WPF : COMMENT FAIRE RAPIDEMENT UN USER CONTROLE

REUTILISABLE

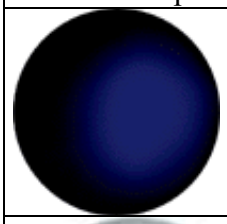
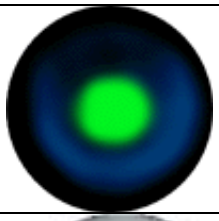
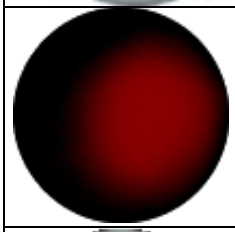
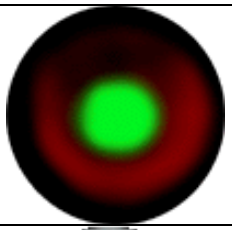
Auteur :Hassan KANDIL
Date de la première publication :12/04/2010
Revue et adaptée le : 02/07/2017

Pour réaliser un contrôle utilisateur en WPF, Les méthodes sont multiples et efficaces. A commencer par le style et les templates du zammel jusqu'à l'écriture en VB, C++ ou C# d'un code qui gère l'affichage de l'objet et son comportement.
Bien sûr nous devons garder en tête avant de commencer l'écriture de l'objet, qu'il y a toujours un objet (ou plusieurs) qui peuvent servir comme ancêtres de notre objet et donc, il vaut mieux se baser sur l'existant pour générer un nouvel objet.
L'exemple que j'ai choisi d'exposer ici, montre un objet d'une utilité moyenne mais avec quelques modifications des images il pourrait s'avérer très utile.
Cet exemple tentera de réaliser un bouton rond avec des aspects « bitmapés ». Il ne prétend pas être le plus ergonomique, mais il pourra être un exemple d'une réalisation très facile.

La première question qui se pose est sur l'utilisation des ressources dans un contrôle utilisateur. Il faut que les ressources soient incorporées, or WPF incorpore automatiquement les ressources, à moins que l'on veuille les inclure dans un dictionnaire de ressources, et dans ce cas, il faut assurer l'ajout du dictionnaire au projet (merging), ce qui me semble non adapté à la situation, surtout qu'un objet de cette nature ne doit pas traîner beaucoup de ressources, pour des raisons d'efficacité

Bouton Rond.

Notre Contrôle utilisateur est un bouton rond cliquable . Il a quatre aspects différents :

En état de repos	Pendant le click	Nom è utiliser dans la clé
		bleu
		elastique
		rouge
		vrai

.

ce qui représente 8 fichiers d'images différentes,

Le traitement regroupe les images 2 par 2 en utilisant un préfixe suivi d'une description identique Origine (repos) – Destination(cliqué) . Ceci est utile dans la constitution de la clé de l'image dans le fichier XAML.

Comme on a dit, les objets graphiques seront inclus dans le contrôle utilisateur D'où la définition du <UserContol> qui contiendra cette partie de ressources.

```
    <UserControl.Resources>

        <ImageBrush x:Key="rougeOrigine" ImageSource ="Images/rougeOrigine.bmp"/>
        <ImageBrush x:Key="rougeDestination" ImageSource ="Images/rougeDest.bmp"/>
        <ImageBrush x:Key="bleuOrigine" ImageSource ="Images/bleuOrigine.bmp"/>
        <ImageBrush x:Key="bleuDestination" ImageSource ="Images/bleuDest.bmp"/>
        <ImageBrush x:Key="vraiOrigine" ImageSource ="Images/vraiOrigine.bmp"/>
        <ImageBrush x:Key="vraiDestination" ImageSource ="Images/vraiDest.bmp"/>
        <ImageBrush x:Key="elastiqueOrigine" ImageSource ="Images/elastiqueOrigine.bmp"/>
        <ImageBrush x:Key="elastiqueDestination" ImageSource ="Images/elastiqueDest.bmp"/>
```

Ceci suppose que les huit fichiers sont placés dans un dossier Images au sein du projet WPF.

Les noms des fichiers importent peu, seuls les clés doivent être structurées d'une manière à pouvoir les reconstituer facilement à partir de la couleur. L'idée est de considérer la clé comme étant couleur+chaîne constante (Origine ou Destination)

Le bouton devrait contenir une template qui permettra de le dessiner et de définir la partie qui nous intéresse de son comportement : Changement d'aspect (image) quand il est cliqué.

Commençons déjà par la création du template et donnons un nom Clé à cette création et bien sûr un target type pour l'appliquer à notre futur objet

```
        <ControlTemplate x:Key="BoutonRondTemplate" TargetType="Button">
Dessinons un aspect rond
                <Ellipse x:Name="BoutonEllipse">
Nous ne mettons pas de Height ni Width dans l'ellipse, mais nous savons que notre bouton devrait avoir
la même valeur pour ces deux propriétés. Nous pouvons lier la taille de l'ellipse à la taille de
l"objet en ajoutant
Height="{Binding ElementName=BRUser,Path=Height}" Width="{Binding ElementName=BRUser,Path=Width}">.
BRUser étant le nom du bouton que l'on utilisera dans le UserControl (ce nom sera interne à l'objet).

Pour le moment remplissons notre Ellipse(circulaire) par l'image au repos, laquelle est une variable
objet que l'on nommera FileOrigine. Cet objet fera partie du code behind et assurera la transmission de
l'image appropriée.

                    <Ellipse.Fill>
                        <ImageBrush ImageSource="{Binding FileOrigine,RelativeSource={RelativeSource
AncestorType=UserControl}, Mode=OneWay}"/>
                    </Ellipse.Fill>
                </Ellipse>

L'objectif de la RelativeSource={RelativeSource AncestorType=UserControl est tout simplement de
signaler l'élément origine de la data c'est à dire notre UserControl. Ceci nous évitera de définir le
DataContext plus tard pour pouvoir utiliser notre objet.
```

Un event nous intéresse particulièrement :la souris cliquée. On peut aussi s'intéresser à d'autres events (le mouse over par exemple).

