Créer un contrôle utilisateur

C# – Fiche Technique

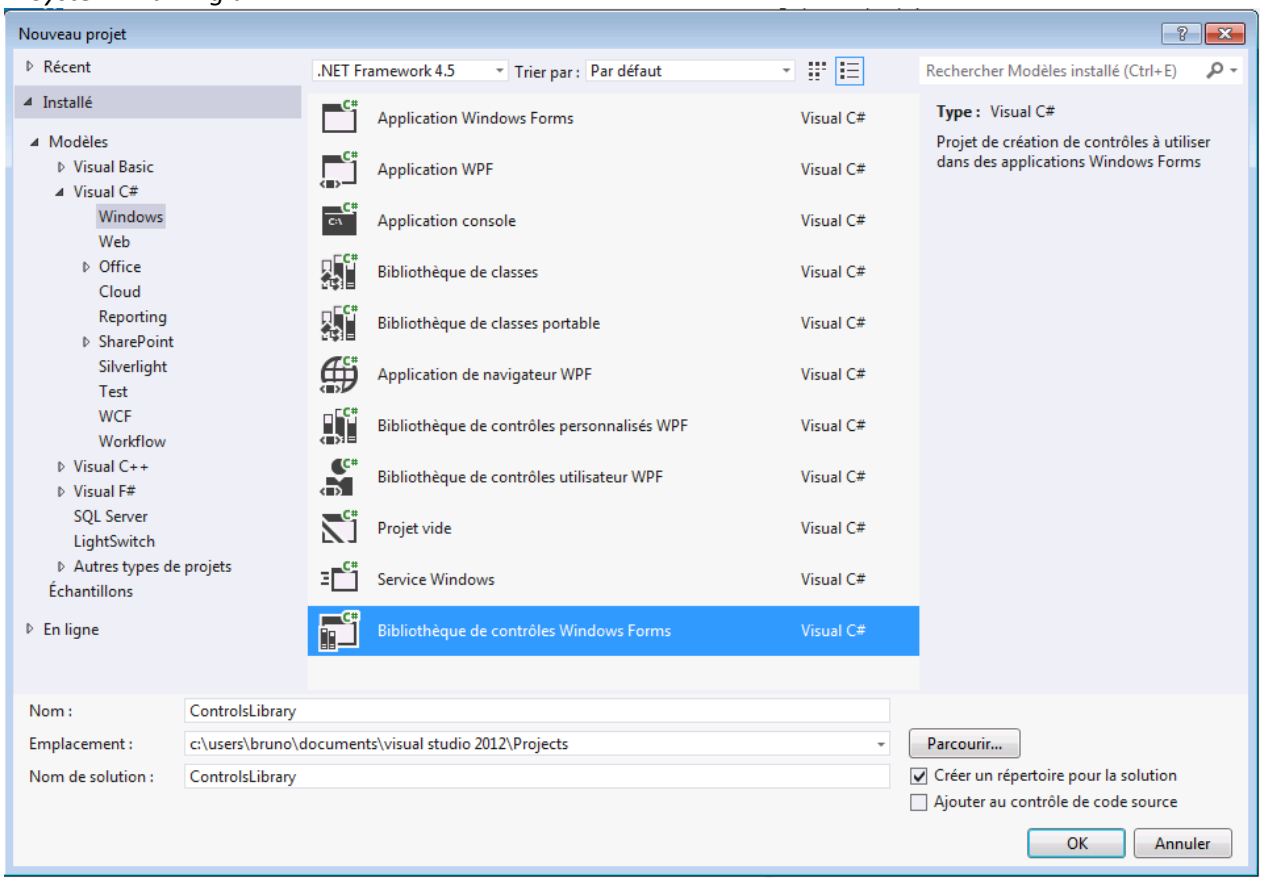
Un contrôle utilisateur permet de regrouper de manière logique des contrôles afin d'obtenir une entité graphique réutilisable sur différents formulaires. Un contrôle utilisateur hérite de la classe **UserControl**.

Il va s'agir ici de créer un contrôle personnalisé simple appelé **NavBar,** permettant de placer sur vos formulaires une « barre de navigation » dotée de quatre boutons et des fonctionnalités suivantes :

