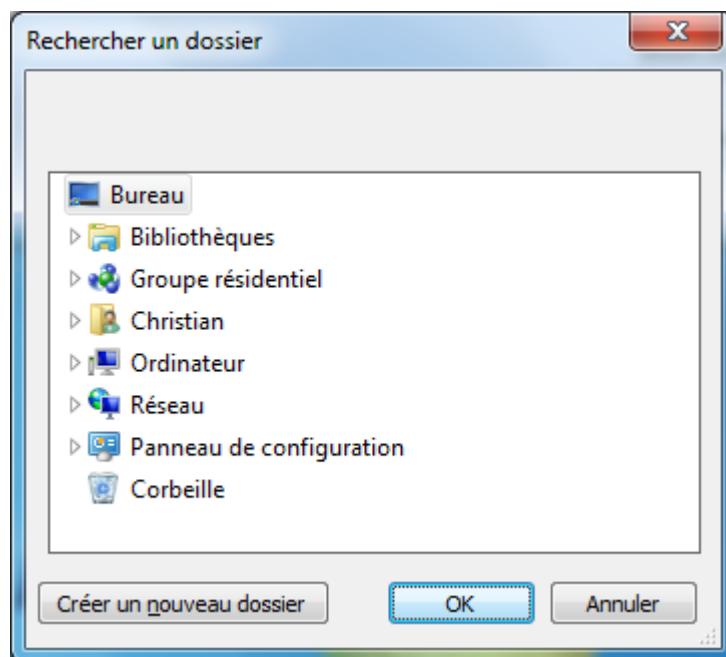
### 6.9.2. LE CONTRÔLE FOLDERBROWSERDIALOG

Ce contrôle affiche une boîte de dialogue permettant la sélection d'un répertoire. Il est aussi possible de créer un répertoire avec ce contrôle.

Si dans la procédure événementielle d'un bouton de commande on ajoute :

```
Private Sub CmdFolder_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles CmdFolder.Click
    FolderBrowserDialog1.ShowDialog()
End Sub
```

A exécution on obtient :



#### 6.9.2.1. UTILISATIONS DES RÉSULTATS

La propriété **SelectedPath** permet de récupérer la sélection effectuée par l'utilisateur grâce à la boîte de dialogue.

A cause de l'existence des boutons OK et annuler, il est nécessaire de tester le résultat du dialogue, d'où l'exemple suivant :

```
Private Sub CmdFolder_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles CmdFolder.Click

    Dim folderName As String
    Dim result As DialogResult

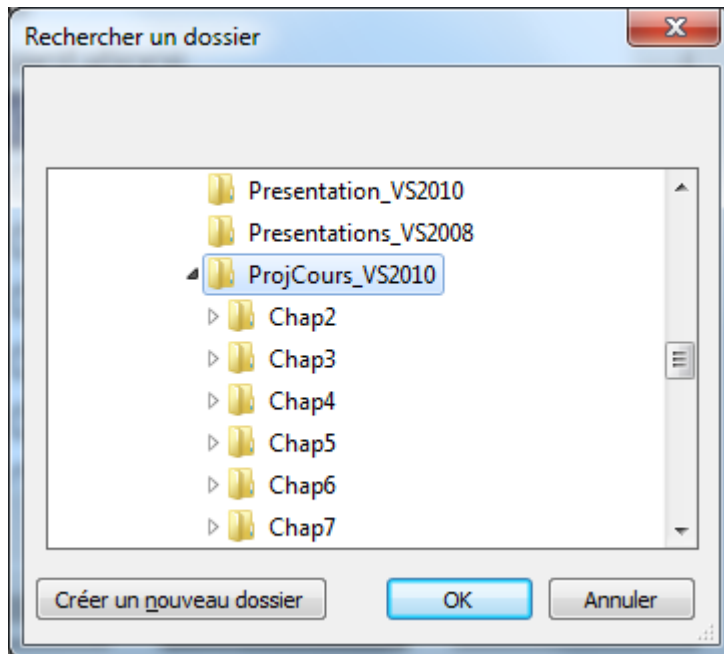
    ' Affiche le dialogue "Rechercher un dossier".
    result = FolderBrowserDialog1.ShowDialog()
```

```

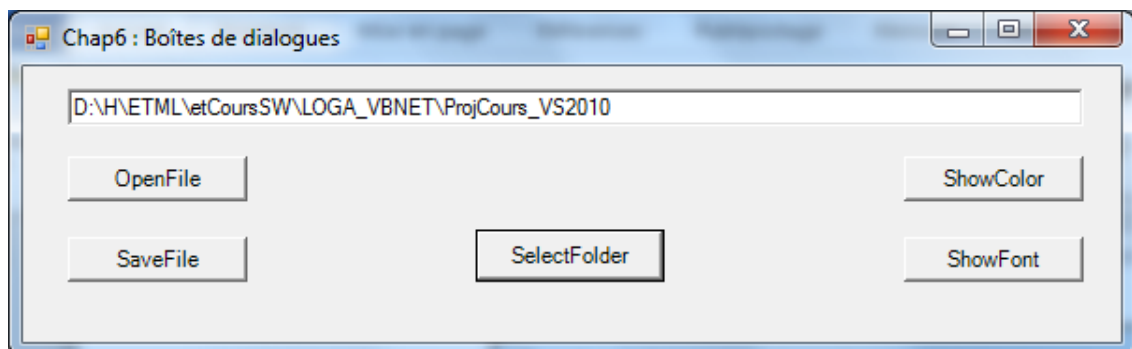
If (result = DialogResult.OK) Then
    folderName = FolderBrowserDialog1.SelectedPath
    ' Montre le chemin obtenu
    TextBox1.Text = folderName
    ' Utilise la sélection comme répertoire initial
    OpenFileDialog1.InitialDirectory = folderName
End If
End Sub

```

Avec la sélection suivante :



On obtient :