2 propriétés : IsPressed (le bouton est cliqué ou validé par le clavier) et IsMouseOver (la souris survole l'objet).

La valeur « True « de chacune de ces propriétés doit déclencher des actions qui seront accomplies à travers des triggers et des setter property-value. Le trigger indique l'event, le setter avec TargetName indique la propriété et sa valeur. Ceci se fait de la manière suivante.

```
            <ControlTemplate.Triggers>
                <Trigger Property="IsPressed" Value="True">
```

```
                    <Setter TargetName="BoutonEllipse" Property="Fill" >
                        <Setter.Value>
                            <ImageBrush ImageSource="{Binding
FileDestination,RelativeSource={RelativeSource AncestorType=UserControl}, Mode=OneWay}"/>
                        </Setter.Value>
                    </Setter>
                </Trigger>
```

Pour le mouse over nous allons nous contenter de jouer sur la couleur du stroke(le bord de l'ellipse) en le rendant rouge. Rien n'empêche d'ajouter une troisième image pour cette situation ou de faire des transformations (réduction/agrandissement ou autres).

```
                <Trigger Property="IsMouseOver" Value="True">
                    <Setter TargetName="BoutonEllipse" Property="Stroke" Value="red"/>
                </Trigger>

            </ControlTemplate.Triggers>
```

Les deux événements sont traités, nous devons fermer les balises

```
        </ControlTemplate>
    </UserControl.Resources>
```

Pour définir l'objet du Template on va utiliser le layout StackPanel et ne pas oublier la taille à donner au bouton qui sera le même attribué à l'ellipse.

```
    <StackPanel>
        <Button x:Name="BRUser" Height="50" Width="50" Template="{StaticResource BoutonRondTemplate}"
Click="Button_Click" Margin="0,0,0,0" />
    </StackPanel>
```

Le click est l'event qui sera traité dans notre programme qui exploitera plus tard le bouton rond.

Note : En fait rien ne nous empêche de le faire elliptique en donnant à Width une valeur différente de Height.

Le bouton est dessiné Le fichier XAML est le suivant:

```
<UserControl x:Class="AsysButton.BoutonRond"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:AsysButton"
             mc:Ignorable="d"
             d:DesignHeight="100" d:DesignWidth="100">
    <UserControl.Resources>
        <ImageBrush x:Key="rougeOrigine" ImageSource ="Images/rougeOrigine.bmp"/>
        <ImageBrush x:Key="rougeDestination" ImageSource ="Images/rougeDest.bmp"/>
        <ImageBrush x:Key="bleuOrigine" ImageSource ="Images/bleuOrigine.bmp"/>
        <ImageBrush x:Key="bleuDestination" ImageSource ="Images/bleuDest.bmp"/>
        <ImageBrush x:Key="vraiOrigine" ImageSource ="Images/vraiOrigine.bmp"/>
        <ImageBrush x:Key="vraiDestination" ImageSource ="Images/vraiDest.bmp"/>
        <ImageBrush x:Key="elastiqueOrigine" ImageSource ="Images/elastiqueOrigine.bmp"/>
        <ImageBrush x:Key="elastiqueDestination" ImageSource ="Images/elastiqueDest.bmp"/>

        <ControlTemplate x:Key="BoutonRondTemplate" TargetType="Button">

            <Grid >
                <Ellipse x:Name="BoutonEllipse">
                    <Ellipse.Fill>
                        <ImageBrush ImageSource="{Binding FileOrigine,RelativeSource={RelativeSource
AncestorType=UserControl}, Mode=OneWay}"/>
                    </Ellipse.Fill>
                </Ellipse>
            </Grid>
            <ControlTemplate.Triggers>
                <Trigger Property="IsPressed" Value="True">
                    <Setter TargetName="BoutonEllipse" Property="Fill" >
                        <Setter.Value>
```

```
                            <ImageBrush ImageSource="{Binding
FileDestination,RelativeSource={RelativeSource AncestorType=UserControl}, Mode=OneWay}"/>
                        </Setter.Value>
                    </Setter>
                </Trigger>
                <Trigger Property="IsMouseOver" Value="True">
                    <Setter TargetName=" BoutonEllipse" Property="Stroke" Value="red"/>
                </Trigger>
            </ControlTemplate.Triggers>
        </ControlTemplate>
    </UserControl.Resources>
    <StackPanel>
        <Button x:Name="BRUser" Height="50" Width="50" Template="{StaticResource BoutonRondTemplate}"
Click="Button_Click" Margin="0,0,0,0" />
    </StackPanel>
</UserControl>
```

La gestion des Méthodes permettant cette liaison créée avec FileOrigine et FileDestination doit être assurée dans le code behind.