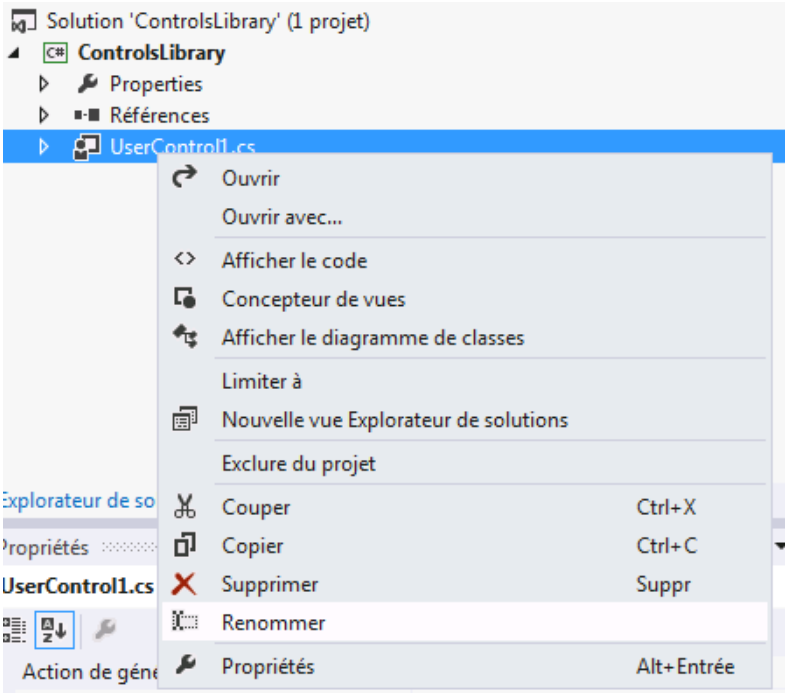
- la possibilité de choisir le libellé et l'icône des boutons via des propriétés ;
- la capacité d'émettre un évènement permettant d'identifier le bouton du contrôle sur lequel l'utilisateur a cliqué pour écrire du code réagissant en conséquence.

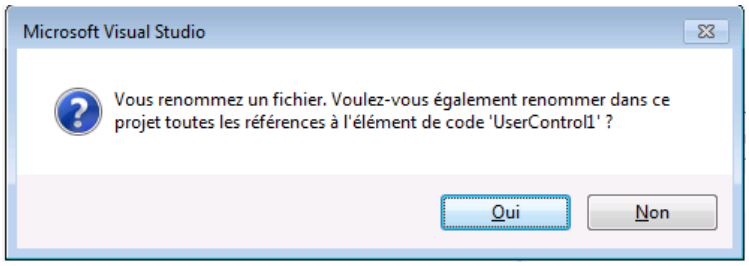
1 – Création de la classe contrôle utilisateur

- Créez un nouveau projet de type *Bibliothèque de contrôles Windows Forms*, nommez-le **ControlsLibrary**. Il constituera notre bibliothèque d'objets pour l'interface graphique. La création sous forme de projet *Bibliothèque de contrôles Windows Forms* permet d'obtenir les références vers les bibliothèques du framework requises pour les objets de l'interface graphique : *System.Windows.Forms.dll* et *System.Drawing.dll*.

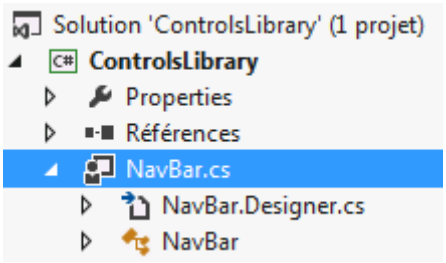


- Renommez le contrôle utilisateur en *NavBar*. Validez le changement.





- Visual Studio affiche le concepteur graphique pour notre contrôle utilisateur. Celui-ci fonctionne comme le concepteur graphique pour un objet *Form* et affiche la surface « de base » du contrôle, largeur et hauteur qu'il possédera par défaut lorsqu'on le posera sur un formulaire. C'est sur cette surface que sont disposés les contrôles standards qui constituent l'interface graphique du contrôle. L'analogie avec un formulaire ne s'arrête pas là puisque, dans l'Explorateur de solutions, il apparaît qu'un fichier intitulé *NavBar.Designer.cs* a également été créé.



- Comme pour les formulaires, ce fichier contient une définition de classe partielle dont le code est modifié automatiquement par les opérations effectuées par le programmeur dans le concepteur graphique. Ouvrez ce fichier pour consulter le code préliminaire généré. La classe déclare un objet **IContainer** (initialisé par la méthode `InitializeComponent` et nettoyé par la méthode `Dispose`) qui lui donne la capacité de contenir d'autres contrôles, nécessaires pour définir son interface graphique à partir de contrôles standards :

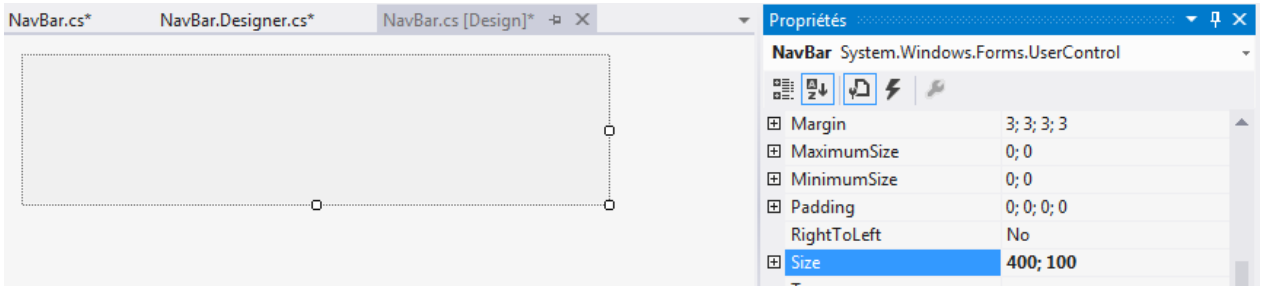
```
namespace ControlsLibrary
{
    partial class NavBar
    {
        /// <summary>
        /// Variable nécessaire au concepteur.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Nettoyage des ressources utilisées.
        /// </summary>
        /// <param name="disposing">true si les ressources managées doivent être supp
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Code généré par le Concepteur de composants

        /// <summary>
        /// Méthode requise pour la prise en charge du concepteur - ne modifiez pas
        /// le contenu de cette méthode avec l'éditeur de code.
        /// </summary>
        private void InitializeComponent()
        {
            components = new System.ComponentModel.Container();
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        }

        #endregion
    }
}
```

