Interactions clavier et souris

Dans un environnement graphique tel que celui fourni par les applications WPF, il est essentiel d'être capable de réagir aux interactions entre l'utilisateur et le logiciel afin de fournir une expérience fluide et cohérente. Ces interactions sont aujourd'hui presque uniquement réalisées à l'aide de deux périphériques : le clavier et la souris. WPF fournit différents événements déclenchés lorsque l'utilisateur effectue une action sur l'un de ces périphériques. Ces événements sont générés par le type `System.Windows.UIElement`, qui est un ancêtre commun de la quasi-totalité des contrôles WPF.

1. Événements clavier

Avec WPF, deux événements principaux sont déclenchés lorsque l'utilisateur appuie sur une touche de son clavier : `KeyDown` et `KeyUp`. Ils correspondent respectivement à l'appui et au relâchement de la touche et sont déclenchés dans ce même ordre.

Le délégué associé à ces deux événements possède la définition suivante :

```
public delegate void KeyEventHandler(object sender, KeyEventArgs e);
```

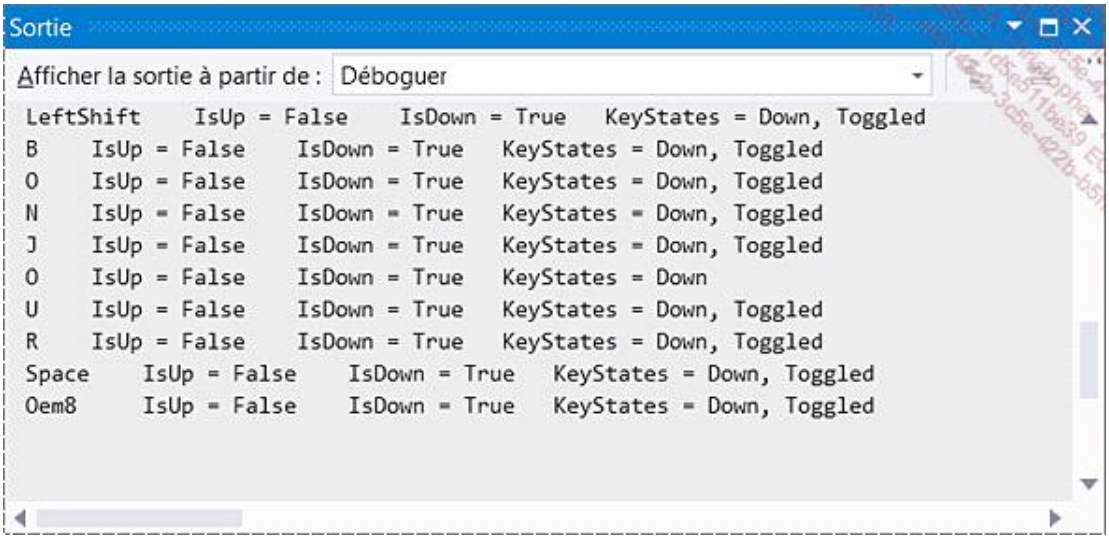
Les données transmises dans le paramètre de type `KeyEventArgs` permettent de connaître la touche physique du clavier impliquée dans l'événement déclenché, ainsi que son état. L'exemple de code suivant affiche dans la fenêtre **Sortie** de Visual Studio le nom de la touche manipulée ainsi que son état au travers des propriétés `IsUp`, `IsDown` et `KeyStates` de l'objet `KeyEventArgs`.

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();