L'entête de ce fichier contiendra les using suivants (importations) :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.ComponentModel;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
```

Remarquons que ces deux éléments doivent retourner des ImageSource … Si elles n'étaient pas capables de l'assurer, on aurait fait appel à un converter qui transformera la ressource en imageSource ou BitmapImage qui sera retournée au Binding. Ceci aurait exigé d'ajouter dans le binding l'élément Converter={staticResource nomdéfinidanslesUserResources}.

Il y aura lieu dans ce cas à ajouter une classe dérivée de IValueConverter avec les deux méthodes Convert et ConvertBack.

Le code Behind doit contenir tout simplement Une dependency property (ce n'est pas la seule façon de faire une INotifyPropertyChanged aurait réalisé la tâche).

Mais Le changement de l'aspect du bouton sera moins élégant que de le faire à travers une dependency property qui peut figurer directement dans le XAML de la fenêtre owner du usercontrol.

La définition des deux attributs qui contiendront la ressource est le premier pas à réaliser dans le code behind

```
        #region Définitions
        private ImageSource fileOrigine;
        private ImageSource fileDestination;
        #endregion
```

Il faut aussi définir une propriété nommée Couleur (Aspect aurait mieux correspondu à notre cas) qui reçoit les valeurs : bleu/rouge/elastique/vrai et qui définit l'aspect du bouton.

Une approche intuitive nous pousse à définir les valeurs possibles (gérées) de la propriété Couleur. Pour ce faire nous utilisons une list<string> qui contiendra toutes les valeurs autorisées. Dans notre cas, quatre valeurs (bleu/rouge/elastique/vrai) sont à stocker dans la liste. Cette liste est unique, elle ne doit pas être régénérée à chaque instanciation du contrôle. C'est pour cette raison qu'elle est définie en static. En plus elle pourrait être exploitée sans instanciation.

```
public static List<string> Couleurs = new List<string>();
```

Pour remplir cette liste, on pourrait procéder de plusieurs façons, une des plus simples est de le faire dans le constructeur et de s'assurer que cette opération n'est pas reproduite à chaque instanciation.

```
#region Constructer
public BoutonRond()
{
    if (Couleurs.Count() == 0)
   {
      //dans un ordre croissant pour utiliser System.Array.BinarySearch(tab, Couleur)
        Couleurs.Add("bleu");
        Couleurs.Add("elastique");
        Couleurs.Add("rouge");
        Couleurs.Add("vrai");
    }
    InitializeComponent();
}
#endregion
```

Les éléments du DataBinding sont au nombre de deux FileOrigine et FileDestination
Ces deux éléments doivent tout simplement reconnaître le choix fait par la propriété dépendante Couleur et associer le fichier Bitmap correspondant.

Pour cela on peut utiliser indifféremment Ressources(qui pourrait être utilisée aussi avec un dictionnaire de resources) et FindRessource (pareil).
L'approche dans ce cas utilisera Application.Current.FindResource (ou resources[]) et procédera à l'import des ressources soit dans l'application soit dans un dictionnaire partagé.

Or nos ressources se trouvent dans le usercontrol XAML, donc on y accède avec this qui représente dans le code behind le usercontrole
Ces deux méthodes d'accès (resources[] et FindResource() ont besoin de la clé de la ressource recherchée).

Cette clé est Constituée de la Couleur+le mot Origine ou le mot Destination (d'après notre vision algorithmique.)
La construction de l'image peut se faire avec le set ou au moment du get ou même à un autre moment (au moment du changement de la valeur de Couleur)

```
#region DataBinding Elments
public ImageSource FileOrigine
{
    get
    {

        fileOrigine = ((ImageBrush)this.FindResource(Couleur + "Origine")).ImageSource;
        return fileOrigine;
    }
    set
    {
        fileOrigine = value;
        NotifyPropertyChange("FileOrigine");
    }
}
public ImageSource FileDestination
{
    get
    {
        fileDestination = ((ImageBrush)this.Resources[Couleur + "Destination"]).ImageSource;
        return fileDestination;
    }
    set
```

```
            {
                fileDestination = value;
                NotifyPropertyChange("FileDestination");
            }
        }
        #endregion
```

L'event Click que nous avons saisi dans le BRUser de notre contrôle doit être confié à l'Owner du contrôle. Ainsi nous aurons à définir une publication de cet event.

```
        #region Traitement de  l'event
        public event EventHandler OnClick;
        private void ThatWillRaisesEvent()
        {
            OnClick?.Invoke(this, new EventArgs());
        }
        private void Button_Click(object sender, RoutedEventArgs e)
        {
            ThatWillRaisesEvent();
        }
        #endregion
```

La définition de la dependency property est la plus standard qu'elle soit à l'exception de deux lignes qui font le lien entre le changement de la propriété couleur et le changement des FileOrigine et FIleDestination, en plus cette propriété doit avoir des valeurs de type string (bleu/rouge/elastique/vrai). La valeur vrai dans la metadata est la valeur par défaut.
On peut faire un setvalue pour associer l'image à la méthode, ou bien déclarer tout simplement une notification de changement de propriété ou bien évidemment Terminer le travail dans le set de Couleur.

System.Array.BinarySearch(tab, Couleur) permet d'éviter l'utilisation des valeurs hors périmètre. Ce qui explique le manque de gestion des exceptions dans le programme.
Bien sûr il faut ajouter de try .. catch ... mais ce n'est pas le propos de l'actuelle démonstration.