- Comme pour les formulaires, il faut éviter autant que possible les interventions directes dans ce fichier, en utilisant les outils dédiés que sont le concepteur et la fenêtre-outil Propriétés. Le code spécifique, ou métier, du contrôle est quant à lui saisi dans la feuille de code, accessible en demandant à afficher le code de *NavBar.cs*. Cette feuille de code est pour l'instant vide à l'exception de la déclaration de la classe, de l'héritage vers la classe UserControl :

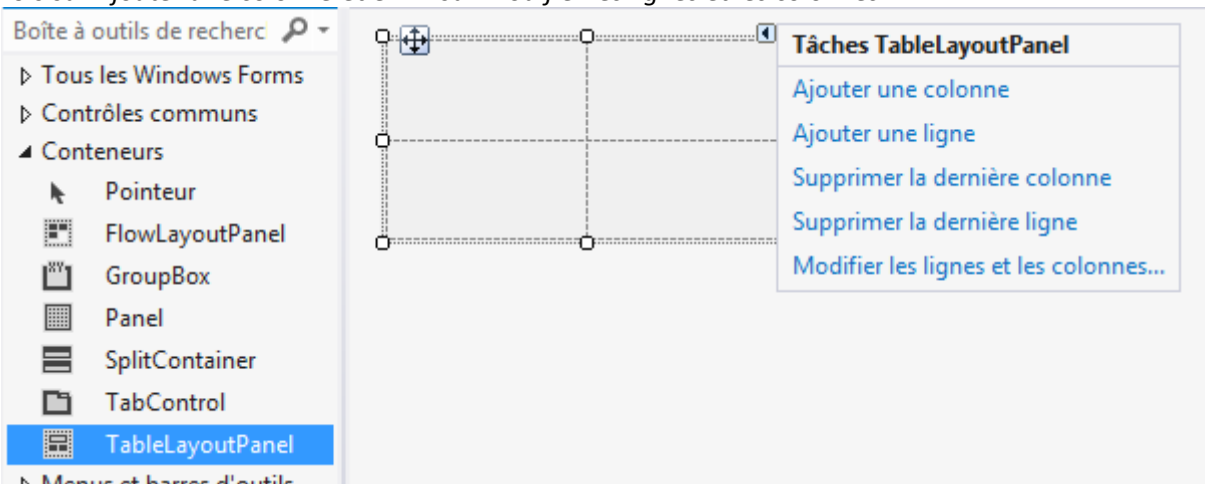
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ControlsLibrary
{
    public partial class NavBar: UserControl
    {
        public NavBar()
        {
            InitializeComponent();
        }
    }
}
```

2 – Définir l'interface graphique du contrôle utilisateur

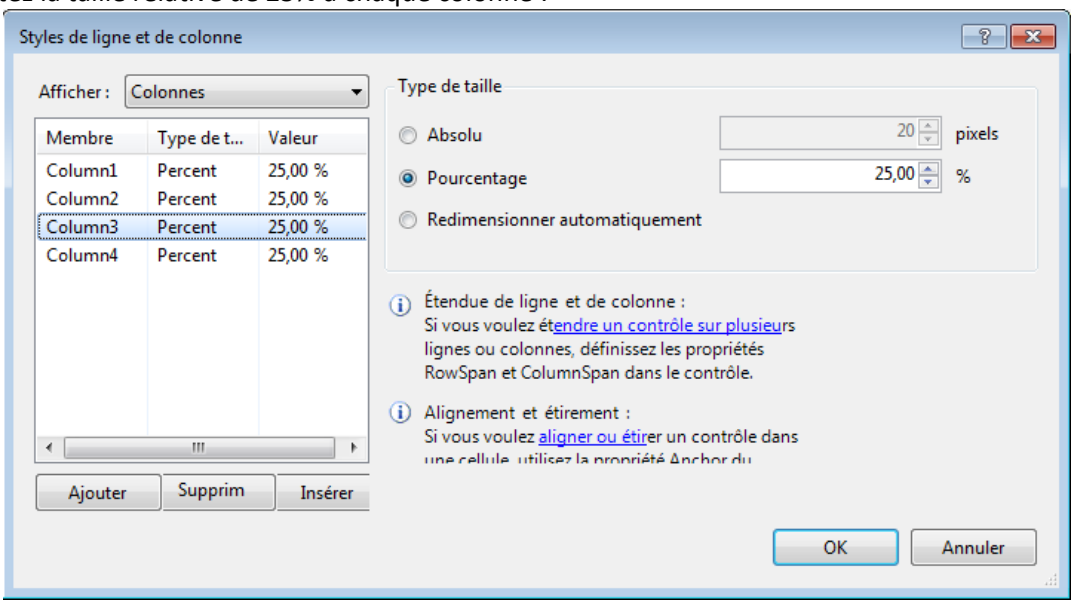
- Activez le concepteur et utilisez les poignées de redimensionnement pour donner sa forme initiale au contrôle (ou utiliser la propriété **Size** dans l'outil Propriétés). Ces valeurs seront celles par défaut lorsque le programmeur qui utilisera le contrôle le posera sur un formulaire, mais il aura bien entendu la possibilité de les redéfinir à sa convenance...