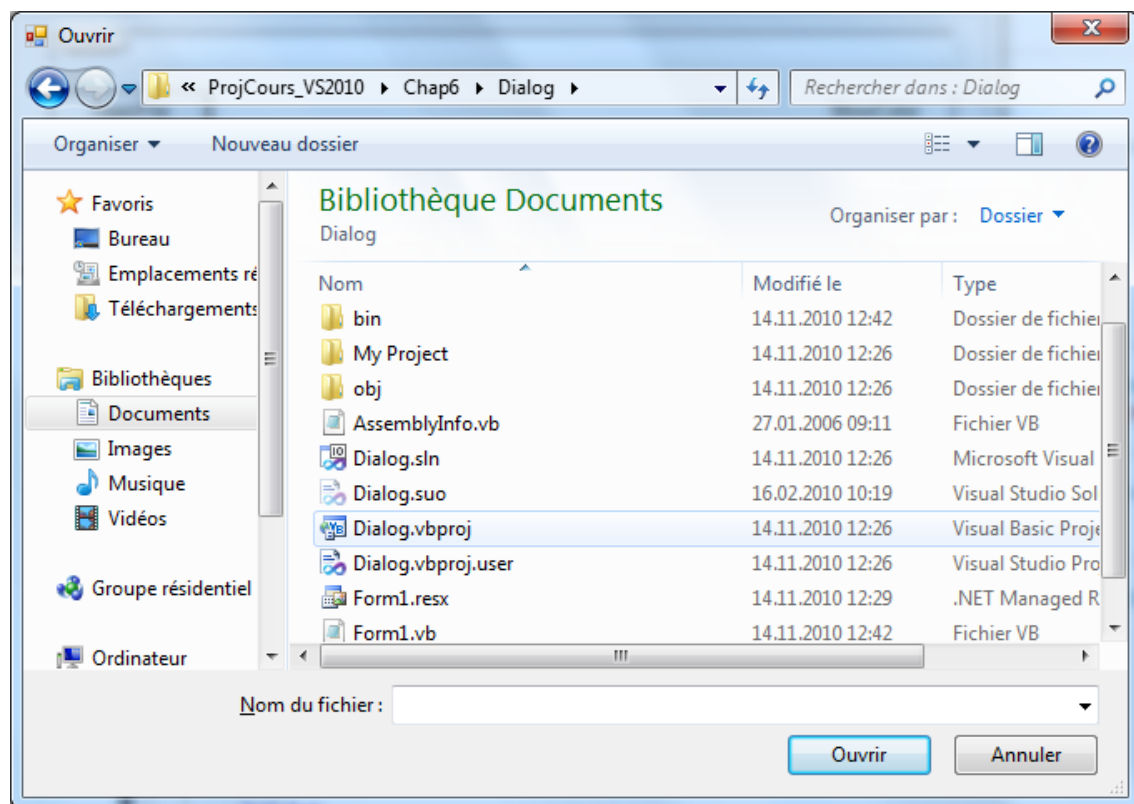
### 6.9.3. LE CONTRÔLE OPENFILEDIALOG

Ce contrôle affiche une boîte de dialogue permettant l'ouverture d'un fichier. Il est possible d'utiliser ce contrôle soit pour obtenir le chemin complet du fichier à ouvrir (propriétés FileName) ou pour l'ouvrir (méthode OpenFile).

Si dans la procédure événementielle d'un bouton de commande on ajoute :

```
Private Sub CmdOpen_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles CmdOpen.Click
    OpenFileDialog1.ShowDialog()
End Sub
```

A l'exécution on obtient :



On obtient la boîte de dialogue **Ouvrir** communes à toutes les applications Windows avec le style propre lié à la version du système d'exploitation utilisé, dans ce cas Windows Seven.

### 6.9.3.1. FILTER, PROPRIÉTÉ (OPENFILEDIALOG)

La propriété **Filter** permet lors de l'exécution du **ShowDialog**, pour un contrôle **OpenFileDialog** ou **SaveFileDialog** de définir les filtres affichés dans la **Erreur ! Des objets ne peuvent pas être créés à partir des codes de champs de mise en forme**.zone à droite du nom de fichier.

Pour chaque option de filtrage, la chaîne de filtrage contient une description du filtre suivie de la barre verticale (|) et du modèle de filtre. Les chaînes correspondant à différentes options de filtrage sont séparées par une barre verticale.

Voici un exemple de chaîne de filtrage :

Text files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*

Vous pouvez ajouter plusieurs modèles de filtre à un filtre en séparant les types de filtres avec des points-virgules, par exemple :

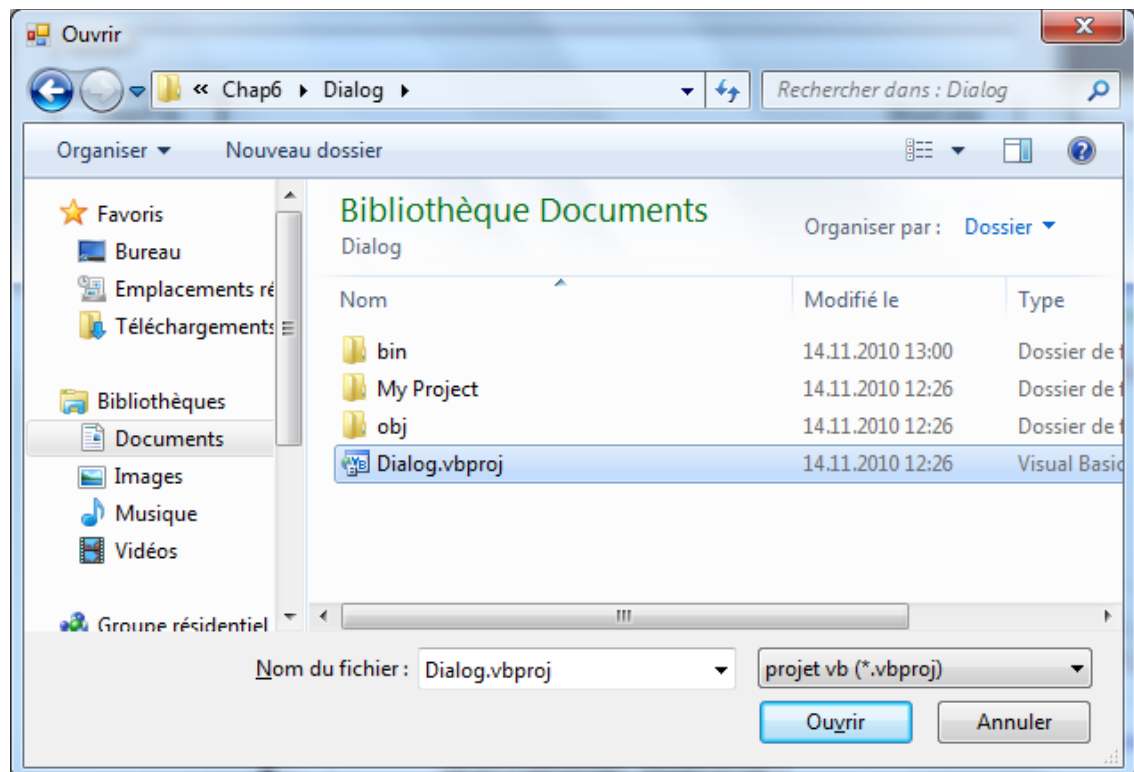
Image Files(\*.BMP;\*.JPG;\*.GIF)|\*.BMP;\*.JPG;\*.GIF|All files (\*.\*)|\*.\*

Utilisez la propriété **FilterIndex** pour définir l'option de filtrage que l'utilisateur voit s'afficher en premier.