```
        #region DependencyProperty Couleur qui indique l'image
        public static readonly DependencyProperty CouleurProperty =
DependencyProperty.Register("Couleur", typeof(string), typeof(BoutonRond), new PropertyMetadata("vrai",
new PropertyChangedCallback(OnCouleurChanged)));

        public string Couleur
        {
            get
            { return (string)GetValue(CouleurProperty); }
            set
            {
               string[] tab= Couleurs.ToArray();
               if (System.Array.BinarySearch(tab, Couleur)>=0)
                     {
                       SetValue(CouleurProperty, value);
                        FileOrigine = ((ImageBrush)this.Resources[Couleur + "Origine"]).ImageSource;
                 FileDestination = ((ImageBrush)this.Resources[Couleur + "Destination"]).ImageSource;
                     }
            }
        }
        private static void OnCouleurChanged(DependencyObject depObj,
DependencyPropertyChangedEventArgs e)
        {
            BoutonRond bRControl = depObj as BoutonRond;
            bRControl.OnCouleurChanged(e);
        }
        private void OnCouleurChanged(DependencyPropertyChangedEventArgs e)
        {
                if(e.NewValue!=null) Couleur = e.NewValue.ToString();
        }
        #endregion
```

Le binding doit toujours s'accompagner par une notification de changement de propriété classique

```
        #region NotifyPropertyChanged
        public event PropertyChangedEventHandler PropertyChanged;
        private void NotifyPropertyChange(String property)
```

```
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(property));
        }
        #endregion
```

Nous avons remarqué le fait que notre UserControle avait un width et un height prédéfinis. Or ces données doivent être logiquement définies comme dependency property pour les manipuler directement en XAML au moment de la creation et même à travers des triggers avec setter pour avoir de l'animation. Nous nous contentons de les rendre souples d'utilisation sans aller plus loin, en ajoutant deux Méthodes qui (seulement en programmation) permettent de changer les valeurs de taille.

```
        #region Méthodes Extra
        public void SetWidth(double dWidth)
        {
            this.BRUser.Width = dWidth;
            this.Width = this.BRUser.Width;
        }
        public void SetHeight(double dHeight)
        {
            this.BRUser.Height = dHeight;
            this.Height = this.BRUser.Height;
        }
        #endregion
```

remarquez que l'on a utilisé dans nos méthodes le nom interne du bouton défini dans le UserControl XAML.

Testons le bouton rond

Voici une fenêtre de test que l'on peut directement utiliser dans notre programme qui contient un contrôle utilisateur et une fenêtre principale

```
<Window
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:AsysButton "
        mc:Ignorable="d"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <local:BoutonRond x:Name="RB" Couleur = "{Binding
SelectedItem,ElementName=comboCouleursRB,Mode=TwoWay}" HorizontalAlignment="Left" Height="100"
VerticalAlignment="Top" Width="100" Margin="180,45,0,0" Click=”BR_OnClick”/>
        <ComboBox x:Name="comboCouleursRB" SelectedItem="{Binding Path=Couleur, Mode=TwoWay}"
IsReadOnly="True" HorizontalAlignment="Left" Height="30" Margin="210,230,0,0" VerticalAlignment="Top"
Width="170"/>
        <TextBox Name="tbRB" HorizontalAlignment="Left" Height="25" Margin="30,185,0,0"
TextWrapping="Wrap" Text="{Binding Path=Couleur, Mode=TwoWay}" VerticalAlignment="Top" Width="110"/>
        <Button Content="Rouge" HorizontalAlignment="Left" Height="30" Margin="30,230,0,0"
VerticalAlignment="Top" Width="110" Click="Button_Click"/>
    </Grid>
</Window>
```

Code behind

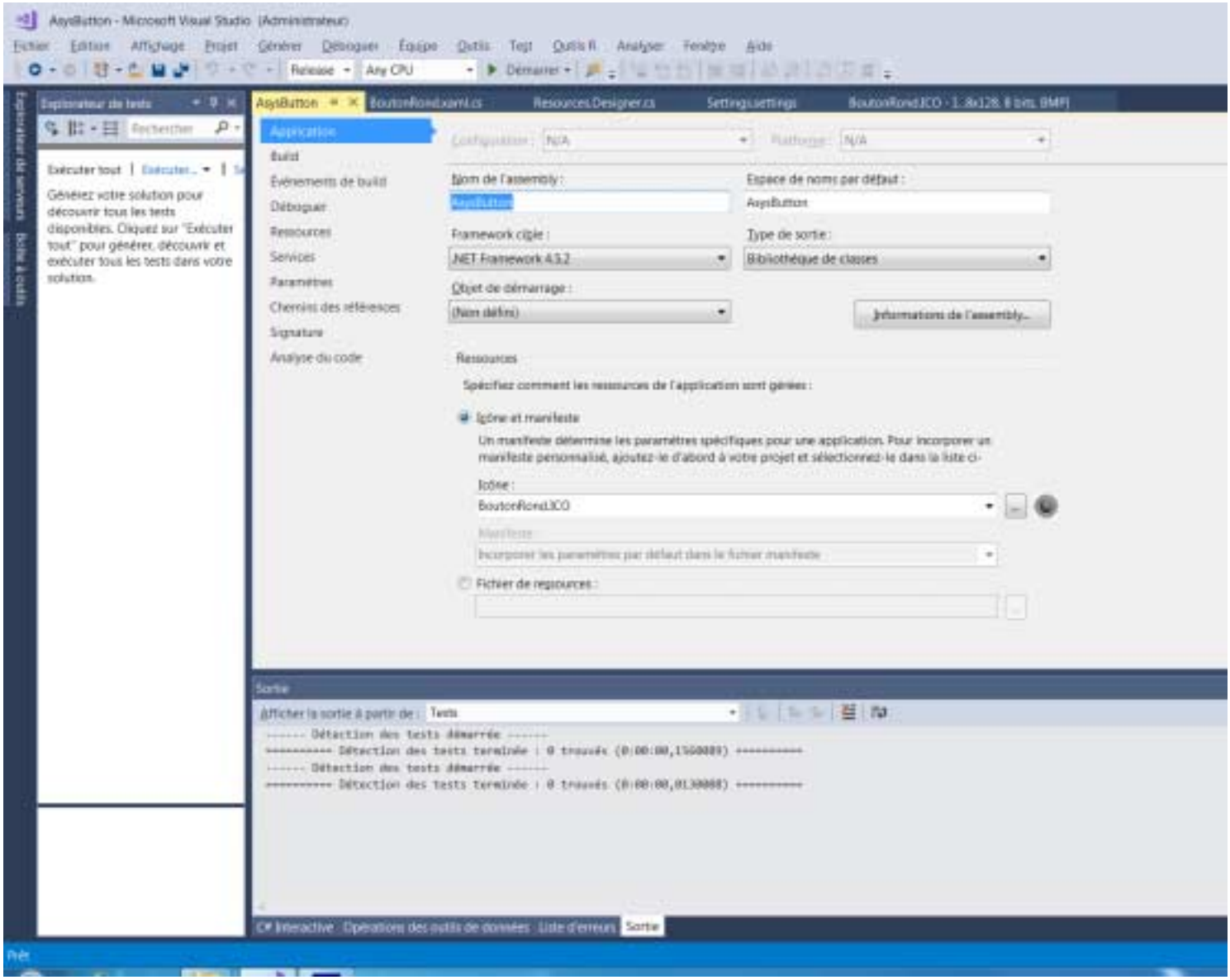
```
namespace AsysButton
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            this.DataContext = new BoutonRond();
//remplir la combobox par les valeurs autorisées
            BoutonRond.Couleurs.ForEach(item => { comboCouleursRB.Items.Add(item); });
        }
        private void BR_OnClick(object sender, RoutedEventArgs e)
       {
```