- À partir de la Boîte à outils, ajoutez un **TableLayoutPanel**. Cliquez sur *Supprimer la dernière ligne* puis deux fois sur *Ajouter une colonne* et enfin sur *Modifier les lignes et les colonnes*.



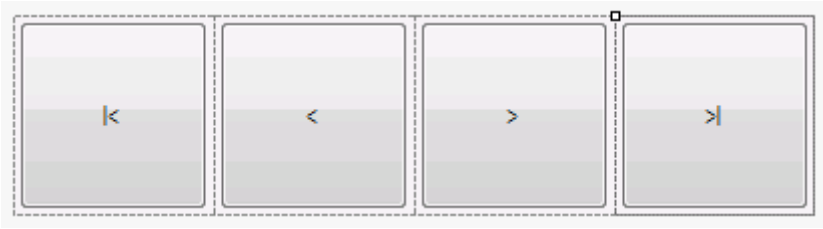
- Affectez la taille relative de 25% à chaque colonne :



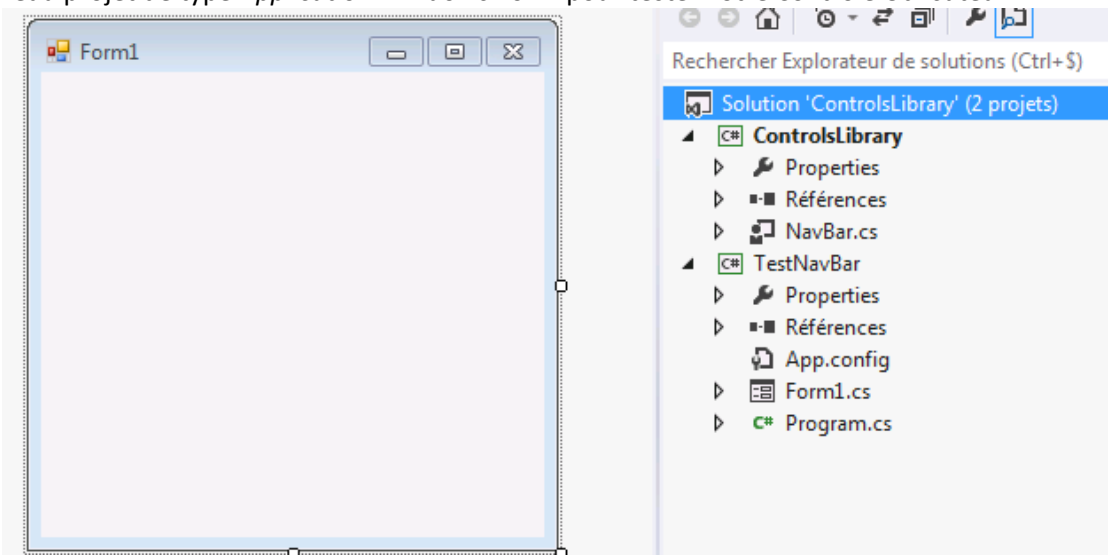
- Mettez la propriété **Dock** de votre **TableLayoutPanel** de votre Contrôle Utilisateur à **Fill** :

- Ajoutez successivement quatre objets **Button**, appelés respectivement `BtnPremier`, `BtnPrecedent`, `BtnSuivant`, `BtnDernier` (propriété **(Name)**), dans chacune des cases du **TableLayoutPanel**. Définissez respectivement la propriété **Text** avec les valeurs `|<`, `<`, `>`, `>|`, qui seront les libellés par défaut des

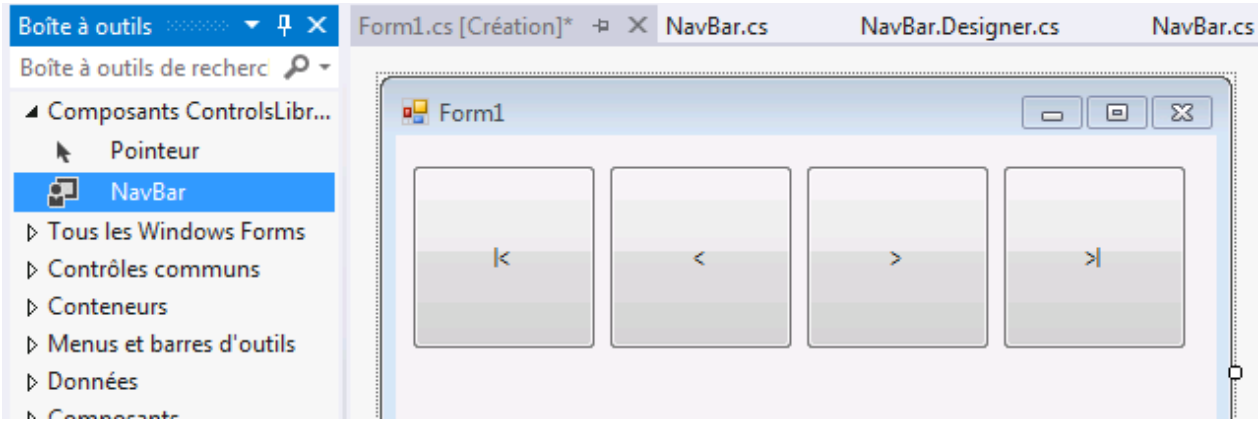
composants du contrôle. Mettez la propriété **Dock** des boutons à **Fill** pour qu'ils occupent toute la place disponible :



