



Concepteur Développeur en Informatique

Développer des composants d'interface

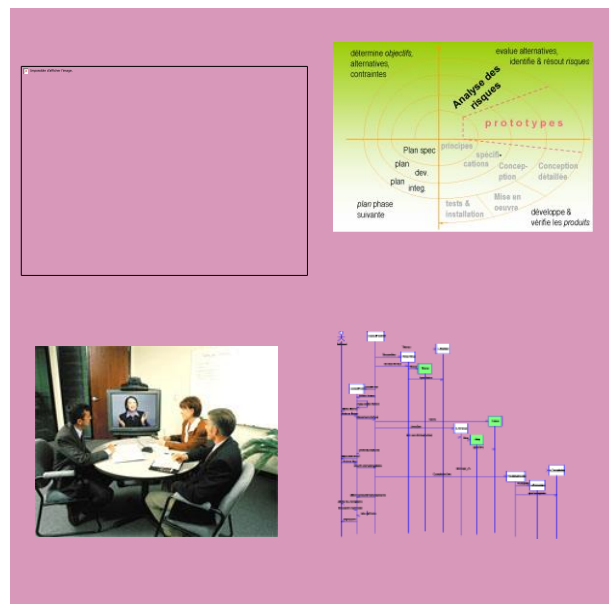
Formulaires Windows – Partie 2

Accueil

Apprentissage

PAE

Evaluation



Localisation : U03-E03-S02

SOMMAIRE

1	Introduction	3
2	Les contrôles de texte	3
2.1	La zone de texte TextBox	3
2.2	Le contrôle de texte évolué MaskedTextBox	5
2.3	Le contrôle d’affichage Label	8
2.3.1	Les LinkLabels	9
3	Les contrôles UpDown	9
4	Boutons et case à cocher	10
4.1	Le contrôle de commande (Button)	11
4.2	La case à cocher (CheckBox)	11
4.3	Le Bouton Radio (RadioButton)	13
5	Les Conteneurs	14
5.1	Contrôles GroupBox et Panel	14
5.2	Contrôles FlowLayoutPanel et TableLayoutPanel	14
5.3	Contrôle SplitContainer	16
5.4	Contrôle TabControl	17
	

1 Introduction

Ce document présente les principaux contrôles graphiques par catégorie.






Ce document ne saurait être exhaustif.

Il reprend les propriétés et méthodes principales des contrôles, sans plus.

Pour approfondir le sujet consultez la documentation relative à la classe de chacun de ces contrôles.

2 Les contrôles de texte

Les contrôles présentés dans ce chapitre permettent d'acquérir et de restituer des valeurs de texte.

	TextBox	Contrôle permettant à l'utilisateur de saisir des caractères au clavier.
	MaskedTextBox	Contrôle TextBox amélioré qui prend en charge une syntaxe déclarative pour accepter ou refuser une entrée d'utilisateur
	Label	Zone d'affichage de texte non modifiable par l'utilisateur
	LinkLabel	Zones d'affichage représentées comme des hyperliens
	RichTextBox	Contrôle implémentant des méthodes de mise en forme de texte évoluées. Il offre des fonctionnalités similaires à un traitement de texte.

2.1 La zone de texte TextBox

La zone de texte est le contrôle d'édition standard de Windows qui permet de saisir du texte.

La classe **TextBox** qui gère les zones de texte dérive de la classe **TextBoxBase**, classe de base pour les classes **TextBox** et **RichTextBox**.

Le contrôle **TextBox** est une simple zone de saisie.

La propriété **MultiLine** permet de saisir plusieurs lignes de texte, dont on peut extraire le contenu avec un tableau de **string** avec la propriété **Lines**.

La propriété **ReadOnly** fait passer le contrôle en lecture seule.

Il est du rôle du développeur d'implémenter toute la logique de vérification des données saisies dans une **TextBox**.

La propriété **MaxLength** indique le nombre de caractères maximum qui peuvent être saisis.

Principales propriétés et méthodes

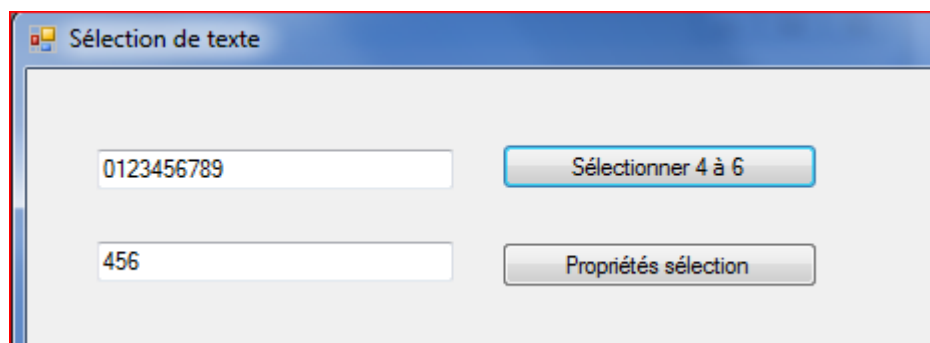
Propriétés	
CharacterCasing	Types de caractères que l'on peut taper (Lower, Normal, Upper)
Lines	Tableau de chaînes de caractères contenant chacun des lignes d'une zone d'édition multi-lignes
MultiLine	Permet de pouvoir saisir plusieurs lignes
MaxLength	Nombre maximal de caractères que l'utilisateur peut introduire dans la zone
Modified	Indique si l'utilisateur a modifié le contenu de la zone d'édition
PasswordChar	Spécifie un symbole de remplacement comme caractère de substitution pour cacher des données sensibles
ReadOnly	Indique si la zone d'édition est en lecture seule
ScrollBars	Indique quelles barres de défilement sont éventuellement affichées (si MultiLine = true)
SelectedText	Texte sélectionné
SelectionLength	Nombre de caractères sélectionnés
SelectionStart	Indice du premier caractère sélectionné
Text	Contient le texte saisi ou affiché dans la zone de texte
TextAlign	Cadrage du texte dans la zone d'édition
TextLenght	Longueur du texte saisi ou affiché
WordWrap	En cas de saisie Multi-lignes, permet de renvoyer la saisie à la ligne au bord droit du contrôle
Méthodes	
void Clear()	Supprime le texte

