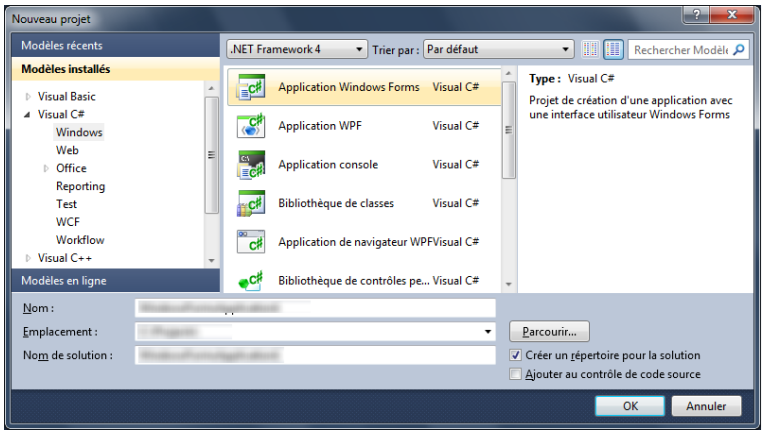
Université de la Rochelle INFO-12605C - IHM
TP7 : Applications fenetrées Windows Forms

© B. Besserer, R. Péteri

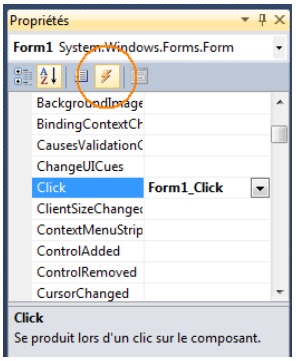
Année universitaire 2015-2016

1 Application fenêtrée *Windows Forms*

Créez un nouveau projet, en sélectionnant application Windows Forms. L'assistant vous créé un projet vide, et vous présente une Form vide qui servira de fenêtre principale. Pour commencer, on va faire simple : vous allez implémenter la fonctionnalité suivante : Un clic dans le fond de la fenêtre va quitter l'application.



Si vous double-cliquez dans la Form, l'assistant vous propose d'écrire une méthode qui sera appelée lors du chargement (load) de la Form. Pour écrire une méthode qui réagira **au clic dans la fenêtre**, vous devez visualiser les événements (même fenêtre que les propriétés, un bouton avec un éclair qui vous permet de visualiser les événements) et sélectionner le clic souris (voir copie d'écran). Vous trouverez rapidement la manipulation qui permettra à l'assistant de générer le squelette votre méthode qui sera attachée à la méthode déléguée (nommée par l'assistant `private void Form1_Click(object sender, EventArgs e))`



L'appel de `this.Close();` ferme la Form et permet de quitter l'application, néanmoins cela ne marche que si votre application n'utilise qu'une seule fenêtre. Sinon, `Application.Exit();` fera l'affaire. Testez le fonctionnement. Ensuite, modifiez les **propriétés** de la Form pour :

- Changez le titre,
- Attribuer à la fenêtre une taille minimale lors d'un redimensionnement,
- Démarrez l'application en mode plein écran,
- Changez la transparence.

2 Création de contrôle IHM par programme

Il est facile peupler la Form en effectuant un glisser-déposer en prélevant un contrôle depuis la barre d'outils (si non visible, faire CTRL+ALT+X). Il est tout aussi facile d'ajouter un contrôle dynamiquement, lors de l'exécution du code. Par exemple, pour créer, afficher et gérer un bouton simple (pushbutton), il faut :

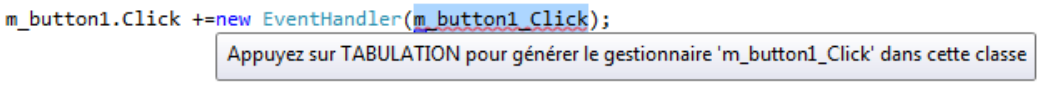
- déclarer, comme donnée membre de la classe Form, une instance de la classe `Button`, que vous nommerez `m_button1`. Cette instance sera créée avec le mot clé `new Button`
- Dans un bloc de code qui n'est exécuté qu'une seule fois (Réfléchissez...), on va définir, par des instructions appropriés, les propriétés de l'instance `m_button1`. Les propriétés minimales à définir sont la taille, l'emplacement et le label. On y ajoutera une couleur pour le fond afin de bien voir ce bouton.