```
        // code exploitant le bouton rond s'il est cliqué
      }

       private void Button_Click(object sender, RoutedEventArgs e)
       {
             //changer le bouton utilize en rouge
           RB.SetCurrentValue(BoutonRond.CouleurProperty,"rouge");
       }
    }
}
```

Ainsi dans cet exemple on verra facilement les divers aspects du bouton soit avec la combobox, soit en cliquant sur le bouton Rouge.

Pour transformer l'ensemble en UserControle réutilisable, il suffit de supprimer la fenêtre MainWindow et de définir le projet comme étant une DLL. D'un projet WPF classique on passe à un projet DLL.
Pour cela cliquer sur le nom du projet et accéder à ses propriétés et modifier les infos du type de sortie – Choisissez Bibliothèque de classes.



Le résultat après régénération du projet sera une DLL que vous pouvez ajouter à d'autres projets .

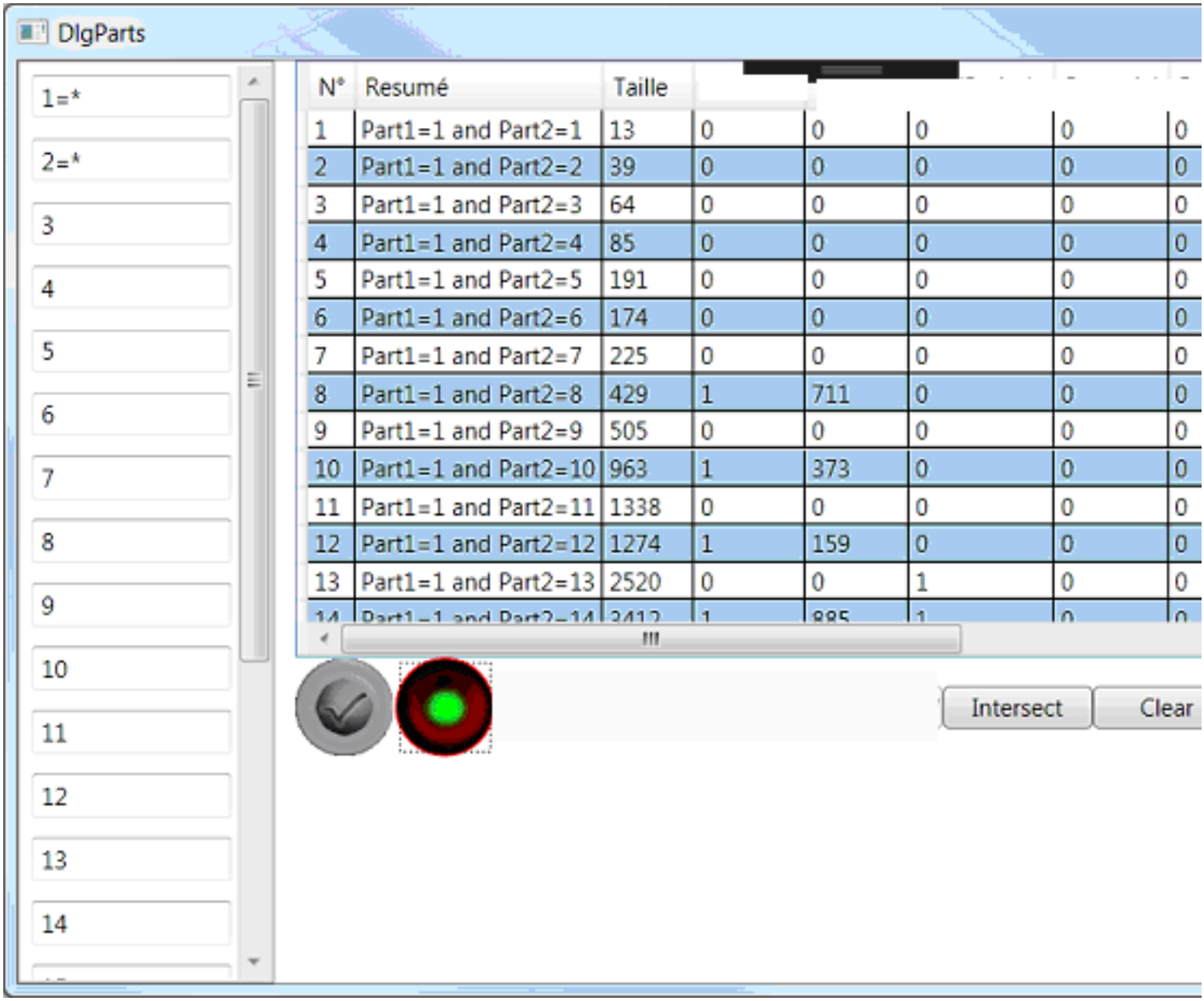