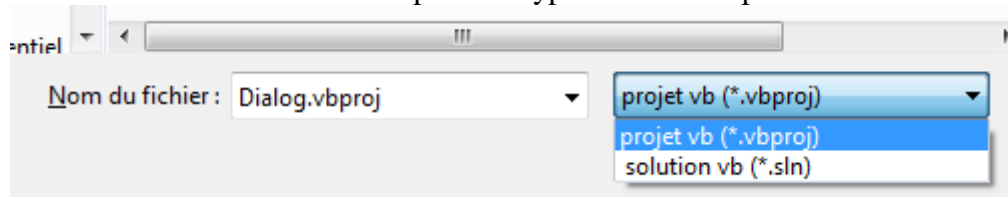
#### 6.9.3.1.1. Filter, exemple

```
Private Sub CmdOpen_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles CmdOpen.Click
    OpenFileDialog1.Filter = _
        "projet vb (*.vbproj)| *.vbproj| solution vb (*.sln)| *.sln"
    OpenFileDialog1.ShowDialog()
End Sub
```

A l'exécution nous obtenons :



On obtient une liste déroulante pour les types de fichiers possible :



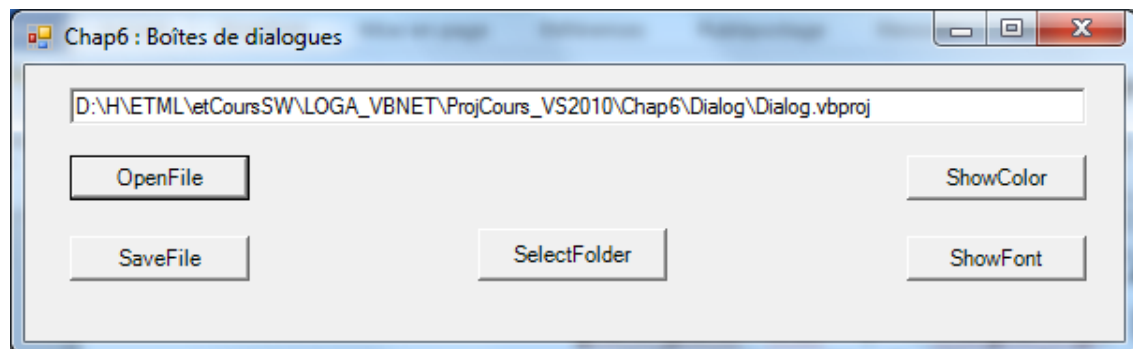
### 6.9.3.2. UTILISATIONS DES RÉSULTATS

La propriété **FileName** permet de récupérer la sélection effectuée par l'utilisateur grâce à la boîte de dialogue. Il est nécessaire de tester le résultat à cause de l'action possible par le bouton "Annuler".

Exemple:

```
Private Sub CmdOpen_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles CmdOpen.Click
    OpenFileDialog1.Filter = _
        "projet vb (*.vbproj)| *.vbproj| solution vb (*.sln)| *.sln"
    If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
        TextBox1.Text = OpenFileDialog1.FileName
    Else
        TextBox1.Text = "Action annulée"
    End If
End Sub
```

Résultat de la propriété FileName :



### 6.9.3.3. INITIALDIRECTORY, PROPRIÉTÉ

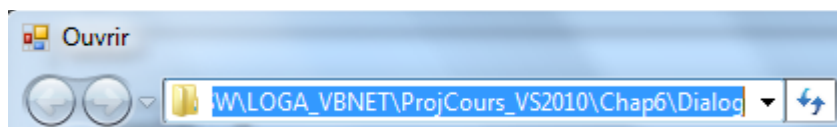
Cette propriété sert à indiquer le dossier initial des boîtes de dialogue Ouvrir et Enregistrer sous. Si cette propriété n'est pas définie, ces boîtes de dialogue utilisent le dossier en cours.

Par exemple, lors du form\_load :