Par exemple, pour le label, on écrira `this.m_button1.Text = "Exit";`
Il faut modifier la propriété `this.m_button1.BackColor` (utilisez les valeurs disponibles dans l'énumération `Color`), la propriété `this.m_button1.Location` (on créera un objet de type `System.Drawing.Point` pour cela), et `this.m_button1.Size` (on créera un objet de type `System.Drawing.Size` pour cela). Enfin, `this.Controls.Add(m_button1);` permet d'ajouter ce nouveau bouton à la hiérarchie des contrôles de la Form. Vous remarquerez que nous n'utilisons pas de gestionnaire de géométrie.
Testez la création d'un bouton poussoir.
Il est tout aussi facile d'attacher une méthode aux différents *delegates* définies pour les événements possibles générés par le bouton. L'assistant fait tout le travail pour vous. Saisissez la ligne suivante :

```
m_button1.Click +=
```

et vous verrez une fenêtre proposant la complétion lors de l'appui sur la touche tabulation :

Appuyer sur Tab et la ligne est complétée, l'assistant vous demande alors s'il faut générer le squelette de la fonction déléguée :



encore une fois, appuyer sur Tab, est le squelette de code est alors créé :

```
void m_button1_Click(object sender, EventArgs e)
{
   throw new NotImplementedException();
}
```

Modifiez votre programme pour que maintenant l'application soit quitté lors du clic sur ce bouton.

Corrigé :

```
private void Form1_Load(object sender, EventArgs e)
{
    //Format controls. Note: Controls inherit color from parent form.
    this.m_button1.BackColor = Color.Gray;
    this.m_button1.Text = "Clear";
    this.m_button1.Location = new System.Drawing.Point(90, 25);
    this.m_button1.Size = new System.Drawing.Size(50, 25);

    this.Controls.Add(m_button1);

    m_button1.Click += new EventHandler(m_button1_Click);

}

void m_button1_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```

3 Tracé graphique

Ajoutez un second bouton. Le premier bouton (renommé Paint) servira maintenant à activer le dessin d'une figure géométrique sur le fond de votre fenêtre, le second bouton (nommé Clear) permettra d'effacer cette figure. **Attention, il faut que la figure tracé reste visible lorsque l'application est mise en icone puis restaurée.** On ne placera donc pas les instructions de dessin n'importe où.

En fait, au niveau du code, une action sur le bouton Paint validera simplement une variable membre booléenne `m_dopaint`. Si cette variable est vraie, alors dessin devra s'afficher si la fenêtre doit "se rafraîchir". Si cette variable est fausse, aucun dessin ne se produit.
Copiez le code ci-dessous au bon endroit, l'exécution de ce code étant conditionné par la variable `m_dopaint`.
On peut forcer un "rafraîchissement" de la fenêtre avec `Invalidate();` - cherchez sur internet des informations concernant cette dernière méthode.

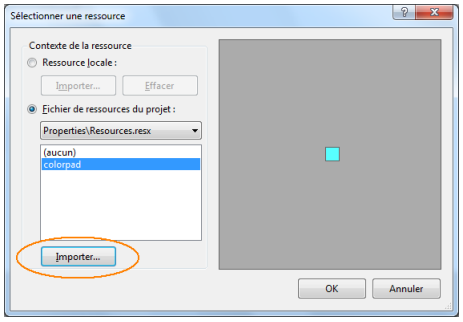
```
Rectangle textRect = new Rectangle();
StringFormat format = new StringFormat();
format.LineAlignment = StringAlignment.Center;
format.Alignment = StringAlignment.Center;