Ajouter dans tout autre projet une référence et parcourir, choisissez la DLL que vous venez de faire. Si l'élément BoutonRond n'apparaît pas dans vos outils, ajoutez le en choisissant « Eléments choisis » et encore une fois parcourir et choisir la DLL. L'élément apparaîtra et vous pouvez le placer dans votre fenêtre .

Par défaut la Bitmap vrai serait choisie, pour modifier ce choix il suffit d'ajouter dans le XAML la propriété Couleur= »bleu » (attention c'est case sensitive à moins que vous ajoutiez un toupper or tolower dans votre code du userControl).

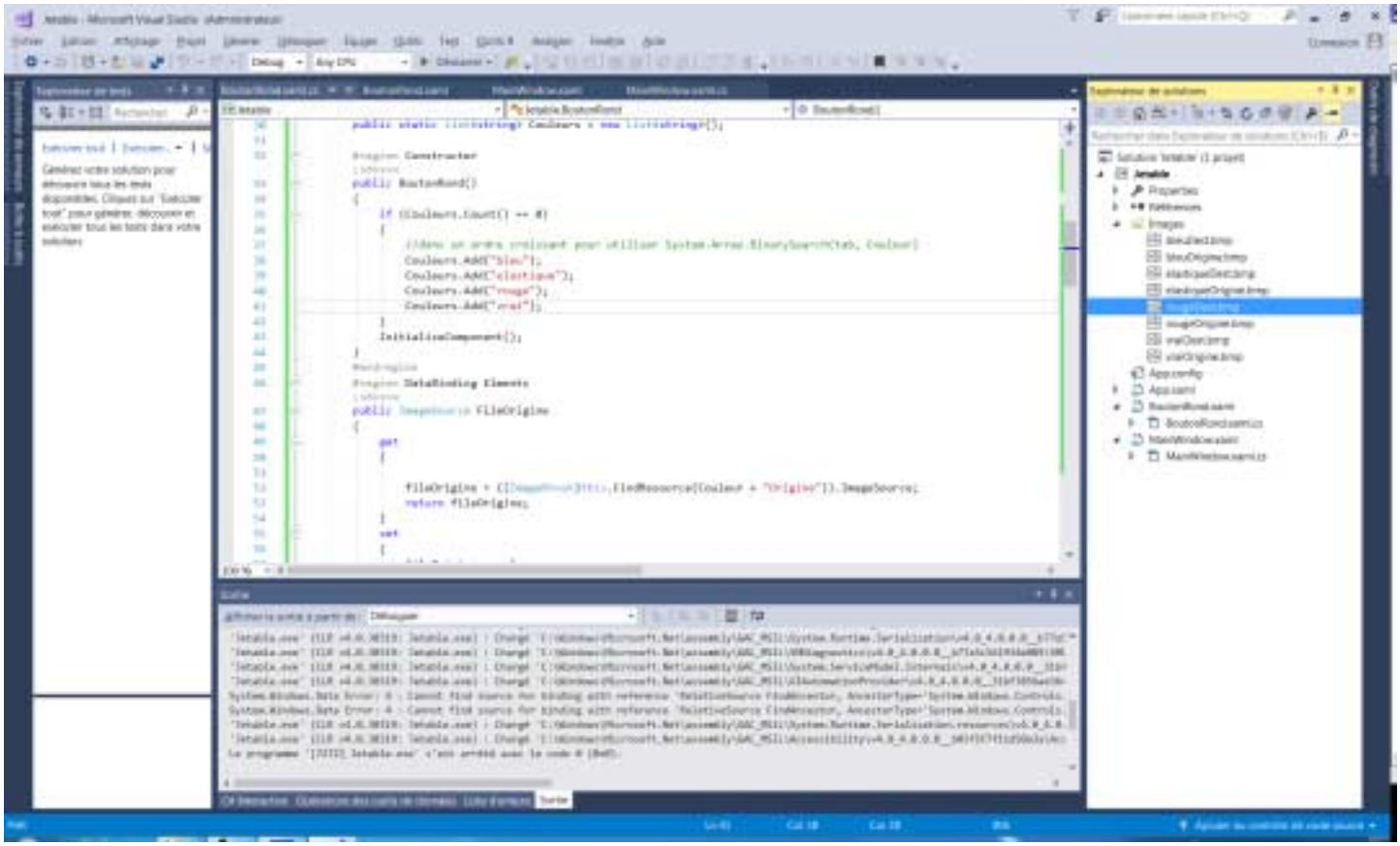
Pour la taille comme dit plus haut il faut le gérer programaticallement avec les deux méthodes SetWidth et SetHeight. Ceci pourrait être fait dans le constructeur de la fenêtre propriétaire du bouton en ajoutant 2 lignes,

NomDuBoutonRond.SetWidth(valeur) ;

NomDuBoutonRond.SetHeight(valeur) ;



Espérons avoir fait le tour de la question même si je pense honnêtement que la programmation en général est un vaste domaine et .net en particulier constitue un sous ensemble démonstratif.



BoutonRond.xaml