Sélection de texte par programme

Ce premier exemple effectue la sélection par programme du texte selon une position et une longueur et le copie dans une autre boîte de texte.

```
private void btnCopierSel_Click(object sender, EventArgs e)
{
    txtCible.Multiline = false;
    txtCible.Height = 20;
    txtCible.Clear();

    txtSource.SelectionStart = 4;
    txtSource.SelectionLength = 3;
    txtCible.Text = txtSource.SelectedText;
}
```



Propriétés du texte sélectionné avec la souris

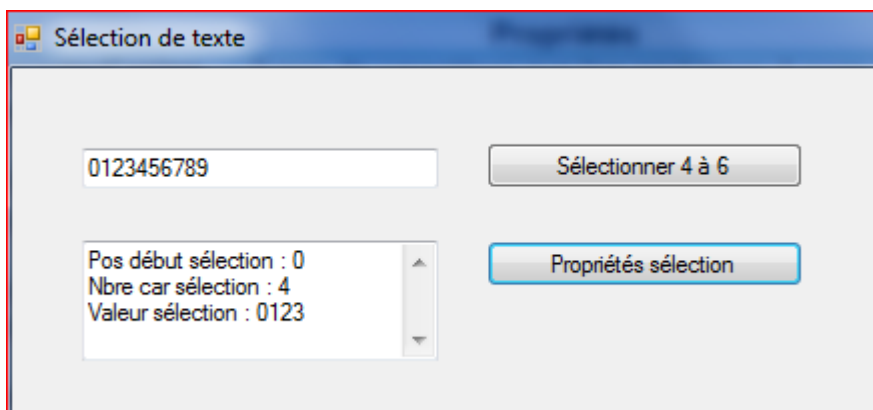
Ce deuxième exemple affiche les propriétés du texte sélectionné à la souris.

```
private void btnPropSel_Click(object sender, EventArgs e)
{
    txtCible.Multiline = true;
    txtCible.Height = 60;
    txtCible.ScrollBars = ScrollBars.Vertical;
    txtCible.Text = "Pos début sélection : " + txtSource.SelectionStart + "\r\n";
    txtCible.Text += "Nbre car sélection : " + txtSource.SelectionLength + "\r\n";
    txtCible.Text += "Valeur sélection : " + txtSource.SelectedText;
}
```

Les propriétés de la liste cible sont modifiées afin de proposer un affichage multi-lignes. Remarquez l'usage de `\r\n` pour passer à la ligne.

0123456789

Sélection à la souris des chiffres 0 à 3



2.2 Le contrôle de texte évolué MaskedTextBox

Contrôle dérivé du contrôle **TextBox**. Il en reprend donc toutes ses fonctionnalités.

Le contrôle **MaskedTextBox** possède une propriété **Mask** qui permet de restreindre la saisie selon un modèle prédéfini.

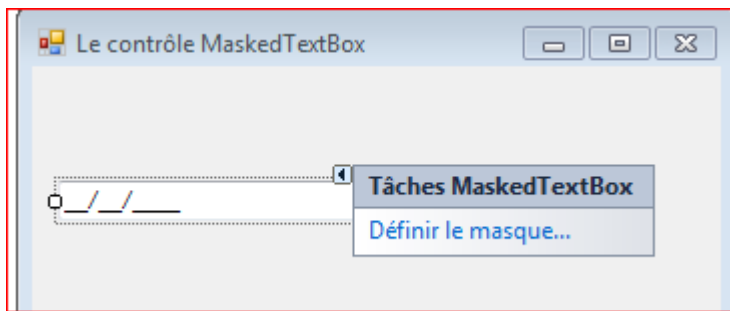
Plusieurs types de masques sont proposés par défaut, avec la possibilité de définir ses propres masques.

La propriété **ValidatingType** permet de définir le type de données attendu. Elle est couplée à l'instruction `typeof()`.

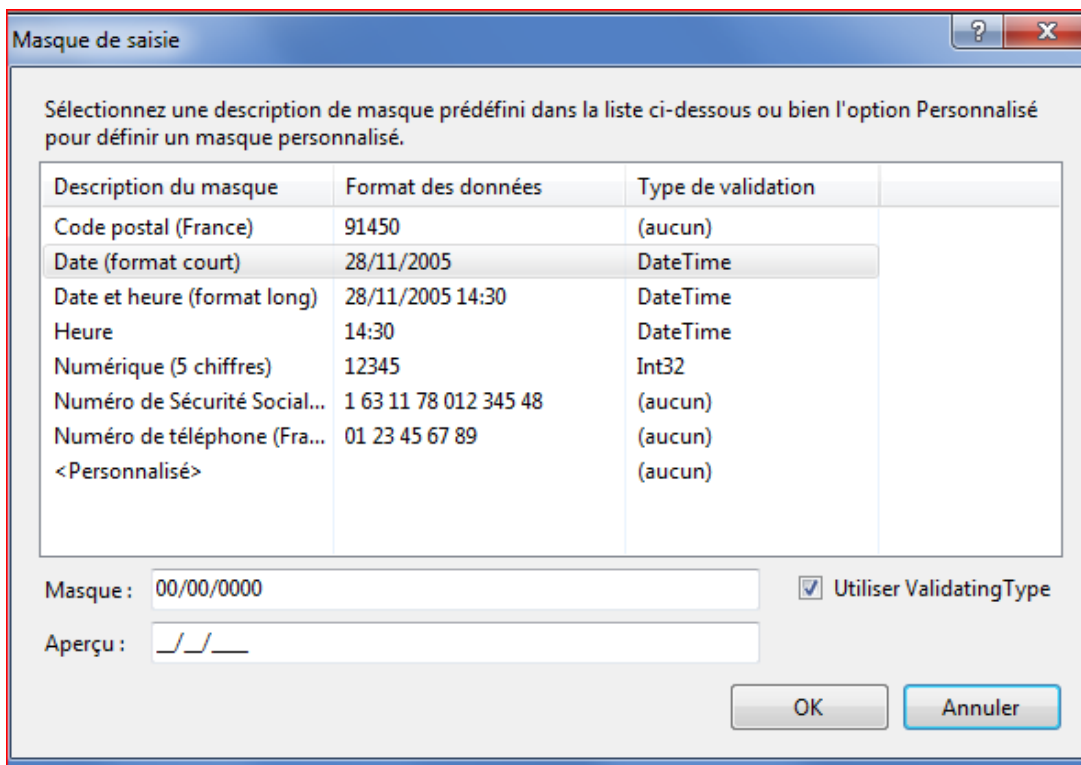
Deux événements à prendre en considération en priorité :

- Détection d'un caractère non valide
- Saisie complète conforme au masque est validée

Vous pouvez sélectionner un masque prédéfini en cliquant sur le smartTag.