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal
    e As System.EventArgs) Handles MyBase.Load
    OpenFileDialog1.InitialDirectory = _
        "D:\H\ETML\etCoursSW\LOGA_VBNET\ProjCours_VS2010\Chap6\Dialog"
End Sub
```



Résultat lors du lancement de l'OpenFileDialog :



#### **6.9.3.4. OPENFILE, MÉTHODE**

Cette méthode ouvre le fichier sélectionné par l'utilisateur en lecture seule. Le fichier est spécifié par la propriété [FileName](#).

Cette méthode s'utilise après l'exécution de la méthode ShowDialog, qui implique normalement la sélection d'un fichier.

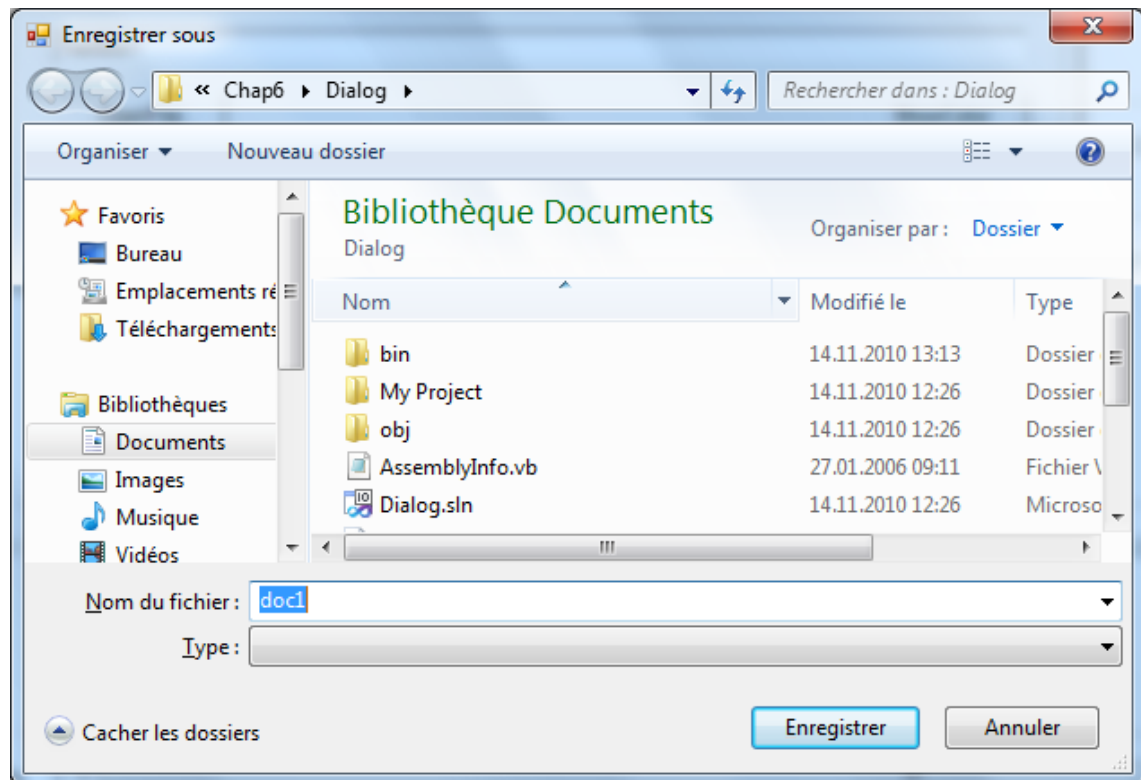
Nous reviendrons sur cette méthode dans le cadre du chapitre sur la gestion des fichiers.

#### 6.9.4. LE CONTRÔLE SAVEFILEDIALOG

Si dans la procédure événementielle d'un bouton de commande on ajoute :

```
Private Sub CmdSave_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles CmdSave.Click
    SaveFileDialog1.ShowDialog()
End Sub
```

A l'exécution on obtient :



Cette boîte de dialogue est très semblable à la boîte de dialogue ouvrir, une différence importante est que la sélection multiple est impossible.

Il est aussi possible d'ajouter un filtre et de récupérer le nom du fichier sélectionné.

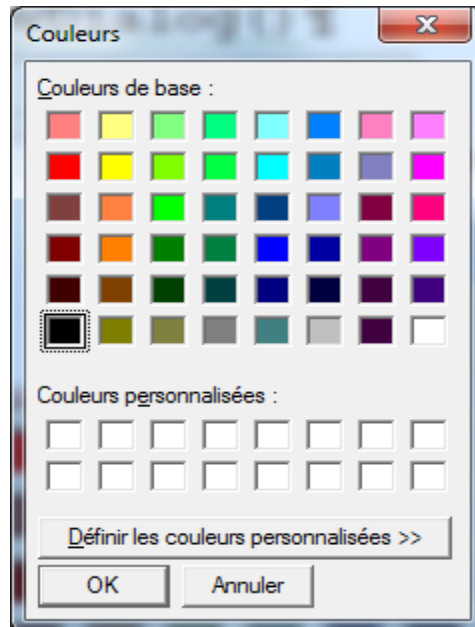
La méthode **SaveFile**, ouvre le fichier sélectionné par l'utilisateur avec les autorisations de lecture/écriture.

### 6.9.5. LE CONTRÔLE COLORDIALOG

Si dans la procédure événementielle d'un bouton de commande on ajoute la ligne suivante :

```
Private Sub CmdColor_Click(ByVal sender As System.Object,  
ByVal e As System.EventArgs) Handles CmdColor.Click  
    ColorDialog1.ShowDialog()  
End Sub
```

On obtient l'ouverture de la boîte de dialogue suivante :

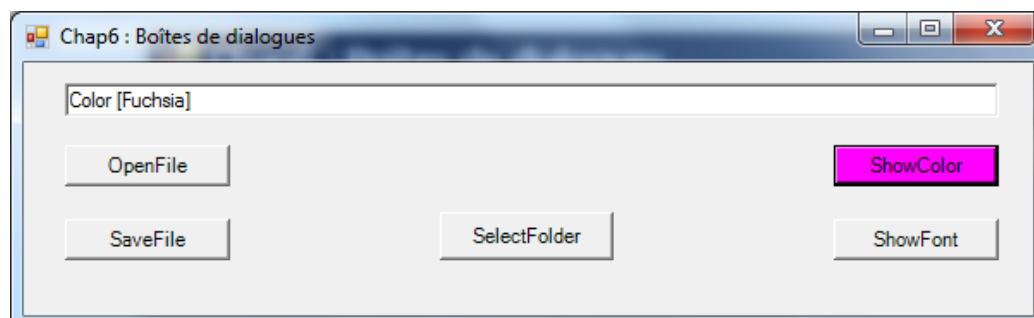


#### 6.9.5.1. UTILISATION DU RÉSULTATS

La valeur sélectionnée par la boîte de dialogue couleur se trouve dans la propriété **Color** du ColorDialog.