e.Graphics.DrawRectangle(drawPen, new Rectangle(50, 50, 200, 200));
e.Graphics.DrawLine(drawPen, 150, 50, 150, 250);
e.Graphics.DrawLine(drawPen, 50, 150, 250, 150);
textRect.Location = new Point(50, 50);
textRect.Size = new Size(100,100);
char letter = '1';
e.Graphics.DrawString(letter.ToString(), DefaultFont, Brushes.Red, textRect, format);
```

Lorsque le programme sera fonctionnel (un bouton pour le dessin, l'autre pour effacer), faites valider.  1

4 Utilisation de la barre de menu

Ajouter une barre de menu, à prélever dans la barre d'outil (une catégorie est dédiée à cela). Le design de la barre de menu est assez facile. Vous devez réaliser une barre de menu qui comprendra 3 entrées : "Fichier", "Dessin" et "Aide". L'entrée Fichier disposera d'un article nommé "Open...", et l'entrée "Dessin" d'un article nommé "Dessin", d'un article nommé "Couleur" et d'un article nommé "dernier caractère :"
En effectuant les réglages adéquats avec les propriétés :

- Le menu d'aide doit se trouver à droite
- L'article de menu "Dessin" doit disposer d'une boite à cocher (dans les propriétés). La propriété CheckOnClick doit aussi être à true.
- L'article de menu "Couleur" doit disposer d'une icone ou image. Pour cela :
  - Utiliser PAINT ou un autre utilitaire pour dessiner un carré de 16 x 16 pixels, en noir, que vous sauvegarderez comme fichier BMP (par exemple colorpad.bmp) dans le repertoire de votre projet.
  - Dans les proprieté de l'article de menu "Couleur", cherchez Image. Cliquez sur le bouton pour choisir une image, et vous obtiendrez une boite de dialogue comme ci-dessous. Activez le bouton radio correspondant aux ressources, puis importez (bouton Import) l'image colorpad.bmp comme ressource, et enfin sélectionnez cette image.

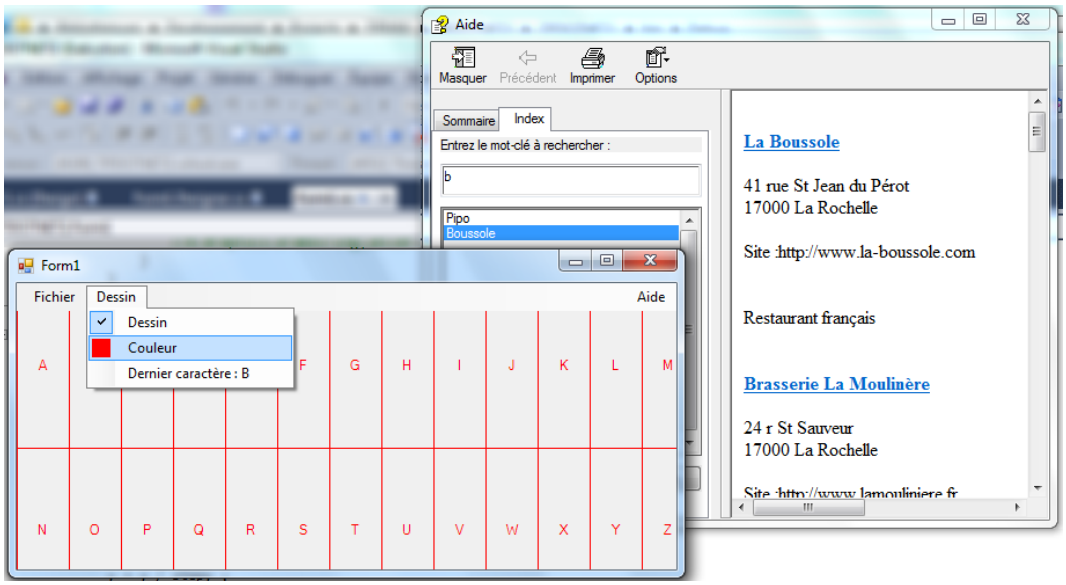


Modifiez votre code pour que les opérations de dessin puisse s'effectuer lors de l'action sur l'article de menu "Dessin". On vous demande de **supprimer** la variable `m_dopaint` et de la remplacer par l'interrogation de l'état checked de l'article de la barre de menu.
En fait, la fonction déléguée qui sera appelée lorsqu'on sélectionne l'article de menu "Dessin" ne contient qu'une seule instruction....  2

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
  if (dessinToolStripMenuItem.Checked == true)
  {
    // code de dessin a mettre ici
  }
}

private void dessinToolStripMenuItem_Click(object sender, EventArgs e)
{
  Invalidate();
}
```