Sélection du masque, ici une date au format court.



Principales Propriétés

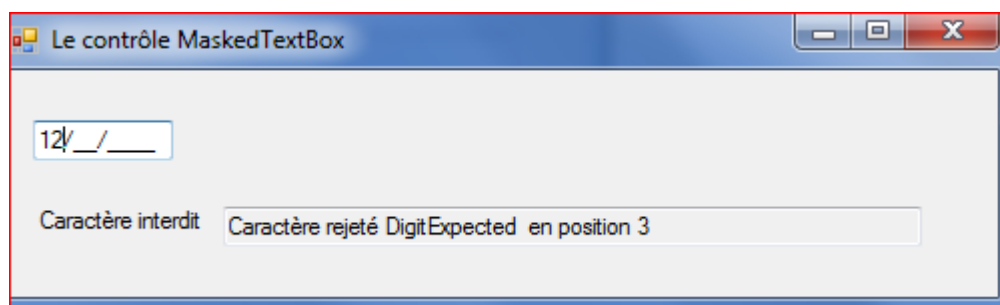
Propriétés	
AsciiOnly	Booléen indiquant si le contrôle MaskedTextBox accepte des caractères en dehors du jeu de caractères ASCII.
CutCopyMakFormat	Obtient ou définit une valeur qui détermine si les littéraux et les caractères d'invite sont copiés
Mask	Obtient ou définit le masque de saisie à utiliser au moment de l'exécution
MaskCompleted	Booléen indiquant si toutes les entrées requises ont été entrées dans le masque de saisie
MaskFull	Booléen indiquant si toutes les entrées requises et facultatives ont été entrées dans le masque de saisie
PromptChar	Caractère utilisé pour représenter un caractère à saisir
RejectInputOnFirstFailure	Booléen indiquant si l'analyse de l'entrée d'utilisateur doit s'arrêter après que le premier caractère non valide est atteint
ResetOnPrompt	Booléen qui détermine comment un caractère d'entrée qui correspond au caractère d'invite doit être géré
ResetOnSpace	Booléen qui détermine comment un caractère d'entrée de type espace doit être géré
SelectedText	Obtient ou définit la sélection en cours dans le contrôle
TextAlign	Obtient ou définit la façon d'aligner le texte dans le contrôle.
TextMaskFormat	Obtient ou définit une valeur qui détermine si les littéraux et les caractères d'invite sont inclus dans la chaîne mise en forme.
ValidatingType	Obtient ou définit le type de données utilisé pour vérifier les données entrées par l'utilisateur.

Evénement de rejet d'un caractère.

La cause de rejet est affichée ainsi que la position du caractère.

```
private void mtbDate_MaskInputRejected(object sender, MaskInputRejectedEventArgs e)
{
    txtCaractereInterdit.Text = string.Format("Caractère rejeté {0} en position {1}",
        e.RejectionHint.ToString(), e.Position.ToString());
}
```

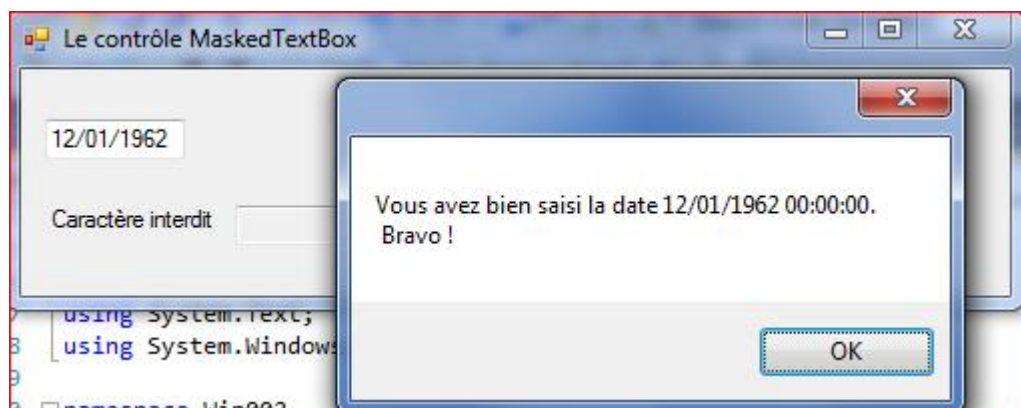
Entrée du caractère **d**



Evénement de validation de la valeur saisie.

```
private void mtbDate_TypeValidationCompleted(object sender, TypeValidationEventArgs e)
{
    if (e.IsValidInput)
    {
        txtCaractereInterdit.Clear();
        MessageBox.Show(string.Format("Vous avez bien saisi la date {0}.\n Bravo !", e.ReturnValue));
    }
    else e.Cancel = true;
}
```

Résultat :



L'événement **TypeValidationCompleted** est déclenché pour exécuter le traitement de la validation de masque ou de type.

Il reçoit un paramètre **TypeValidationEventArgs** qui contient des informations sur la conversion, par exemple, le membre **IsValidInput** qui indique si la conversion a réussi et **ReturnValue** la valeur convertie.

Si la valeur n'est pas correcte, vous pourrez annuler la validation et forcer l'utilisateur à corriger sa saisie **e.Cancel = true**.

Nous reviendrons plus en détails sur cette notion de validation des contrôles.

2.3 Le contrôle d'affichage Label

Un label est une zone d'affichage pouvant être initialisée et modifiée par programme (propriété **Text**), mais qui ne peut être modifiée par l'utilisateur. Souvent associé à un contrôle pour définir son domaine.