```
<UserControl x:Class="AsysButton.BoutonRond"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:AsysButton"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
    <UserControl.Resources>
        <ImageBrush x:Key="rougeOrigine" ImageSource ="Images/rougeOrigine.bmp"/>
        <ImageBrush x:Key="rougeDestination" ImageSource ="Images/rougeDest.bmp"/>
        <ImageBrush x:Key="bleuOrigine" ImageSource ="Images/bleuOrigine.bmp"/>
        <ImageBrush x:Key="bleuDestination" ImageSource ="Images/bleuDest.bmp"/>
        <ImageBrush x:Key="vraiOrigine" ImageSource ="Images/vraiOrigine.bmp"/>
        <ImageBrush x:Key="vraiDestination" ImageSource ="Images/vraiDest.bmp"/>
        <ImageBrush x:Key="elastiqueOrigine" ImageSource ="Images/elastiqueOrigine.bmp"/>
        <ImageBrush x:Key="elastiqueDestination" ImageSource ="Images/elastiqueDest.bmp"/>

        <ControlTemplate x:Key="BoutonRondTemplate" TargetType="Button">

            <Grid >
                <Ellipse x:Name="BoutonEllipse">
                    <Ellipse.Fill>
                        <ImageBrush ImageSource="{Binding FileOrigine,RelativeSource={RelativeSource
AncestorType=UserControl}, Mode=OneWay}"/>
                    </Ellipse.Fill>
                </Ellipse>
            </Grid>
```

```
            <ControlTemplate.Triggers>
                <Trigger Property="IsPressed" Value="True">
                    <Setter TargetName="BoutonEllipse" Property="Fill" >
                        <Setter.Value>
                            <ImageBrush ImageSource="{Binding
FileDestination,RelativeSource={RelativeSource AncestorType=UserControl}, Mode=OneWay}"/>
                        </Setter.Value>
                    </Setter>
                </Trigger>
                <Trigger Property="IsMouseOver" Value="True">
                    <Setter TargetName="BoutonEllipse" Property="Stroke" Value="red"/>
                </Trigger>
            </ControlTemplate.Triggers>
        </ControlTemplate>
    </UserControl.Resources>
    <StackPanel>
        <Button x:Name="BRUser" Height="50" Width="50" Template="{StaticResource BoutonRondTemplate}"
Click="Button_Click" Margin="0,0,0,0" />
    </StackPanel>
</UserControl>
```

Code behind

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.ComponentModel;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;



namespace AsysButton
{
    /// <summary>
    /// Logique d'interaction pour BoutonRond.xaml
    /// </summary>
    public partial class BoutonRond : UserControl,INotifyPropertyChanged
    {
        #region Définitions
        private ImageSource fileOrigine;
        private ImageSource fileDestination;
        #endregion
        public static List<string> Couleurs = new List<string>();

        #region Constructer
        public BoutonRond()
        {
            if (Couleurs.Count() == 0)
            {
                //dans un ordre croissant pour utiliser System.Array.BinarySearch(tab, Couleur)
                Couleurs.Add("bleu");
                Couleurs.Add("elastique");
                Couleurs.Add("rouge");
                Couleurs.Add("vrai");
            }
            InitializeComponent();
        }
```

```
        #endregion
        #region DataBinding Elments
        public ImageSource FileOrigine
        {
            get
            {

                fileOrigine = ((ImageBrush)this.FindResource(Couleur + "Origine")).ImageSource;
                return fileOrigine;
            }
            set
            {
                fileOrigine = value;
                NotifyPropertyChange("FileOrigine");
            }
        }
        public ImageSource FileDestination
        {
            get
            {
                fileDestination = ((ImageBrush)this.Resources[Couleur + "Destination"]).ImageSource;
                return fileDestination;
            }
            set
            {
                fileDestination = value;
                NotifyPropertyChange("FileDestination");
            }
        }
        #endregion
        #region Traitement de  l'event
        public event EventHandler OnClick;
        private void ThatWillRaisesEvent()
        {
            OnClick?.Invoke(this, new EventArgs());
        }
        private void Button_Click(object sender, RoutedEventArgs e)
        {
            ThatWillRaisesEvent();
        }
        #endregion
        #region DependencyProperty Couleur qui indique l'image
        public static readonly DependencyProperty CouleurProperty =
DependencyProperty.Register("Couleur", typeof(string), typeof(BoutonRond), new PropertyMetadata("vrai",
new PropertyChangedCallback(OnCouleurChanged)));

        public string Couleur
        {
            get
            { return (string)GetValue(CouleurProperty); }
            set
            {
                string[] tab = Couleurs.ToArray();
                if (System.Array.BinarySearch(tab, Couleur) >= 0)
                {
                    SetValue(CouleurProperty, value);
                    FileOrigine = ((ImageBrush)this.Resources[Couleur + "Origine"]).ImageSource;
                    FileDestination = ((ImageBrush)this.Resources[Couleur +
"Destination"]).ImageSource;
                }
            }
        }
        private static void OnCouleurChanged(DependencyObject depObj,
DependencyPropertyChangedEventArgs e)
        {
            BoutonRond bRControl = depObj as BoutonRond;
            bRControl.OnCouleurChanged(e);
        }
        private void OnCouleurChanged(DependencyPropertyChangedEventArgs e)
        {
```

```
            if (e.NewValue != null) Couleur = e.NewValue.ToString();
        }
        #endregion
        #region NotifyPropertyChanged
        public event PropertyChangedEventHandler PropertyChanged;
        private void NotifyPropertyChange(String property)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(property));
        }
        #endregion
        #region Méthodes Extra
        public void SetWidth(double dWidth)
        {
            this.BRUser.Width = dWidth;
            this.Width = this.BRUser.Width;
        }
        public void SetHeight(double dHeight)
        {
            this.BRUser.Height = dHeight;
            this.Height = this.BRUser.Height;
        }
        #endregion
    }
}
```