        KeyDown += Window_ManipulationToucheClavier;
    }

    private void Window_ManipulationToucheClavier(object sender,
KeyEventArgs e)
    {
        Console.WriteLine("{0} \t IsUp = {1} \t IsDown = {2} \t
KeyStates = {3}", e.Key, e.IsUp, e.IsDown, e.KeyStates);
    }
}
```

Ainsi, lorsque la fenêtre de l'application est active, la saisie du texte `Bonjour !` affiche le résultat suivant dans la fenêtre **Sortie**.



La touche `LeftShift` est utilisée pour écrire la première lettre en majuscule. La touche `Oem8` que l'on trouve en dernière position correspond au point d'exclamation.

Il est possible d'obtenir le texte saisi caractère par caractère en utilisant l'événement `TextInput`. Le délégué qui lui est associé est le suivant :

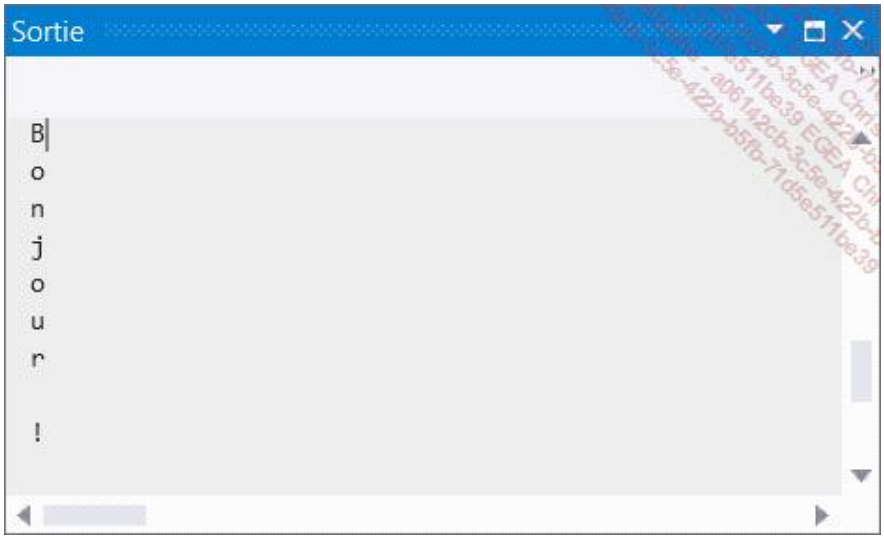
```
public delegate void TextCompositionEventHandler(object sender,
TextCompositionEventArgs e);
```

Le paramètre de type `TextCompositionEventArgs` possède une propriété `Text` qui contient le texte associé à un appui sur une touche, si celle-ci génère du texte.

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();

        TextInput += MainWindow_TextInput;
    }

    void MainWindow_TextInput(object sender,
TextCompositionEventArgs e)
    {
        Console.WriteLine(e.Text);
    }
}
```



2. Événements souris

La souris génère beaucoup plus d'événements différents que le clavier. Il est en effet possible d'appuyer sur ses boutons, de faire rouler la molette ou de la déplacer. Une dizaine d'événements dédiés à la gestion des interactions par la souris sont ainsi disponibles sur chaque contrôle. Leur nom commence systématiquement par `Mouse`.

MouseDown et MouseUp

Ces deux événements sont déclenchés lorsqu'un bouton de la souris est appuyé ou relâché. Le bouton concerné peut être le gauche, le droit, un bouton placé au centre qui est généralement la molette, ou encore un bouton étendu, c'est-à-dire qu'il ne correspond pas fonctionnellement à une souris classique. Les boutons étendus, lorsqu'ils sont présents, sont généralement placés sur le côté.

Le paramètre `MouseButtonEventArgs` passé aux gestionnaires de ces événements possède une propriété `ChangedButton` indiquant le bouton qui a été manipulé. Il fournit également une propriété `ButtonState` qui indique si le bouton est pressé ou non, et une propriété `ClickCount` qui permet quant à elle de savoir si plusieurs clics ont été effectués. Si vous souhaitez implémenter une fonctionnalité de gestion du triple clic, c'est vers cette dernière propriété que vous devez vous diriger.

MouseLeftButtonDown et MouseLeftButtonUp

Ces événements sont déclenchés spécifiquement lorsque le bouton gauche de la souris est manipulé. Leur fonctionnement est parfaitement identique à celui de `MouseDown` et `MouseUp`.

MouseRightButtonDown et MouseRightButtonUp

Ces événements sont déclenchés spécifiquement lorsque le bouton droit de la souris est manipulé. Leur fonctionnement est parfaitement identique à celui de `MouseDown` et `MouseUp`.

MouseMove

Cet événement est déclenché à chaque mouvement du curseur de la souris au-dessus du contrôle pour lequel l'événement est géré. Les données fournies aux gestionnaires d'événements ne concernent pas la position de la souris. Pour connaître les coordonnées du curseur, il convient d'utiliser la méthode `GetPosition` de la classe statique `Mouse`. Cette méthode accepte en paramètre un objet implémentant l'interface `IInputElement` :

n'importe quel contrôle WPF peut être fourni à cette méthode.

Dans le fichier `.xaml` :

```
<TextBox x:Name="txtNom" />
```

Dans le fichier `.xaml.cs` :

```
double positionHorizontale = Mouse.GetPosition(txtNom).X
double positionVerticale = Mouse.GetPosition(txtNom).Y
```

MouseEnter et MouseLeave

Ces événements sont déclenchés lorsque le curseur entre ou sort d'un contrôle. Pour ces deux événements, le gestionnaire accepte un paramètre de type `MouseEventArgs`. Celui-ci permet d'analyser l'état courant de la souris grâce à ses propriétés `LeftButton`, `RightButton`, `MiddleButton`, `X1Button` et `X2Button`. Ces propriétés fournissent le statut des différents boutons du périphérique. Ces événements peuvent notamment être particulièrement intéressants si vous souhaitez implémenter vous-même la totalité du code nécessaire à la mise en place du glisser-déposer.

3. Glisser-déposer

Lorsque l'on utilise WPF, la gestion du glisser-déposer est simplifiée par différents éléments :

- Chaque contrôle peut être une source ou une cible de glisser-déposer.
- La méthode statique `DoDragDrop` de la classe `DragDrop` implémente le stockage temporaire des données déplacées et s'occupe également de la gestion des effets visuels associée à l'opération.
- Plusieurs événements permettent de fournir des traitements et des retours visuels à l'utilisateur en fonction de l'évolution de l'opération.

