

Les images en .NET

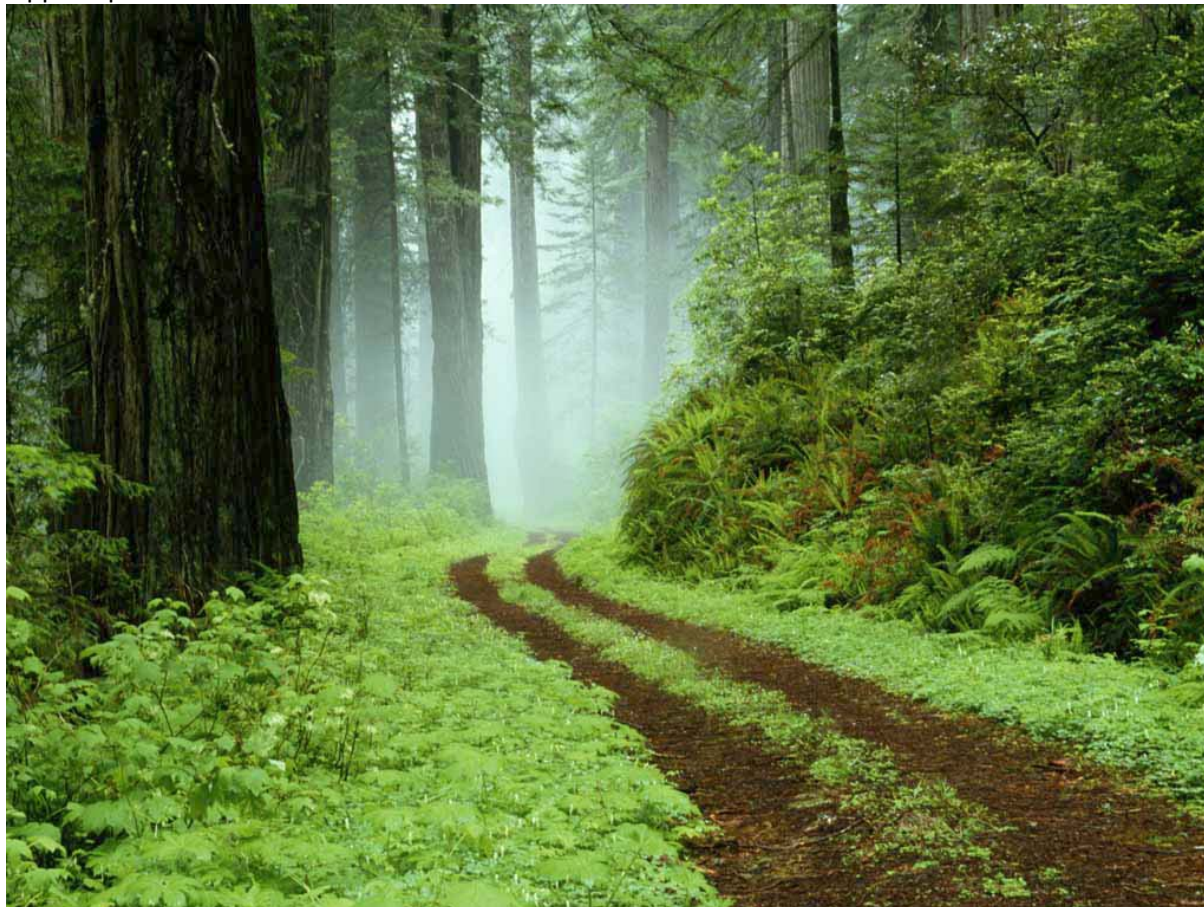
Sommaire

Les images en .NET	1
1 Introduction.....	2
2 Les outils graphiques	3
3 Les images dans le .NET Framework	5
4 Dessiner des formes	9
4.1 La classe Graphics.....	9
4.2 La classe Pen.....	13
4.3 L'outil Brush.....	14
4.4 Dessiner du texte.....	17
4.5 Formater le texte.....	18
5 Conclusion	22

Dotnet-France Association

1 Introduction

Jusqu'ici nous travaillions sur des données formatées dans un type spécial et n'ayant de rapport qu'avec les textes.