- Le contrôle peut être d'ores et déjà être testé dans un formulaire « client ». Pour cela, il faut **impérativement générer, ou regénérer, le projet** contenant la classe. Il en sera de même après chaque modification du contrôle pour qu'elle soit prise en compte au niveau du projet client. Puis ajoutez un nouveau projet de type *Application Windows Form* pour tester votre *Contrôle Utilisateur* :



- Dans la boîte à outils, votre *Contrôle Utilisateur* NavBar apparaît dans une nouvelle rubrique : *ControlsLibrary*. Déplacez-le sur la fenêtre comme n'importe quel autre composant, une instance du contrôle étant créée dans le code du formulaire.

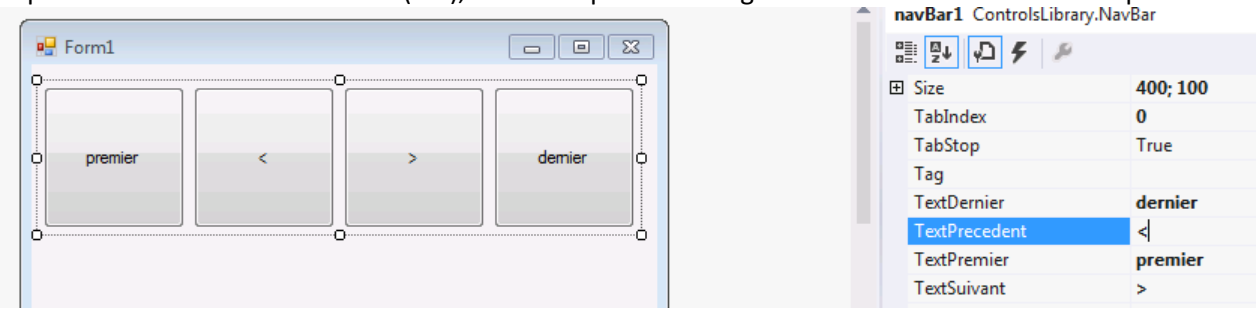


3 – Définir des propriétés spécifiques au contrôle

- Le contrôle NavBar est déjà doté de toutes les propriétés communes aux composants graphiques, mais il va falloir le doter de propriétés permettant à son utilisateur de redéfinir les libellés et les icônes des boutons le constituant.
- Ouvrez la feuille de code de la classe NavBar. Écrivez des propriétés intitulées `TextPremier`, `TextPrecedent`, `TextSuivant` et `TextDernier`. Ces propriétés vont modifier les propriétés **Text** de chaque bouton :

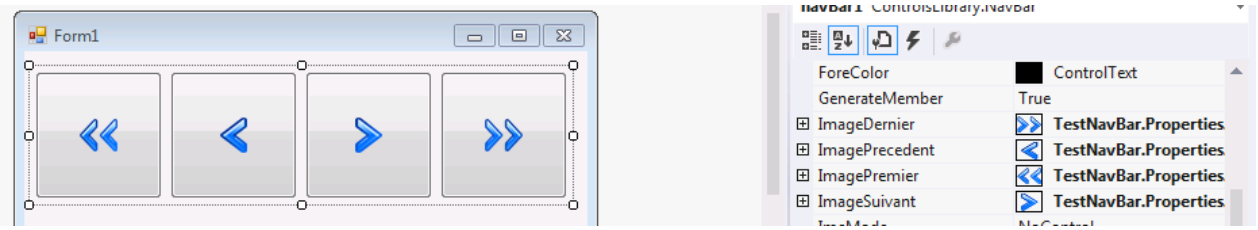
```
#region "Définir les propriétés complémentaires propres au controle"