La mise en œuvre de cette action sera effectuée au travers d'un exemple simple : le déplacement d'un disque rouge sur un rectangle vert colore le rectangle en rouge.

- Pour commencer, créez un nouveau projet d'application WPF nommé `GlisserDeposer`.
- Modifiez le code du fichier `MainWindow.xaml` de manière à positionner un contrôle `Ellipse` et un contrôle `Rectangle`.

```
<Window x:Class=" GlisserDeposer.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="468.4" Width="553.8">
    <Canvas>
        <Ellipse Canvas.Left="50" Canvas.Top="50" Height="100"
                 Width="100" Fill="#FF0000" />

        <Rectangle Height="100" Width="100" Canvas.Left="400"
```

```
                    Canvas.Top="280" Stroke="#000000"
                    Fill="#00FF00" StrokeThickness="3"
    </Canvas>
</Window>
```

La structure visuelle est en place. À partir de là, trois éléments doivent être pris en compte :

- Le contrôle `Ellipse` est la source du glisser-déposer.
- Le `Rectangle` est la cible de cette action.
- Une donnée est transmise de l'`Ellipse` vers le `Rectangle` : sa couleur.

Commençons par autoriser le `Rectangle` à être la cible d'une opération de glisser-déposer.

➔ Ajoutez l'attribut `AllowDrop` au `Rectangle` et valorisez-le à `True`.

```
<Rectangle .... AllowDrop="True" />
```

Il est maintenant nécessaire de gérer le déclenchement de l'action. Pour cela, il nous faut repérer un mouvement de souris sur le contrôle `Ellipse` pendant lequel le bouton gauche de la souris est appuyé. L'événement `MouseMove` est idéal pour gérer tout cela.

Abonnez-vous à l'événement `MouseMove` du contrôle `Ellipse` et fournissez-lui le gestionnaire suivant :

```
private void Ellipse_MouseMove(object sender, MouseEventArgs e)
{
    if (e.LeftButton == MouseButtonState.Pressed)
    {
        var ellipse = (Ellipse)sender;
        DragDrop.DoDragDrop(ellipse, ellipse.Fill.ToString(),
DragDropEffects.Copy);
    }
}
```

Ce gestionnaire déclenche l'opération de glisser sur le contrôle `Ellipse`, stocke la valeur de la couleur de remplissage de ce contrôle de manière à pouvoir la réutiliser plus tard, et enfin applique l'effet visuel "Copie". Cet effet modifie le curseur lorsque l'on atteint une cible telle que le contrôle `Rectangle`.

Puisque nous sommes en mesure de faire glisser notre `Ellipse`, il reste maintenant à gérer la partie "déposer" de l'opération.

Pour cela, abonnez-vous à l'événement `Drop` du `Rectangle` et créez son gestionnaire d'événement.

```
private void Rectangle_Drop(object sender, DragEventArgs e)
{
    Rectangle rectangle = sender as Rectangle;
    if (rectangle != null)
    {

       //On vérifie l'existence d'une donnée stockée
```

```
        //pour le glisser-déposer

        if (e.Data.GetDataPresent(DataFormats.StringFormat))
        {

            //On récupère la donnée. C'est ici une chaîne de
            //caractères ayant pour valeur "#FFFF0000"

            string dataString =
(string)e.Data.GetData(DataFormats.StringFormat);

            //On utilise un objet BrushConverter pour transformer
            //cette chaîne de caractères en Brush applicable
            //à la propriété Fill du rectangle

            BrushConverter converter = new BrushConverter();
            if (converter.IsValid(dataString))
            {
                Brush newFill =
(Brush)converter.ConvertFromString(dataString);
                rectangle.Fill = newFill;
            }
        }
    }
}
```

Lors de l'événement `Drop`, on récupère la donnée stockée précédemment par la méthode `DoDragDrop`. On transforme cette donnée de manière à pouvoir l'utiliser : ici, on utilise un `BrushConverter` dont le but est la conversion d'une chaîne de caractères en objet `Brush` (pinceau) utilisable pour colorer un élément. On applique enfin la valeur obtenue à la propriété `Fill` du contrôle `Rectangle`.

Lancez l'application et effectuez l'opération de glisser-déposer du cercle vers le rectangle : le rectangle change de couleur !

Les événements `DragEnter` et `DragLeave` permettent de rajouter du code entre le démarrage de l'opération et sa fin. `DragEnter` est déclenché lorsqu'un contrôle est glissé au-dessus d'un contrôle autorisant cette opération, sans le lâcher. `DragLeave` est exécuté lorsque le contrôle est glissé à l'extérieur des limites d'un autre contrôle. Ces deux événements offrent ainsi la capacité de générer, entre autres, une prévisualisation de l'effet d'un glisser-déposer.