```
Private Sub CmdColor_Click(ByVal sender As System.Object,  
ByVal e As System.EventArgs) Handles CmdColor.Click  
    If ColorDialog1.ShowDialog() = DialogResult.OK Then  
        CmdColor.BackColor = ColorDialog1.Color  
        TextBox1.Text = ColorDialog1.Color.ToString  
    End If  
End Sub
```

Le code ci-dessus modifie la couleur du bouton utilisé à partir de la boîte de dialogue Couleurs et affiche la couleur dans le TextBox. (Utilisation de ToString).



### 6.9.5.2. CONFIGURATION DE LA BOITE DE DIALOGUE COULEUR

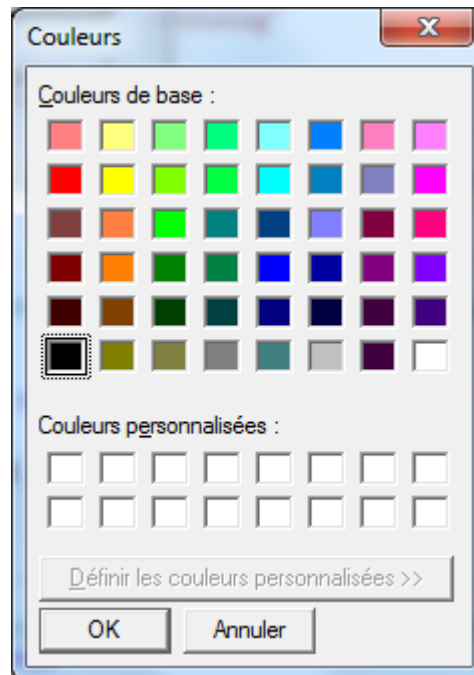
Les propriétés comme **AllowFullOpen** ou **SolidColorOnly** permettent de limiter ou d'autoriser certaine action de l'utilisateur.

Par exemple avec :

```
ColorDialog1.SolidColorOnly = True
```

```
ColorDialog1.AllowFullOpen = False
```

On obtient :



On constate que cela rend inactif (grisé) le bouton "Définir les couleurs personnalisées".

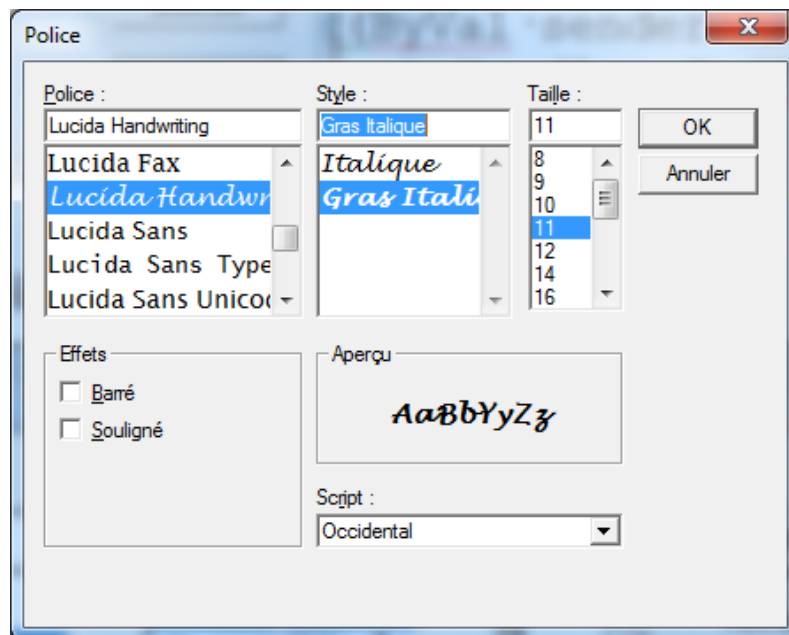
Remarque l'action de SolidColorOnly ne semble pas avoir d'effet.

### 6.9.6. LE CONTRÔLE FONTDIALOG

Si dans la procédure événementielle d'un bouton de commande on ajoute le code suivant, on obtient :

```
Private Sub CmdFont_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CmdFont.Click
    FontDialog1.ShowDialog()
End Sub
```

On obtient après sélection par l'utilisateur:

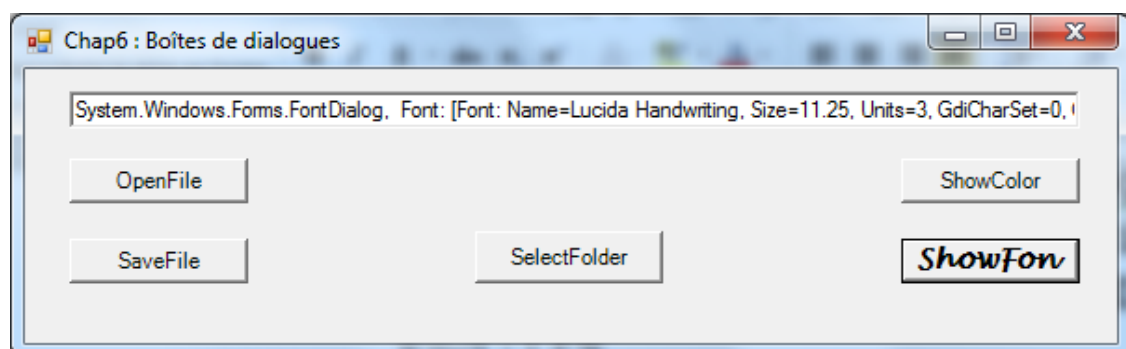


#### 6.9.6.1. UTILISATION DU RÉSULTATS

La valeur sélectionnée par la boîte de dialogue police se trouve dans la propriété **Font** du FontDialog.

Voici par exemple la modification de la police du bouton de commande utilisé pour appeler la boîte de dialogue police. Notez le test, permettant de vérifier si l'utilisateur a cliqué sur OK.