public string TextPremier
{
    get { return btnPremier.Text; }
    set { btnPremier.Text = value; }
}

public string TextPrecedent
{
    get { return btnPrecedent.Text; }
    set { btnPrecedent.Text = value; }
}
```

- Régénérez le projet **ControlsLibrary**, réactivez le concepteur de **Form1** dans **TestNavBar** et sélectionnez l'objet **NavBar1**. L'outils Propriétés affiche maintenant les nouvelles propriétés de l'objet (Get) et vous permet de redéfinir leur valeur (Set), automatiquement assignée au libellé du bouton correspondant.



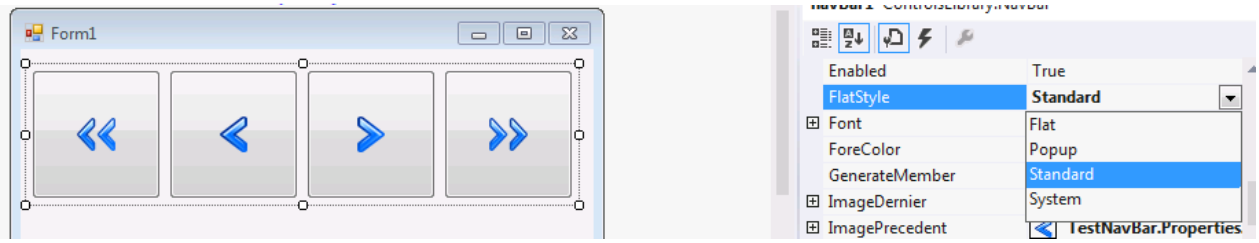
- Il en est de même si vous souhaitez définir des propriétés pour affecter une icône pour chaque bouton :

```
public Image ImagePremier {
    get { return btnPremier.Image; }
    set { btnPremier.Image = value; }
}

public Image ImagePrecedent ...

public Image ImageSuivant ...

public Image ImageDernier ...
```



- Ou pour changer l'apparence des boutons :

```
public FlatStyle FlatStyle
{
    get { return btnPremier.FlatStyle; }
    set
    {
        btnPremier.FlatStyle = value;
        btnPrecedent.FlatStyle = value;
        btnSuivant.FlatStyle = value;
        btnDernier.FlatStyle = value;
    }
}

public int ButtonBorder
{
    get { return btnPremier.FlatAppearance.BorderSize; }
    set
    {
        btnPremier.FlatAppearance.BorderSize = value;
        btnPrecedent.FlatAppearance.BorderSize = value;
        btnSuivant.FlatAppearance.BorderSize = value;
        btnDernier.FlatAppearance.BorderSize = value;
    }
}
```



4 – Définir un événement pour le contrôle

- Au clic sur l'un des boutons de son interface par l'utilisateur de l'application, le contrôle *NavBar* va réagir par l'émission d'un événement. Le formulaire pourra ainsi définir du code pour gérer cet événement. Cet événement sera conforme aux standards de Visual Studio en fournissant deux arguments, une référence vers l'objet à l'origine de l'évènement et une instance d'une classe dérivée de **System.EventArgs** qui détiendra dans une propriété l'information sur le bouton activé. Dans la classe **NavBar**, créez pour commencer une énumération, **NavActionEnum**, pour les différentes valeurs possibles pour cette propriété :

```
#region "Evenement de la NavBar"
public enum NavActionEnum
{
    premier,
    precedent,
    suivant,
    dernier
}
```

- Créez ensuite, sous forme d'une classe interne à **NavBar**, la classe intitulée **NavBarEventArgs** qui fournira l'argument de type **EventArgs** pour l'évènement. Définissez dans cette classe la propriété **NavAction**, destinée à stocker la valeur **NavActionEnum** correspondant au bouton cliqué, et un constructeur prenant en paramètre une valeur de type **NavActionEnum** destinée précisément à valoriser la propriété **NavAction** de l'instance à sa création :

```
public class NavBarEventArgs : EventArgs
{

    private NavActionEnum _navAction;
    public NavActionEnum navAction
    {
        get { return _navAction; }
        set { _navAction = value; }
    }

    public NavBarEventArgs(NavActionEnum navAction)
    {
        this.navAction = navAction;
    }
}
```

- Déclarez maintenant l'évènement lui-même, intitulé **`Navigation`**, ainsi que la méthode **onNavigation** propageant l'évènement si un gestionnaire est associé pour le traiter :

```
public event EventHandler<NavBarEventArgs> Navigation;
protected virtual void onNavigation(NavBarEventArgs e)
{
    EventHandler<NavBarEventArgs> handler = Navigation;
    if (handler != null) handler(this, e);
}
```

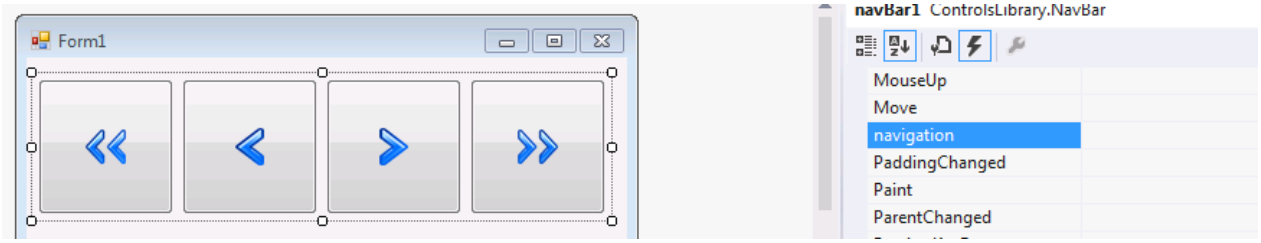
- Pour finir, il faut faire émettre par le contrôle l'évènement. Il faut donc gérer dans **NavBar** l'évènement `Click` sur chacun des boutons et appeler notre méthode de déclenchement d'évènement **onNavigation**. L'argument sera systématiquement une nouvelle instance de **NavBarEventArgs** construite avec la valeur de **NavActionEnum** correspondant au bouton (`premier` pour BtnPremier, `dernier` pour BtnDernier, etc.) :