5 Application avec tracé graphique

Les fichiers d'aide de Windows (format `.HLP` sous 3.1, win95 et NT, puis format `.CHM`) peuvent être créés avec une table d'index. Les bibliothèques win32, MFC et bien évidemment .NET disposent de fonctions ou de méthodes permettant d'ouvrir un tel fichier d'aide et d'aller directement sur la page dont on spécifie la clé d'index. En C# et .NET, la classe `Help` dispose de la méthode `ShowHelp()`
Un fichier d'aide, `realworld.chm` est disponible au téléchargement sous Moodle ; il possède un mot clé (*Keyword*) pour (pratiquement) chaque lettre individuelle, ainsi qu'une page pour chaque lettre. Idéalement, copiez ce fichier dans le répertoire ou se trouve l'exécutable que vous générez.



L'application que vous devez écrire dessine sur la Form une grille (dans une double boucle (lignes, colonnes) et tracée avec des primitives graphiques genre DrawLine et non pas Rectangle) et les case de cette grille comporte les différentes lettres de l'alphabet (soit un tableau de 13 x 2 cases, voir copie d'écran). **Important :** Pour afficher ces 26 lettres de l'alphabet on utilisera dans la boucle une valeur d'offset par rapport au code ASCII de la première lettre 'A'. Pour tracer le caractère, on utilisera DrawString. Cette grille doit être **redimensionné dynamiquement** lors de la modification de géométrie de la fenêtre, avec un placement correct des lettres. On interdira à l'utilisateur de rétrécir la fenêtre en deçà d'un taille minimale (facile à faire avec les *properties* de la Form). Le code donnée dans la section 3 peut vous être utile. Le tracé sera fait dans la méthode attachée à l'événement Paint.

Elements de correction : *Code a placer dans la méthode déléguée*
`Form1_Paint(object sender, PaintEventArgs e)`

```
StringFormat format = new StringFormat();
format.LineAlignment = StringAlignment.Center;
format.Alignment = StringAlignment.Center;

stepx = this.Size.Width / 13;
stepy = this.Size.Height / 2;

int ascii_for_A = (int)'A';

using (Pen drawPen = new Pen(m_DrawColor))
using (SolidBrush drawBrush = new SolidBrush(m_DrawColor))
{
    for (j = 0; j < this.Size.Height; j += stepy)  // verticales
    {
        e.Graphics.DrawLine(drawPen, 0, j, this.Size.Width, j);
        for (i = 0; i < this.Size.Width - stepx; i += stepx) // horizontales
        {
            e.Graphics.DrawLine(drawPen, i, 0, i, this.Size.Height);
            textRect.Location = new Point(i, j);
            textRect.Size = new Size(stepx, stepy);
            char test = Convert.ToChar(ascii_for_A++);
            string toto = test.ToString();
            e.Graphics.DrawString(toto, DefaultFont, drawBrush, textRect, format);
        }
    }
}
```

Ensuite, un clic (normal, bouton gauche) dans cette grille doit ouvrir ce fichier "d'aide contextuelle", c'est à dire que le fichier d'aide doit s'ouvrir et se positionner sur une page d'aide correspondant au mot clé passé en paramètre : En cliquant dans la case 'C', le fichier d'aide doit se positionner sur la page correspondant à la lettre C. Une des difficultés est bien sûr de déterminer la lettre correspondant à l'emplacement du clic...Il faut donc effectuer des tests sur les coordonnées du clic.
Pour vérifier le bon fonctionnement, une fois les coordonnées du clic décodés et attribués à un caractère (disons X), affichez ce caractère dans l'article de menu correspondant, qui doit alors afficher "dernier caractère : X".