MainWindow.xaml

```
<Window x:Class="AsysButton.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:AsysButton"
        mc:Ignorable="d"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <local:BoutonRond x:Name="RB" Couleur = "{Binding
SelectedItem,ElementName=comboCouleursRB,Mode=TwoWay}" HorizontalAlignment="Left" Height="100"
VerticalAlignment="Top" Width="100" Margin="180,45,0,0" OnClick="RB_OnClick"/>
        <ComboBox x:Name="comboCouleursRB" SelectedItem="{Binding Path=Couleur, Mode=TwoWay}"
IsReadOnly="True" HorizontalAlignment="Left" Height="30" Margin="210,230,0,0" VerticalAlignment="Top"
Width="170"/>
        <TextBox Name="tbRB" HorizontalAlignment="Left" Height="25" Margin="30,185,0,0"
TextWrapping="Wrap" Text="{Binding Path=Couleur, Mode=TwoWay}" VerticalAlignment="Top" Width="110"/>
        <Button Content="Button" HorizontalAlignment="Left" Height="30" Margin="30,230,0,0"
VerticalAlignment="Top" Width="110" Click="Button_Click"/>
    </Grid>
</Window>
```

MainWindow Code behind

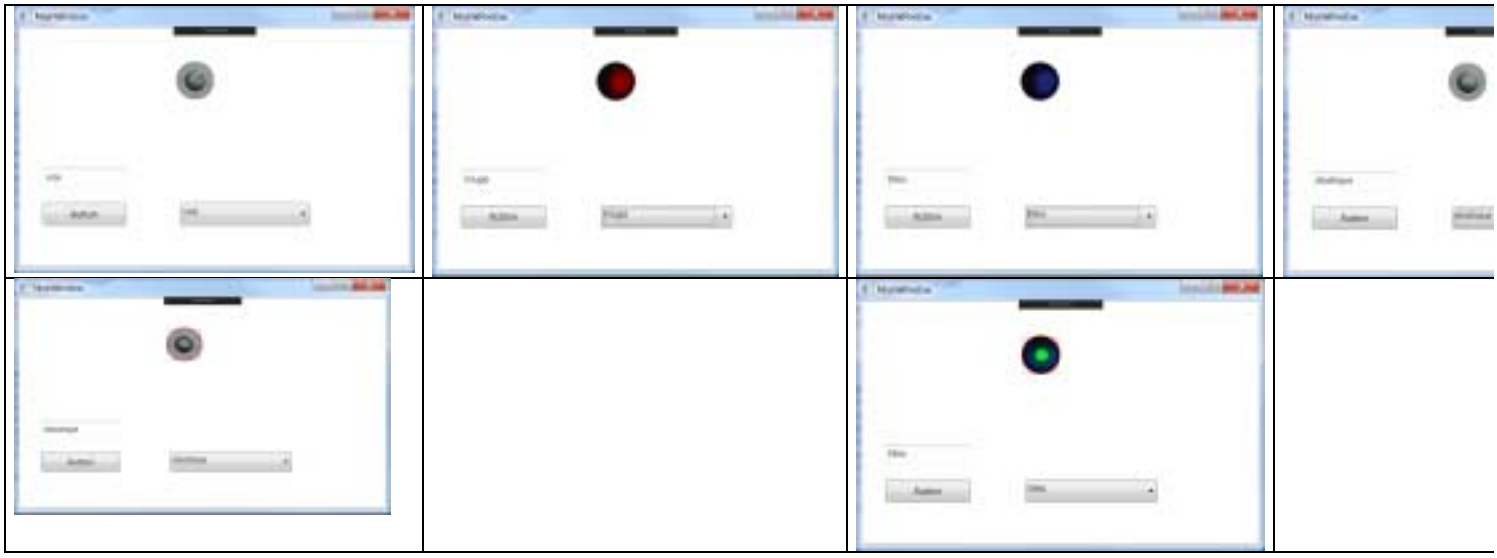
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
```

```
namespace AsysButton
{
    /// <summary>
    /// Logique d'interaction pour MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            this.DataContext = new BoutonRond();
            //remplir la combobox par les valeurs autorisées
            BoutonRond.Couleurs.ForEach(item => { comboCouleursRB.Items.Add(item); });
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            //changer le bouton utilize en rouge
            RB.SetCurrentValue(BoutonRond.CouleurProperty, "rouge");
        }

        private void RB_OnClick(object sender, EventArgs e)
        {

        }
    }
}
```



Choisir les éléments