```
If FontDialog1.ShowDialog() = DialogResult.OK Then
    CmdFont.Font = FontDialog1.Font
    TextBox1.Text = FontDialog1.ToString
End If
```



## 6.10. LES BOÎTES D'IMPRESSION

Au niveau du VB 2010 les boîtes de dialogue pour l'impression sont classées dans la boîte à outils dans la section Impression.

Pour la gestion de l'impression d'un document, il est possible d'utiliser les contrôles **PrintDialog**, **PrintPreviewDialog**, **PageSetupDialog** en combinaison avec le composant **PrintDocument**.

Impression		Contrôle	Boîte de dialogue affichée
	Pointeur		
	PageSetupDialog	PageSetupDialog	Mise en page
	PrintDialog	PrintDialog	Impression
	PrintDocument	PrintDocument	aucune
	PrintPreviewControl	PrintPreviewControl	aucune
	PrintPreviewDialog	PrintPreviewDialog	Aperçu avant impression

**PrintPreviewControl**, représente la partie aperçu brute de l'aperçu avant impression à partir d'une application Windows Forms, sans boîtes de dialogue ni boutons.

### 6.10.1. PRINTDOCUMENT, VUE D'ENSEMBLE

## Prise en charge de l'impression dans les Windows Forms



[Envoyer des commentaires](#)

L'impression dans les Windows Forms revient essentiellement à utiliser le composant **PrintDocument Component (Windows Forms)** pour permettre à l'utilisateur d'imprimer, et le contrôle **PrintPreviewDialog Control (Windows Forms)** et les composants **PrintDialog Component (Windows Forms)** et **PageSetupDialog Component (Windows Forms)** pour fournir une interface graphique familière aux utilisateurs habitués au système d'exploitation Windows.

En règle générale, vous créez une nouvelle instance du composant **PrintDocument**, vous définissez les propriétés qui décrivent les éléments à imprimer à l'aide des classes **PrinterSettings** et **PageSettings**, puis vous appelez la méthode **Print** pour procéder à l'impression du document.

Au cours d'une impression initialisée dans une application Windows, le composant **PrintDocument** affiche une boîte de dialogue d'annulation d'impression afin d'avertir les utilisateurs de l'impression en cours et de leur permettre d'annuler le travail d'impression.

## 6.10.2. PRINTDOCUMENT, AIDE

### PrintDocument Component (Windows Forms)



[Envoyer des commentaires](#)

Le composant **PrintDocument** Windows Forms permet de définir les propriétés spécifiant ce qui doit être imprimé, puis d'imprimer le document dans des applications Windows. Il peut être utilisé avec le composant [PrintDialog](#) afin de contrôler tous les aspects de l'impression d'un document.

#### Dans cette section

[PrintDocument Component Overview \(Windows Forms\)](#)

Présente les concepts généraux attachés au composant **PrintDocument**, lequel vous permet de définir des propriétés décrivant ce qui doit être imprimé et d'effectuer l'impression dans une application Windows.

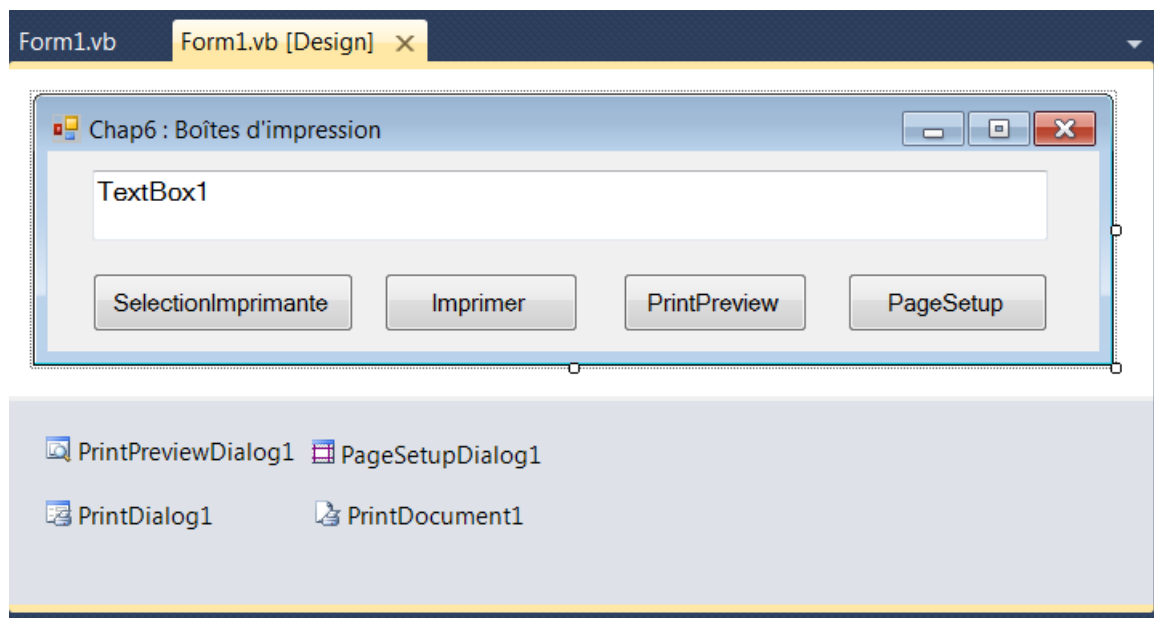
#### Référence

[PrintDocument](#)

Fournit des informations de référence sur la classe et ses membres.

## 6.10.3. APPLICATION POUR LES EXEMPLES

Elle comporte 4 boutons et les 4 contrôles.



#### 6.10.4. UTILISATION DU COMPOSANT PRINTDOCUMENT

Le composant **PrintDocument** étant placé dans le formulaire, il faut utiliser les listes déroulantes de la page de code pour ajouter un événement **PrintPage**.

Le paramètre **e** de l'événement va permettre de définir un objet graphique et de lui appliquer la méthode **DrawString** afin d'obtenir un texte à imprimer.

```
Private Sub PrintDocument1_PrintPage(ByVal sender As
Object, ByVal e As
System.Drawing.Printing.PrintPageEventArgs) Handles
PrintDocument1.PrintPage
    e.Graphics.DrawString(TextBox1.text,
        New Font("Arial", 40, FontStyle.Bold),
        Brushes.Black, 150, 125)
End Sub
```

Cette action a préparé un document avec le texte contenu dans le TextBox1. A partir de là, il est possible d'utiliser les autres boîtes dialogues pour préparer l'impression.

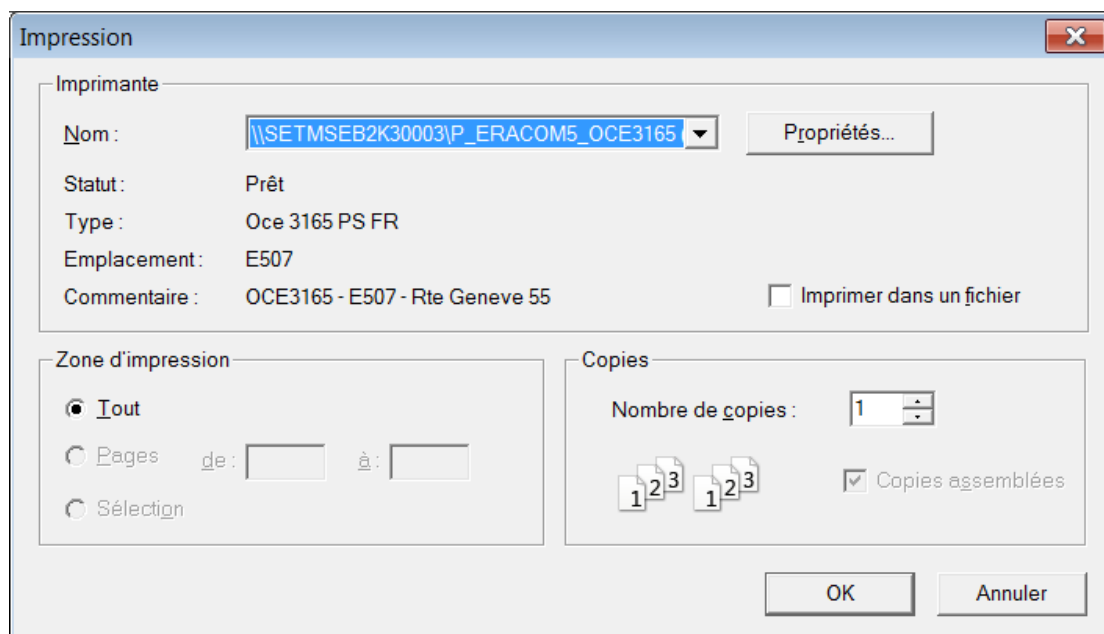
#### 6.10.5. LE CONTRÔLE PRINTDIALOG

Avant d'appeler la méthode **ShowDialog**, il est nécessaire d'initialiser la propriété **Document** du Contrôle **PrintDialog**.

Voici le code minimal à placer dans la procédure événementielle d'un bouton de commande :