Propriétés	
Autosize	La taille du contrôle s'adapte au texte à afficher
Image	Image à afficher
ImageAlign	Alignement de l'image
ImageIndex	Index de l'image sélectionnée dans la liste d'images (contrôle <i>ImageList</i>)
ImageList	Liste d'images à utiliser
Text	Texte affiché dans la zone d'affichage
TextAlign	Cadrage du texte dans la zone d'affichage

2.3.1 Les LinkLabels

Les contrôles **LinkLabel** sont des zones d'affichage, mais qui présentent une caractéristique de bouton : tout le texte du contrôle (propriété **Text**) ou une partie seulement (propriété **LinkArea**) peut servir d'hyperlien.

Le texte peut même comprendre plusieurs hyperliens (propriété **Links** utilisable par programme uniquement).



L'hyperlien peut avoir n'importe quel usage, à programmer dans la fonction de traitement de l'événement **LinkClicked**.

Propriétés	
LinkArea	Portion du texte devant être considéré comme hyperlien
LinkBehavior	Comportement de l'hyperlien (soulignement par défaut ou non)
LinkColor	Couleur d'affichage de l'hyperlien (par défaut bleu)
Links	Collection d'hyperliens définis dans Text , chaque élément étant un objet Link
LinkVisited	Indique si un hyperlien doit être affiché différemment quand il a été visité
VisitedLinkColor	Couleur d'un lien déjà visité

3 Les contrôles UpDown

Les contrôles Up et Down associent des petites flèches à une zone d'édition.

Il est possible d'incrémenter ou de décrémenter la valeur affichée dans la zone d'édition en utilisant les flèches.

	NumericUpDown	Un contrôle qui peut contenir des nombres
	DomainUpDown	Un contrôle qui peut contenir du texte

Propriétés	
DecimalPlaces	Nombre de décimales (0 par défaut)
Hexadecimal	Indique si la partie zone d'édition affiche les valeurs hexadécimales
Increment	Valeur d'incrément ou de décrément (à chaque fois que l'utilisateur clique sur l'une des flèches)
InterceptArrowsKeys	Indique si les touches de direction ↑ et ↓ peuvent être utilisées pour modifier le contenu de la zone d'édition
Maximum	Valeur maximale (100 par défaut)
Minimum	Valeur minimale (0 par défaut)
ThousandsSeparator	Indique si le séparateur des milliers doit être affiché.
UpDownAlign	Position des flèches par rapport à la zone d'édition
Méthodes	
void DownButton()	Simule un clic sur la flèche vers le bas
void UpButton()	Simule un clic sur la flèche vers le haut
Evènements	
ValueChanged()	L'utilisateur a cliqué sur l'une des flèches ou la valeur a été directement modifiée à partir du clavier

La classe **DomainUpDown** permet de sélectionner une chaîne de caractères : elle présente de nombreuses similitudes avec une boîte combo.

Ce contrôle est associé à une collection d'items qui seront les éléments à afficher dans la liste.

Noms	Description
InterceptArrowsKeys	Indique si les touches de direction ↑ et ↓ peuvent être utilisées pour modifier le contenu de la zone d'édition
Items	Collection des éléments présentés
Sorted	Indique si les éléments sont triés
Text	Libellé de l'élément sélectionné
UpDownAlign	Position des flèches par rapport à la zone d'édition
Wrap	Indique s'il y a passage automatique de la dernière valeur à la première
Méthodes	
void DownButton()	Simule un clic sur la flèche vers le bas
void UpButton()	Simule un clic sur la flèche vers le haut
Evènements	
SelectedItemChanged()	L'utilisateur a sélectionné un nouvel élément-

4 Boutons et case à cocher.




La plupart des méthodes et propriétés des classes **Button**, **RadioButton** et **CheckBox** sont héritées de la classe de base **ButtonBase**. Vous trouverez ainsi des méthodes et des propriétés similaires chez ces 3 contrôles.

Le principal événement traité sur ces classes est l'événement **Click**.

Les contrôles **CheckBox** et **RadioButton** possèdent une fonction similaire : ils permettent à l'utilisateur d'effectuer son choix parmi une liste d'options.

Les contrôles **CheckBox** permettent à l'utilisateur de sélectionner une combinaison d'options.

Les contrôles **RadioButton** permettent par contre à l'utilisateur d'effectuer son choix parmi des options s'excluant mutuellement.

 Button	Boutons de commande
 CheckBox	Cases à cocher (CheckBox)
 RadioButton	Boutons radio (RadioButton)

4.1 Le contrôle de commande (Button)

Les contrôles **Button**, sélectionnés par un clic de souris, servent à déclencher une action.

Définissez la propriété **AcceptButton** ou **CancelButton** de la feuille pour permettre aux utilisateurs de sélectionner un bouton en appuyant sur la touche ENTRÉE ou ÉCHAP même si le bouton n'a pas le focus. C'est particulièrement important sur les dialogues de validation de données en mode modal pour accepter ou refuser les modifications.

Propriétés	
ImageList	Référence à une liste d'images
ImageIndex	Numéro d'images dans la liste d'images
FlatStyle	Définit la manière dont les bords des contrôles sont affichés.

4.2 La case à cocher (CheckBox)

La propriété **Appearance** détermine si **CheckBox** s'affiche sous la forme de **CheckBox** traditionnelle ou sous la forme d'un bouton.

La propriété **ThreeState** détermine si le contrôle prend en charge deux ou trois états.

Utilisez la propriété **Checked** pour obtenir ou définir la valeur d'un contrôle **CheckBox** à deux états et utilisez la propriété **Checkstate** pour obtenir ou définir la valeur d'un contrôle **CheckBox** à trois états.

Le troisième état correspond à indéfini.