Dans cette partie, nous allons nous intéresser à un type de données particulier : les données binaires représentant des images.

Dans un premier temps, nous verrons comment sont stockées/maniées les images dans le .NET Framework, puis nous verrons comment remplir ces images avec des formes simples de couleurs et possédant un style particulier.

Important : Etant donné que les images ne peuvent plus être affichées dans une console, nous utiliserons un projet Winforms. Si vous n'êtes pas habitué avec ce type de projet, nous vous recommandons de lire les tutoriels concernant les Winforms disponible sur Dotnet-France.

2 Les outils graphiques

L'espace de nom `System.Drawing` met à notre disposition un ensemble de classes et de structures nécessaires à la création de dessins. Vous allez pouvoir ainsi dessiner des formes plus ou moins complexes ou manipuler des images. La plupart de ces classes font partie de l'API GDI+ (Graphics Device Interface, une API de gestion d'outils graphique pour Windows).

Voici un tableau présentant les classes les plus utilisés de l'espace de nom :

Classe	Description
<code>Bitmap</code>	Permet de manipuler des images. L'objet <code>Bitmap</code> permet d'encapsuler une image bitmap GDI+. Cela comprend des informations sur les pixels, et les attributs de l'image. La classe <code>Bitmap</code> gère les formats d'images suivant BMP, GIF, EXIF, JPG, PNG et TIFF.
<code>Brush</code>	Classe abstraite, elle ne peut pas être instanciée. <code>Brush</code> définit les comportements de bases lorsqu'il s'agit de remplir l'intérieur de formes géométriques, par une couleur ou une texture par exemple.
<code>Brushes</code>	Permet grâce à des propriétés statiques en lecture seule de retourner des <code>Brush</code> de couleur. Par exemple : <code>Brushes.Green</code> retourne un <code>Brush</code> de couleur verte.
<code>ColorConverter</code>	Permet de convertir une couleur dans un autre type de donnée (par exemple en string). Vous devez utiliser la méthode <code>GetConverter</code> de <code>TypeDescriptor</code> pour accéder aux membres de cette classe.
<code>ColorTranslator</code>	Permet de convertir une couleur GDI+ en un autre type (HTML, OLE, Win32) et vice versa.
<code>Font</code>	Permet de formater un texte. Notamment la taille, la police, et les styles du texte (gras, italique...)
<code>FontConverter</code>	Permet de déterminer une police d'écriture à partir d'une chaîne de caractères (Par exemple, analyser "Comic Sans MS, 15pt" créera une police basée sur Comic Sans MS et de taille 15).
<code>FontFamily</code>	Permet de créer des groupes de <code>Font</code> dont le design de base est identique mais avec des styles différents.
<code>Graphics</code>	Permet de dessiner des formes géométriques. L'objet <code>Graphics</code> permet à GDI+ d'encapsuler une surface à dessiner.
<code>Icon</code>	Permet de définir une icône. Les icônes peuvent être des bitmaps transparents, mais leur taille est définie par le système.
<code>IconConverter</code>	Permet de convertir un objet <code>Icon</code> dans un autre type de donnée (par exemple en string). Vous devez utiliser la méthode <code>GetConverter</code> de <code>TypeDescriptor</code> pour accéder aux membres de cette classe.
<code>Image</code>	Classe abstraite qui définit les comportements de bases de <code>Bitmap</code> et <code>Metafile</code>
<code>ImageAnimator</code>	Anime une image en utilisant le principe de frames (Par exemple un GIF animé).
<code>ImageConverter</code>	Permet de convertir un objet <code>Image</code> dans un autre type de donnée (par exemple en string). Vous devez utiliser la méthode <code>GetConverter</code> de <code>TypeDescriptor</code> pour accéder aux membres de cette classe.
<code>ImageFormatConverter</code>	Permet de convertir une couleur dans un autre type de donnée (par exemple en string). Vous devez utiliser la méthode <code>GetConverter</code> de <code>TypeDescriptor</code> pour accéder aux membres de cette classe.
<code>Pen</code>	Un objet <code>Pen</code> permet de dessiner des formes géométriques