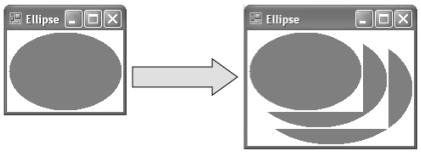
Cette affichage doit être réactualisé pour chaque clic. Vérifiez si, lors de 2 clics successifs, les caractères ne se cumulent pas dans l'inscription de la barre de menu.

Elements de correction : *Verifier si, lors de 2 clics successifs, les caractères ne se cumulent pas dans l'inscription de la barre de menu.*

```
char car = 'A';  // en réalité le caractère déterminé par le clic
dernierCaractereToolStripMenuItem.Text =
   dernierCaractereToolStripMenuItem.Text.Remove(dernierCaractereToolStripMenuItem.Text.Length-1) + car;
```

Enfin, le fichier d'aide doit aussi s'ouvrir et s'afficher lors d'un clic sur l'article Help du menu, et également lors d'un appui sur la touche de fonction F1 (événement de type KeyDown).
Vérifiez que, lorsqu'on ferme l'application, le fichier d'aide doit se ferme également.

Si vous avez des problèmes de tracé graphique lors des redimensionnements, et que le comportement ressemble à celui illustré ci-contre (seule la portion agrandie est redessinée), il faut ajouter la ligne `this.ResizeRedraw = true;` dans le constructeur de la Form.



3

5.1 Version avec gestionnaire de géométrie

Mettez en commentaire le code de dessin de votre application. Placer sur le fond de la Form un contrôle de type TableLayoutPanel.Pour cette objet, mettre le nombre de colonnes (ColumnCount) à 13. Dans le code, au **chargement de la Form**, créer par programme 26 boutons qui seront mis dans les cases de ce gestionnaire de géométrie. Remarquez par de quelle façon on associe un *callback* unique à chaque bouton.

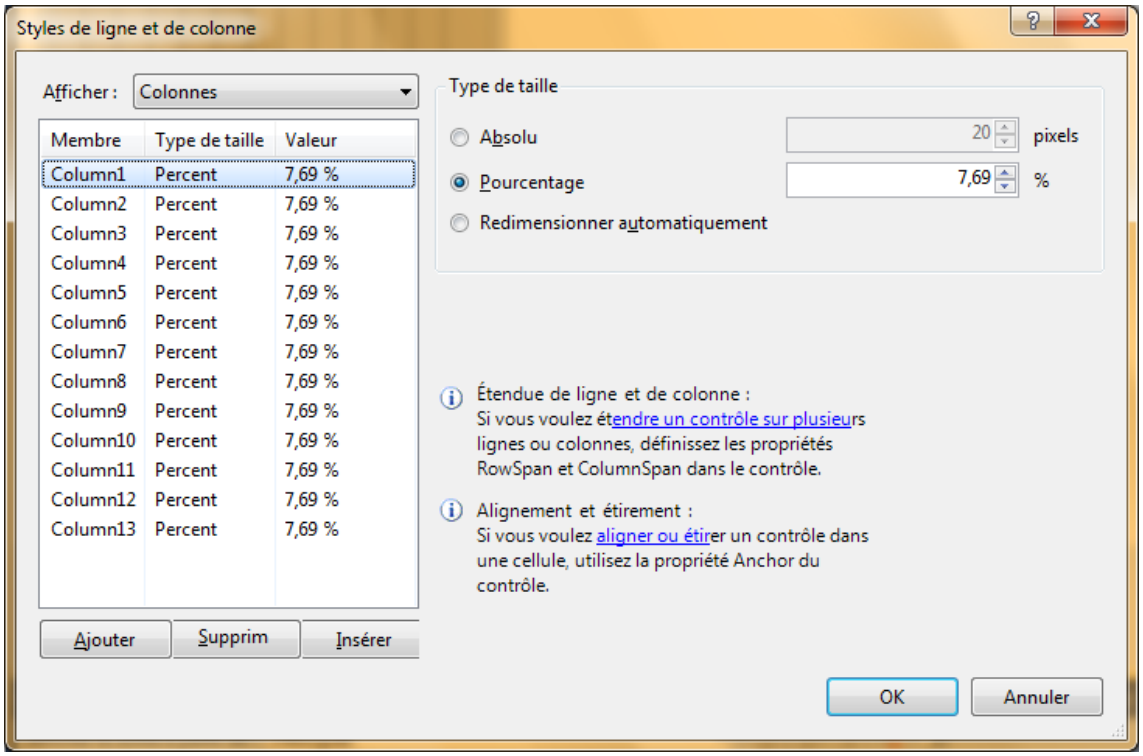
```
private void Form1_Load(object sender, EventArgs e)
{
    for (int j = 0; j < 2; j++)
    {
        for (int i = 0; i < 13; i++)
        {
            Button b = new Button();
            char character = 'A';
            character = (char)((int)character + (i + (13 * j)));
            b.Text = character.ToString();
            b.Click += new EventHandler(b_Click);
            this.tableLayoutPanel1.Controls.Add(b, i, j);
}   }   }
void b_Click(object sender, EventArgs e)
{
    Button b = sender as Button;
    // recupération facile de la valeur de b.Text et traitement en fonction du caractere
}
```