Les cases à cocher sont en générales regroupées dans un conteneur de type **GroupBox**

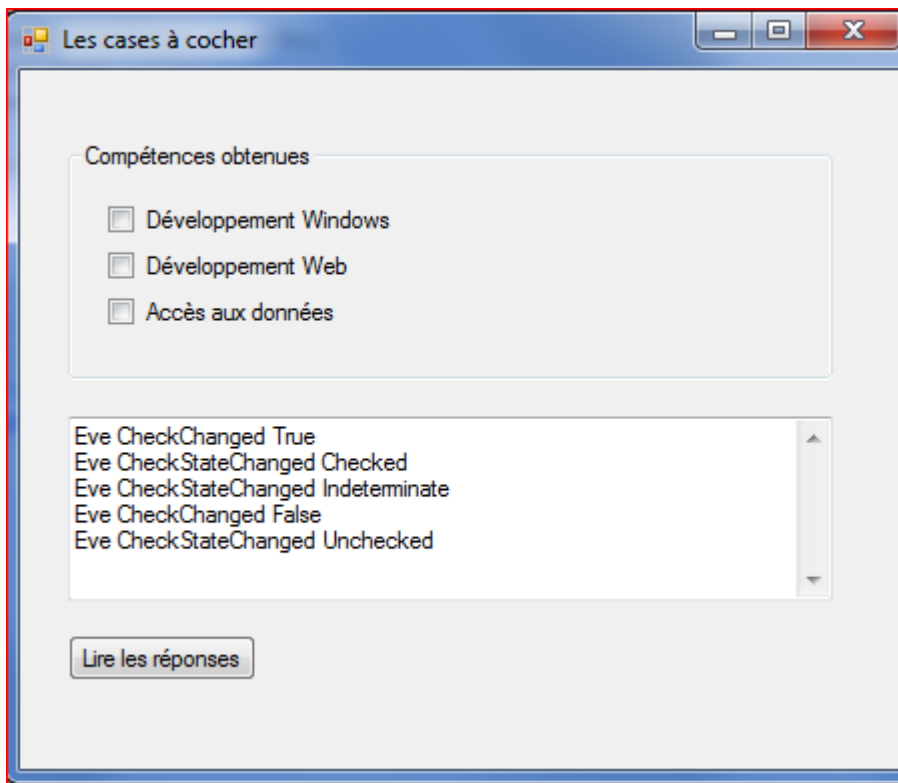
Propriétés	
ThreeState	Indique si la case est à 2 ou 3 états false : 2 états (checked et unchecked) true : 3 états (état supplémentaire : undefined)
Checked	Etat (coché ou non) d'une case à 2 états
CheckedState	Etat d'une case à 3 états
Evénements	
CheckedChanged	Etat d'une case à 2 états a changé
CheckedStateChanged	Etat d'une case à 3 états a changé

Dans l'exemple qui suit, un gestionnaire d'événement a été associé aux événements **CheckedChanged** et **CheckedStateChanged** qui sont similaires mais dont il faut savoir apprécier les différences du fait de cases à cocher à 2 ou 3 états.

```
private void ckDevWindows_CheckedChanged(object sender, EventArgs e)
{
    txtChangements.Text += string.Format("Eve CheckChanged {0} \r\n",
        (sender as CheckBox).Checked.ToString());
}

private void ckDevWindows_CheckStateChanged(object sender, EventArgs e)
{
    txtChangements.Text += string.Format("Eve CheckStateChanged {0} \r\n",
        ((sender as CheckBox).CheckState.ToString());
}
```

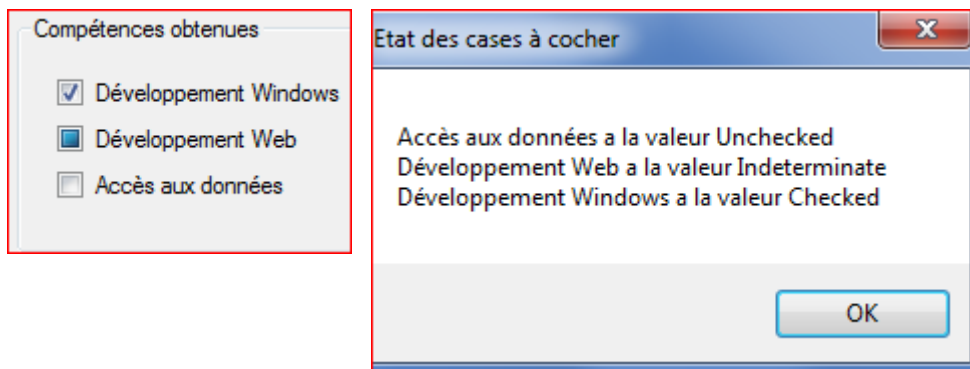
Résultat lorsque nous passons par les 3 valeurs est stocké dans la boîte de texte.



Vous pouvez constater que l'état **Indeterminate** ne déclenche pas l'événement CheckChanged.

Pour lire les réponses, nous pouvons nous appuyer sur nos connaissances en objet et dans la manipulation des collections.

```
string message = string.Empty;
foreach (Control item in this.gbCompétences.Controls)
{
    CheckBox box = item as CheckBox;
    if (box != null)
    {
        message += string.Format("{0} a la valeur {1} \r\n",
            box.Text, box.CheckState);
    }
}
MessageBox.Show(message, "Etat des cases à cocher");
```



4.3 Le Bouton Radio (RadioButton)

Les boutons Radio présentent les mêmes propriétés et méthodes que les cases à cocher, sauf qu'elles ne présentent pas l'état supplémentaire, et ne possèdent donc ni les propriétés ni les événements relatifs à cet état.

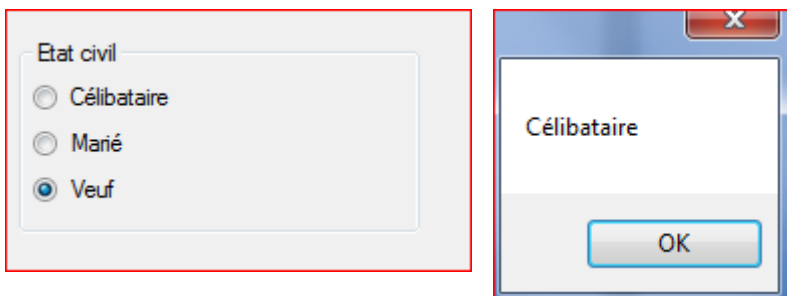
Ces contrôles permettent à l'utilisateur de choisir une option particulière parmi **plusieurs options réunies dans un groupe**.

Il est impossible de sélectionner simultanément plusieurs options dans un même groupe.

Ils doivent être impérativement regroupés sur un contrôle de type **GroupBox**.

Dans l'exemple suivant, nous allons récupérer le texte en fonction du bouton sélectionné lors du changement d'état du bouton.

Nous allons à cette occasion programmer un gestionnaire d'événement commun aux 3 événements `CheckedChanged` des boutons.