```
Private Sub CmdPrint_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CmdPrint.Click
    PrintDialog1.Document = PrintDocument1
    PrintDialog1.ShowDialog()
End Sub
```

On obtient:





### 6.10.5.1. LA PROPRIÉTÉ PRINTERSETTINGS

La propriété **PrinterSettings** obtient ou définit les **PrinterSettings** que la boîte de dialogue modifie.

### 6.10.6. LE CONTRÔLE PRINTPREVIEWDIALOG

Permet d'obtenir l'aperçu avant impression du document spécifié par **PrintDocument**.

Avant d'appeler la méthode **ShowDialog**, il est nécessaire d'initialiser la propriété **Document** du Contrôle **PrintPreviewDialog**.

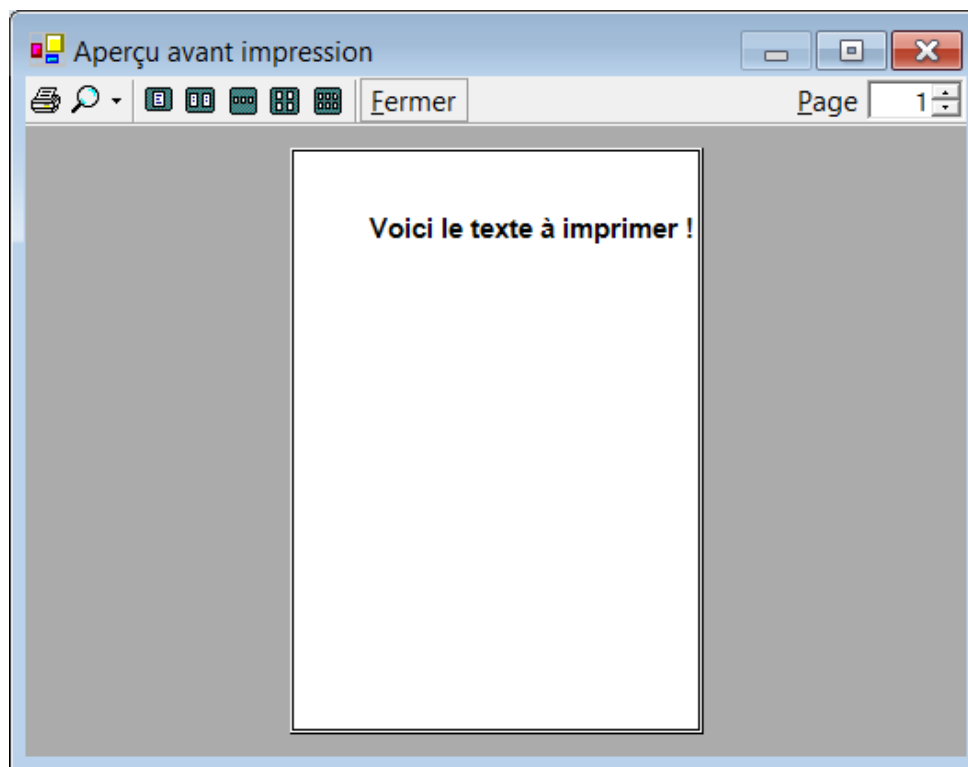
Voici le code minimal à placer dans la procédure événementielle d'un bouton de commande :

```
Private Sub CmdPreview_Click(ByVal sender As System.Object,  
ByVal e As System.EventArgs) Handles CmdPreview.Click
```

```
    PrintPreviewDialog1.Document = PrintDocument1  
    PrintPreviewDialog1.ShowDialog()
```