```
#region "propager l'évènement aux composants utilisant le userControl"
private void btnPremier_Click(object sender, EventArgs e)
{
    onNavigation(new NavBarEventArgs(NavActionEnum.premier));
}

private void btnPrecedent_Click(object sender, EventArgs e)
{
    onNavigation(new NavBarEventArgs(NavActionEnum.precedent));
}

private void btnSuivant_Click(object sender, EventArgs e)
{
    onNavigation(new NavBarEventArgs(NavActionEnum.suivant));
}

private void btnDernier_Click(object sender, EventArgs e)
{
    onNavigation(new NavBarEventArgs(NavActionEnum.dernier));
}
#endregion
```

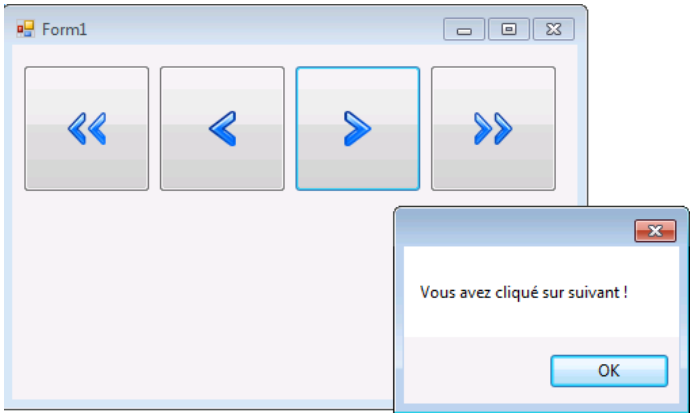
- Pour que l'événement que vous avez créé soit l'événement par défaut pour ce contrôle, rajouter le tag suivant avant la déclaration de la classe **NavBar**.

```
[DefaultEvent("Navigation")]
```

- Regénérez la bibliothèque **ControlsLibrary** et retourner dans le projet **TestNavBar**. Cliquez sur la NavBar, parmi ses événements, un nouveau est apparu : navigation. Double-cliquez dessus pour générer la méthode à appeler lorsque l'événement se produit ou bien double-cliquez sur la NavBar dans votre fenêtre, cela revient au même puisque cet événement est celui par défaut.




```
private void navBar1_Navigation(object sender, ControlsLibrary.NavBar.NavBarEventArgs e)
{
    MessageBox.Show("Vous avez cliqué sur " + e.navAction.ToString() + " !");
}
```



5 – Définir des propriétés pour activer ou désactiver une partie du contrôle

- Pour être un peu plus complet, le contrôle va définir des propriétés qui vont permettre au programmeur de gérer l'état des boutons en fonction de la position dans le groupe d'éléments parcourus à l'aide de la NavBar. Ces propriétés vont lui permettre de désactiver `BtnPrecedent` et `BtnPremier` lorsque le premier élément du groupe sera atteint, `BtnSuivant` et `BtnDernier` dans le cas du dernier élément.

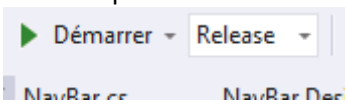
```
#region "Gestion de l'activité des boutons"

public bool avantEnable
{
    get { return btnPrecedent.Enabled; }
    set
    {
        btnPremier.Enabled = value;
        btnPrecedent.Enabled = value;
    }
}

public bool apresEnable
{
    get { return btnSuivant.Enabled; }
    set
    {
        btnSuivant.Enabled = value;
        btnDernier.Enabled = value;
    }
}

#endregion
```

- Regénérez le projet **ControlsLibrary** et dans le code de **Form1** et modifiez la procédure événementielle `NavBar1_navigation` afin de tester les nouvelles propriétés de l'objet (notez l'utilisation de l'argument `sender` pour récupérer la référence sur l'objet émetteur de l'évènement) :

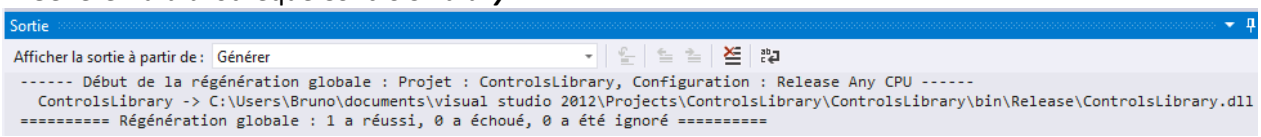
```
int courant = 2;
int max = 4;
private void navBar1_Navigation(object sender, ControlsLibrary.NavBar.NavBarEventArgs e)
{
    MessageBox.Show("Vous avez cliqué sur " + e.navAction.ToString() + " !");