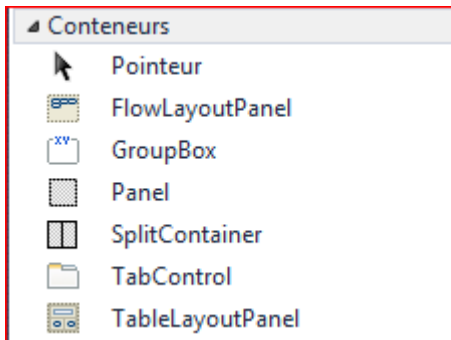
Code de l'exemple.

Vous pouvez observer une fois de plus le recours à l'opérateur **as** pour convertir des types références. Si la conversion échoue, la cible de la conversion est **null**.

```
public FrmRadiosButton()
{
    InitializeComponent();
    this.rbCelibataire.CheckedChanged += new System.EventHandler(this.EtatCivil_Changed);
    this.rbMarie.CheckedChanged += new System.EventHandler(this.EtatCivil_Changed);
    this.rbVeuf.CheckedChanged += new System.EventHandler(this.EtatCivil_Changed);
}

private void EtatCivil_Changed(object sender, EventArgs e)
{
    RadioButton bouton = sender as RadioButton;
    if (bouton != null && bouton.Checked)
    {
        MessageBox.Show(string.Format("{0}", bouton.Text));
    }
}
```

5 Les Conteneurs



Tous les contrôles de type Conteneur ont pour but de manipuler des groupes de contrôles comme un ensemble. Ils incorporent d'autres contrôles.

Ces contrôles incorporés peuvent être aussi des conteneurs.

Ils sont tous regroupés au sein d'un onglet Conteneurs

5.1 Contrôles GroupBox et Panel

Un conteneur du type **Panel** ou **GroupBox** contiendra plusieurs contrôles qui seront fonctionnellement regroupés dans sa collection **Controls**.

Le contrôle **Panel** offre pratiquement les mêmes capacités que le contrôle **GroupBox**. Les différences sont les suivantes :

- Le contrôle **GroupBox** permet d'afficher un titre (propriété *Text*), pas le contrôle **Panel**
- Le contrôle **GroupBox** affiche toujours une bordure, alors que le **Panel** offre une propriété **BorderStyle**

Le contrôle **Panel** dérive également de la classe **ScrollBarControl**.

Il prend donc en charge les barres de défilement de façon automatique si sa propriété **AutoScroll** est à true.

Le **GroupBox** est souvent utilisé pour regrouper des boutons radios ou des cases à cocher ?

Les boutons radios sont exclusifs entre eux dans un même conteneur.

Il est seulement nécessaire de les déposer dans le conteneur pour assurer leur logique de fonctionnement.

5.2 Contrôles FlowLayoutPanel et TableLayoutPanel

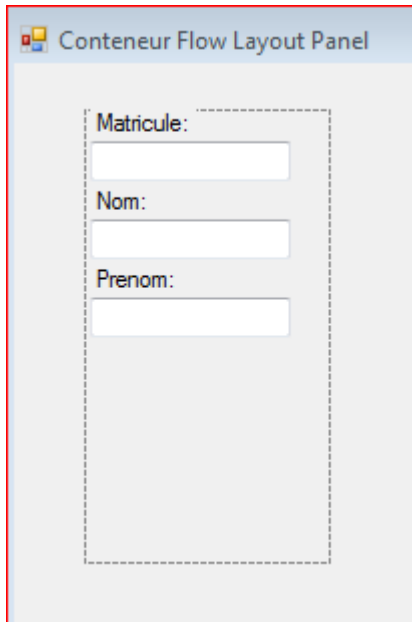
Ces deux contrôles héritent de la classe **Panel**.

Le **FlowLayoutPanel** permet de présenter les contrôles dans le flux à la manière d'un navigateur Web.

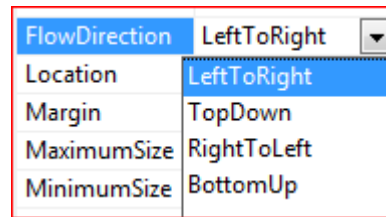
Ils sont disposés les uns à la suite des autres.

Le redimensionnement du contrôle **FlowLayoutPanel** s'accompagne d'une réorganisation des contrôles qu'il contient, afin qu'ils occupent tout l'espace disponible.

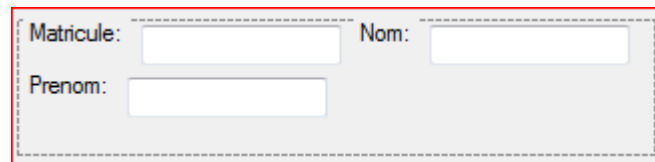
La propriété **FlowDirection** indique la direction dans laquelle les contrôles sont ajoutés au conteneur.



Dans cet exemple, la propriété **FlowDirection** indique que les contrôles seront disposés depuis la gauche vers la droite.



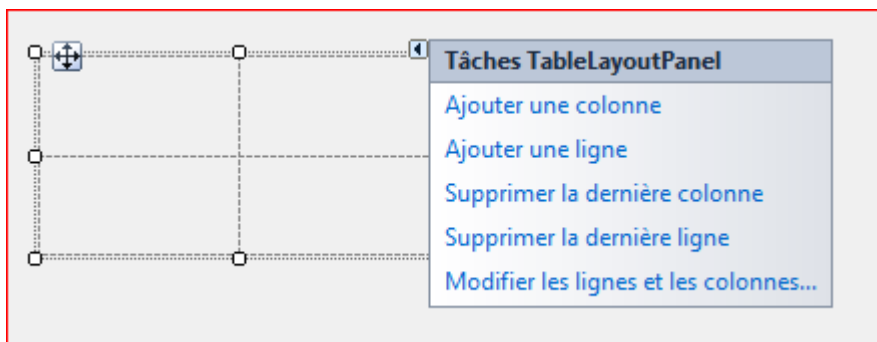
En modifiant la largeur du contrôle nous obtenons la disposition suivante :



Le **TableLayoutPanel** présente les contrôles en les disposant à l'intérieur des cellules d'un tableau, chaque contrôle occupant une cellule.

Le **TableLayoutPanel** est particulièrement utile en cas de création dynamique de contrôle.