Cherchez (dans les *properties* des contrôles et sur internet) le moyen d'implémenter les fonctionnalités :
- Dimension du bouton = celle de la case du tableau
- Dimension du tableau = dimension de la fenêtre, avec une taille minimale.

Indication : Dans la vue designer, clic-droit dans le tableau et choisir "modifier les lignes et les colonnes"...

Elements de correction : *Pour que le contrôle prenne la taille du parent, il faut mettre la property Dock à la valeur DockStyle.Fill*

Dans la vue designer, il faut sélectionner le contrôle TableLayoutControl, mettre Dock à Fill, puis faire clic-droit dans le tableau et choisir "modifier les lignes et les colonnes", et entrer les valeurs en pourcentages (on peut selectionner toutes les colonnes ensemble)




```
private void Form1_Load(object sender, EventArgs e)
 {
     for (int j = 0; j < 2; j++)
     {
         for (int i = 0; i < 13; i++)
         {
             Button b = new Button();
             char character = 'A';
             character = (char)((int)character + (i + (13 * j)));
             b.Text = character.ToString();
             b.Dock = DockStyle.Fill;
             b.FlatStyle = FlatStyle.Flat;
             b.Click += new EventHandler(b_Click);
             this.tableLayoutPanel1.Controls.Add(b, i, j);
         }
     }
 }

 void b_Click(object sender, EventArgs e)
 {
     Button b = sender as Button;
     // recupération facile de la valeur de b.Text
 }
```

5.2 Choix de la couleur du tracé

Déclarez une bitmap en tant que donnée membre et chargez la ressource dans cette bitmap (une seule fois, au lancement de l'application) :

```
Bitmap m_ColorImage;
m_ColorImage = new Bitmap(TPDOTNET2.Properties.Resources.colorpad);
```

Ajoutez une boite de dialogue de **sélection de couleur**. La boite de dialogue est disponible dans la barre d'outil, vous faites un Drag & Drop sur la fenêtre de l'application. Ouvrez le dialogue de sélection de couleur lors du clic sur l'article de menu "Couleur" (`colorDialog1.ShowDialog()`). Lorsque la couleur est sélectionnée, récupérez cette couleur (qui sera stockée dans une variable membre `Color m_DrawColor;` puis remplissez le bitmap de cette couleur :

```
using (Graphics gfx = Graphics.FromImage(m_ColorImage))
using (SolidBrush brush = new SolidBrush(m_DrawColor))
{
  gfx.FillRectangle(brush, 0, 0, m_ColorImage.Width, m_ColorImage.Height);
}
```

et enfin affectez cette image modifiée à l'article de menu :

```
couleurToolStripMenuItem.Image = m_ColorImage;
```

Modifiez votre code pour que le tracé (ou l'affichage des boutons) soit effectué dans la couleur choisie.  4