    ControlsLibrary.NavBar barre = (ControlsLibrary.NavBar)sender;
    switch (e.navAction)
    {
        case ControlsLibrary.NavBar.NavActionEnum.premier :
            courant = 0;
            break;
        case ControlsLibrary.NavBar.NavActionEnum.precedent :
            courant -= 1;
            break;
        case ControlsLibrary.NavBar.NavActionEnum.suivant :
            courant += 1;
            break;
        case ControlsLibrary.NavBar.NavActionEnum.dernier :
            courant = max;
            break;
    }
    barre.avantEnable = (courant != 0);
    barre.apresEnable = (courant != max);
}
```

6 – Utiliser son contrôle dans d'autres solutions

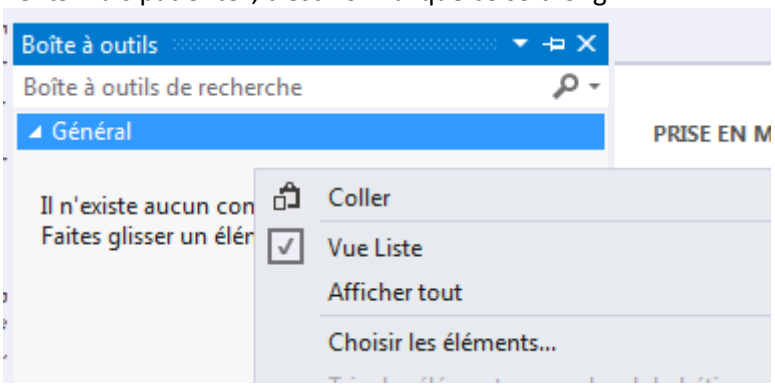
- Nous avons vu comment créer un nouveau composant graphique qui est indépendant de tout projet. Cela n'a un réel intérêt que si nous pouvons le réutiliser dans d'autres projets.
- La première étape est de créer une bibliothèque. Passez en mode ***Release***.



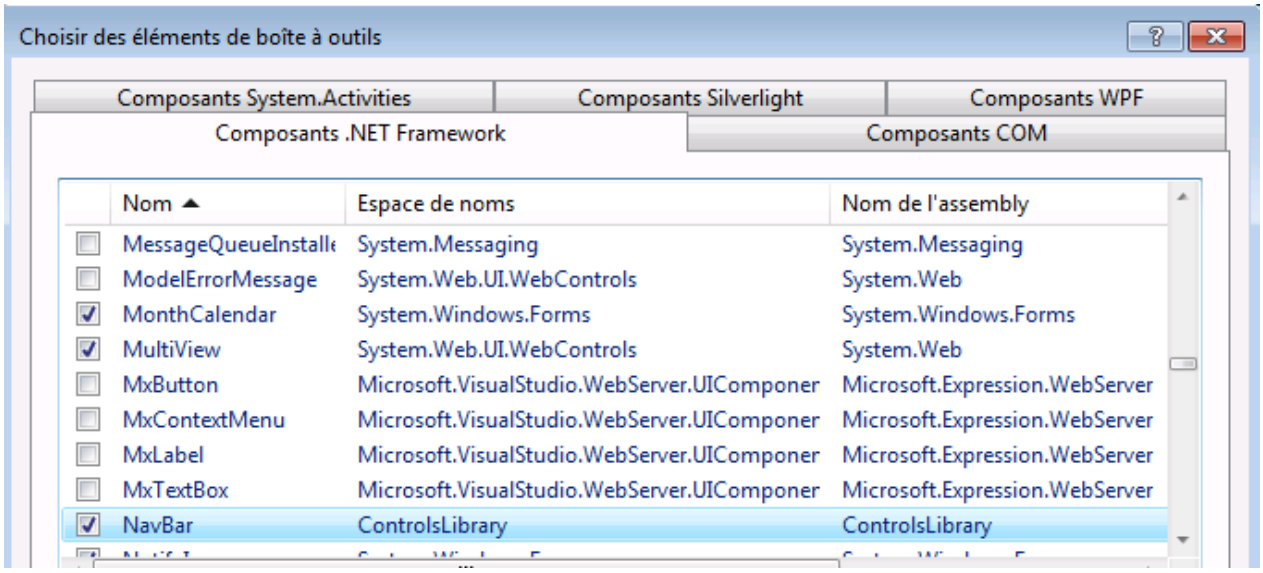
- Générez la bibliothèque ***ControlsLibrary*** :



- Dans un autre *Visual Studio*, cliquez avec le bouton secondaire de la souris dans la boîte à outils et cliquez sur ***Choisir les éléments***. Puis patientez, c'est normal que ce soit long…



- Cliquez sur parcourir et choisissez la bibliothèque que vous venez de générer. Elle est située par rapport à la racine de votre projet ***ControlsLibrary*** à l'adresse suivante : ControlsLibrary\bin\Release\ ControlsLibrary.dll.



- Après avoir validé la fenêtre, votre contrôle *NavBar* apparaît dans votre boîte à outils :