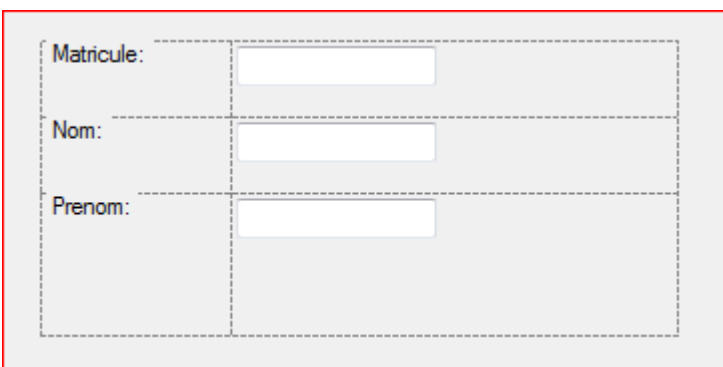
Sa propriété **GrowStyle** permet de spécifier comment seront ajoutés les prochains contrôles si la table est pleine.



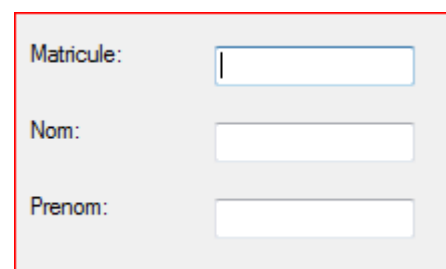
La liste des tâches, accessible via le **smartTag** propose un éditeur pour dimensionner les cellules du tableau

Il donne la possibilité d'ajouter ou supprimer ligne et colonne.

La propriété **CellBorderStyle** permet de spécifier l'apparence des bordures des cellules, un contrôle pouvant s'étendre sur plusieurs lignes ou plusieurs colonnes grâce à ses propriétés supplémentaires, **Rowspan** et **Colspan**.



Le même exemple avec cette fois une présentation sous forme d'une table :



5.3 Contrôle SplitContainer

Utilisez le contrôle **SplitContainer** pour diviser la zone d'affichage d'une feuille et autoriser l'utilisateur à redimensionner les contrôles ajoutés aux panneaux **SplitContainer**.

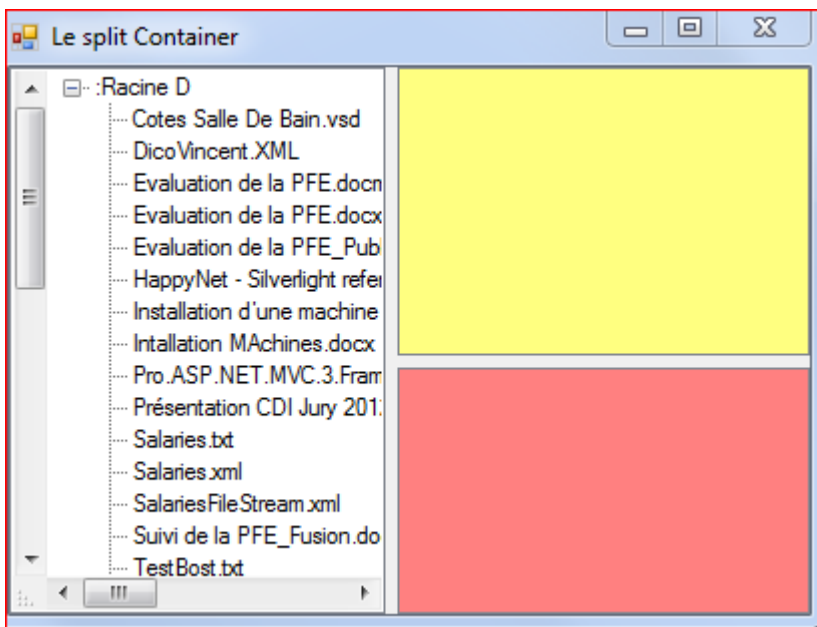
Vous connaissez déjà ce principe avec le gestionnaire de fichiers de windows ou les interfaces de style Outlook. Vous disposez d'un séparateur qui peut être horizontal ou vertical.

Lorsque l'utilisateur passe le pointeur de la souris sur le séparateur, le curseur se modifie pour indiquer que les contrôles situés à l'intérieur du contrôle **SplitContainer** peuvent être redimensionnés.

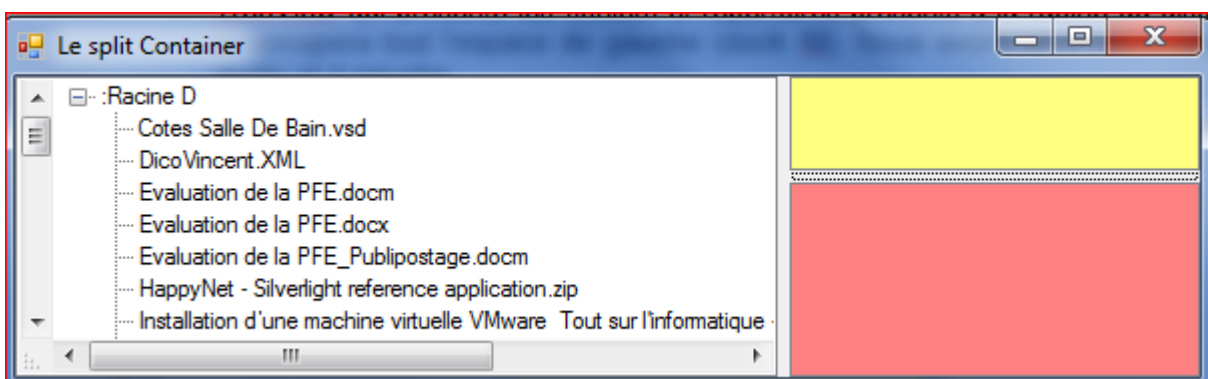
La propriété **SplitterDistance** spécifie l'emplacement du séparateur, en pixels, à partir du bord gauche ou supérieur, alors que **SplitterIncrement** spécifie de combien de pixels le séparateur se déplace à la fois et **Panel1MinSize** et **Panel2MinSize** représentant la taille minimum des 2 panneaux.