```
End Sub
```

On obtient:



### 6.10.7. LE CONTRÔLE PAGESETUPDIALOG

Permet d'ouvrir la boîte de dialogue de mise en page, pour agir sur le document spécifié par PrintDocument.

Avant d'appeler la méthode ShowDialog, il est nécessaire d'initialiser la propriété **Document** du Contrôle **PageSetupDialog**.

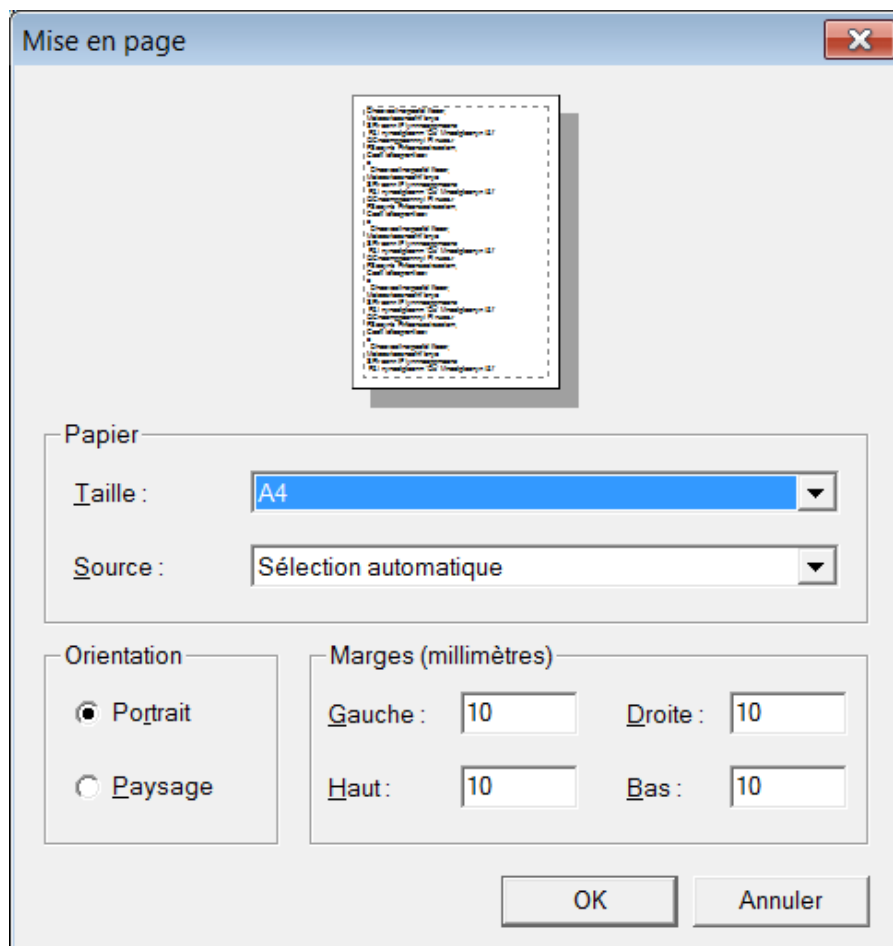
Voici le code minimal à placer dans la procédure événementielle d'un bouton de commande :

```
Private Sub CmdPageSetup_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
CmdPageSetup.Click

    PageSetupDialog1.Document = PrintDocument1
    PageSetupDialog1.ShowDialog()

End Sub
```

On obtient:



#### 6.10.7.1. LA PROPRIÉTÉ PAGESETTINGS

La propriété **PageSettings** obtient ou définit une valeur indiquant les paramètres de page à modifier.

### 6.10.8. IMPRESSION DU DOCUMENT

Pour imprimer le document il faut utiliser la méthode **Print** du composant **PrintDocument**.

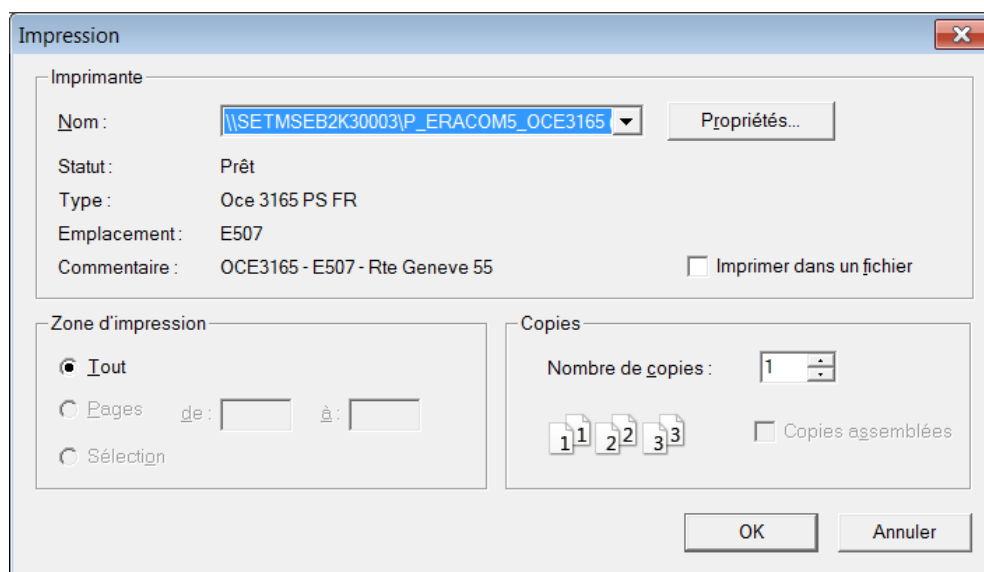
Voici un exemple avec l'appel de la sélection de l'imprimante :

```
Private Sub CmdDoPrint_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CmdDoPrint.Click

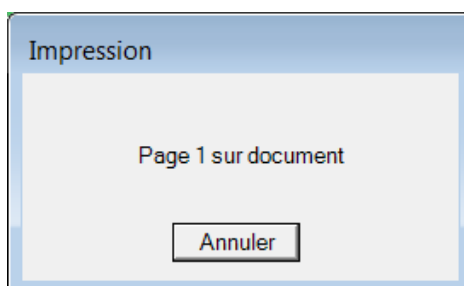
    PrintDialog1.Document = PrintDocument1
    If PrintDialog1.ShowDialog() = DialogResult.OK Then
        'Impression du document
        PrintDocument1.Print()
    Else
        MessageBox.Show _
            ("Il faut sélectionner une imprimante", _
            "Chap 6 : Impression")
    End If
End Sub
```

Remarque : les actions des dialogues de configurations agissent sur les arguments du composant PrintDocument, par le fait qu'un lien a été créé entre la propriété **Document** des contrôles et le composant PrintDocument.

Lors de l'exécution on obtient la boîte de dialogue impression et on doit trouver le document imprimé.



Une confirmation apparaît brièvement :



## **6.11. CONCLUSION**

Ce chapitre a regroupé une bonne partie des éléments permettant l'interaction avec l'utilisateur. Au niveau des possibilités du graphique ce n'est qu'une base. Pour la réalisation d'applications graphiques complexes une des solutions consiste à acheter des composants réalisés par des entreprises spécialisées.

## **6.12. HISTORIQUE**

### **6.12.1. VERSION 1.0 JUIN 2016**

Création du chapitre 6 en "traduisant" la version 3.3 du chapitre 6 du VB 2010.  
A jour MessageBox et clavier.

### **6.12.2. VERSION 1.0 B JUIN 2016**

A jour gestion de la souris.