Dans l'exemple ci-dessous, j'ai créé deux séparations :

- Une séparation verticale qui permet sur la gauche d'afficher un contrôle de type TreeView qui présente les fichiers et répertoires présents à la racine du disque D. Celui-ci occupera tout l'espace de gauche (dock fill). Nous avons ainsi deux volets (panel) à droite et à gauche
- Le volet de droite contiendra un autre SplitContainer qui permettra de diviser celui-ci en deux nouvelles zones d'affichages avec une séparation horizontale des deux volets.



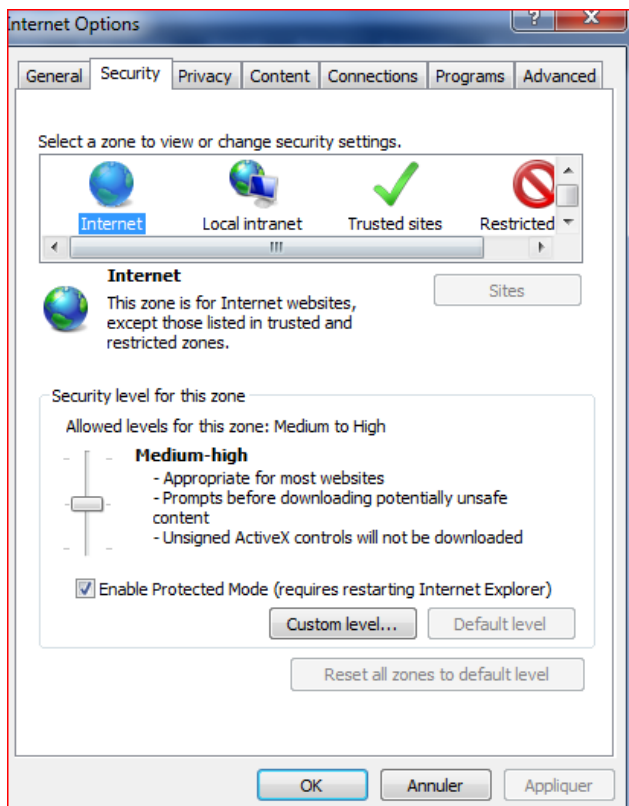
Il sera alors possible d'ajuster la taille des volets gauche/droite et haut/bas de la partie droite.



5.4 Contrôle TabControl

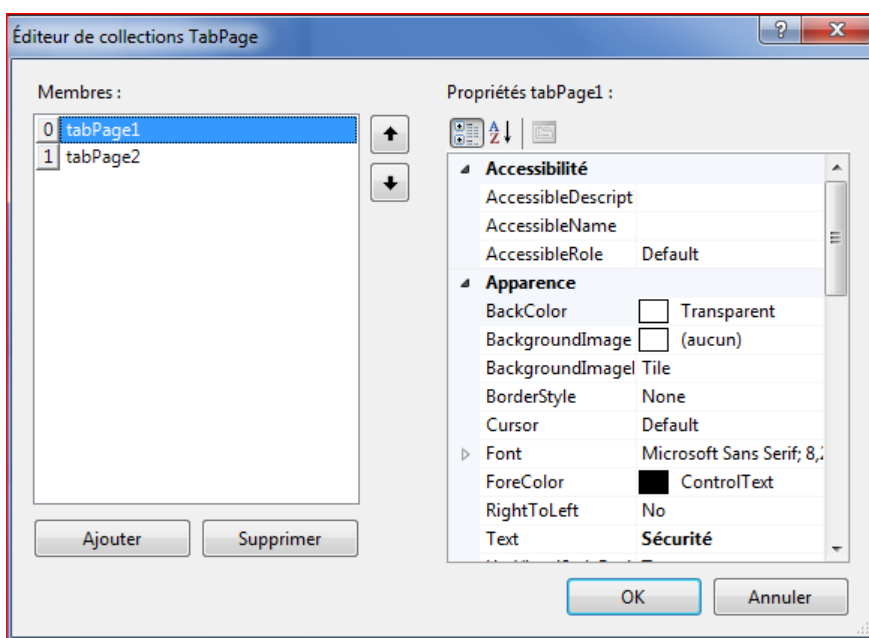
Le contrôle **TabControl** contient des pages d'onglets qui sont représentées par des objets **TabPage**, ajoutés ou supprimés, en conception grâce aux options de la liste des tâches, ou par programme à travers la collection **TabPage**, modifiable directement dans la fenêtre des propriétés, grâce à un éditeur.

Ce contrôle est tout à fait adapté pour présenter un nombre important d'informations qui peuvent être regroupées logiquement mais qui sont indépendantes (non liées) les unes des autres.



Des boutons d'action programmés en dehors des pages d'onglets permettront de valider ou d'invalider l'ensemble des modifications en cours non encore appliquées.

Le paramétrage des options du navigateur Internet illustre bien ce principe.



Lorsque l'onglet actif change, 4 événements surviennent dans cet ordre :

- Deselecting : une demande pour quitter l'onglet courant. En cas d'erreur, il est possible d'annuler la poursuite des événements. C'est à ce moment qu'il est nécessaire de vérifier la cohérence des données.
- Deselected : une page d'onglets vient être désélectionnée
- Selecting : demande de sélection. C'est au cours de cet événement que se font les préparations relatives à l'onglet demandé
- Selected : le nouvel onglet cible de la demande est sélectionné.

Dans tous ces événements vous pouvez avoir accès à des informations sur :

- L'action en cours
- La page d'onglet
- L'index de la page d'onglet

```
private void tabControl1_Deselecting(object sender, TabControlCancelEventArgs e)
{
    // si pas valide alors gestion des erreurs et l'on reste sur l'onglet
    // e.Cancel = true;
}

private void tabControl1_Deselected(object sender, TabControlEventArgs e)
{
    MessageBox.Show(string.Format("Action {0} Index page {1} Page Onglet {2}",
        e.Action,
        e.TabPageIndex,
        e.TabPage));
}

private void tabControl1_Selecting(object sender, TabControlCancelEventArgs e)
{
    MessageBox.Show(string.Format("Action {0} Index page {1} Page Onglet {2}",
        e.Action,
        e.TabPageIndex,
        e.TabPage));
}
```

Exemple lors du passage de la boîte Sécurité à la boîte Réseau