Pens	Permet grâce à des propriétés statiques en lecture seule de retourner des Pen de couleur. Par exemple : <code>Pens.Green</code> retourne un Pen de couleur verte.
PointConverter	Permet de convertir un objet Point dans un autre type de donnée (par exemple en string). Vous devez utiliser la méthode <code>GetConverter</code> de <code>TypeDescriptor</code> pour accéder aux membres de cette classe.
RectangleConverter	Permet de convertir un objet Rectangle dans un autre type de donnée (par exemple en string). Vous devez utiliser la méthode <code>GetConverter</code> de <code>TypeDescriptor</code> pour accéder aux membres de cette classe.
Region	Permet de découper une forme graphique en zone auquel on peut associer un contrôle.
SizeConverter	Permet de convertir une taille dans un autre type de donnée (par exemple en string). Vous devez utiliser la méthode <code>GetConverter</code> de <code>TypeDescriptor</code> pour accéder aux membres de cette classe.
StringFormat	Encapsule un texte avec ses informations de mise en page, d'affichage et les fonctionnalités OpenType.
SystemBrushes	Contient plusieurs propriétés qui permettent de récupérer un SolidBrush de la couleur d'un élément de la fenêtre (par exemple ButtonShadow qui est la couleur de l'ombre d'un bouton).
SystemColors	Contient plusieurs propriétés qui permettent de récupérer des structures Colors de la couleur d'un élément de la fenêtre.
SystemFonts	Permet de spécifier la Font permettant d'écrire du texte dans les éléments de la fenêtre.
SystemIcons	Contient plusieurs propriétés qui permettent de récupérer un objet Icon représentant des icônes du system Windows.
SystemPens	Contient plusieurs propriétés qui permettent de récupérer un Pen de la couleur d'un élément de la fenêtre et de largeur 1.
[ToolboxBitmap()]	Vous pouvez appliquer cet attribut à un contrôle afin d'associer un bitmap à ce contrôle dans la boîte à outils. L'icône doit être de 16x16 pixels par défaut.

Ce second tableau présente les principales structures de l'espace de nom `System.Drawing`:

Structure	Description
CharacterRange	Permet de spécifier une plage de caractères dans une chaîne, afin par exemple de mettre en surbrillance un mot d'une chaîne.
Color	Représente une couleur. La méthode <code>FromArgb</code> permet de créer une couleur à partir de plusieurs octets.
Point	Définit un point dans un plan 2D par une paire d'entiers X et Y.
PointF	Définit un point dans un plan 2D par une paire de flottants X et Y.
Rectangle	Permet de définir un rectangle grâce à quatre entiers. Deux représente la position par rapport à l'origine, et deux la taille du rectangle.
RectangleF	Permet de définir un rectangle grâce à quatre flottants. Deux représente la position par rapport à l'origine, et deux la taille du rectangle.
Size	Permet de définir une taille grâce à deux entiers représentant la hauteur et la largeur.
SizeF	Permet de définir une taille grâce à deux flottants représentant la hauteur et la largeur.

3 Les images dans le .NET Framework

La classe `Image` est une classe abstraite qui permet le chargement, la modification et la sauvegarde d'images aux formats Bitmap (BMP), Jpeg (JPG), Tagged Image File (TIFF), Portable Network Graphics (PNG) ...

Les principales méthodes et propriétés de cette classe sont reprises ci-dessous :

Membre	Description
<code>FromFile</code>	Retourne un objet <code>Image</code> , chargé à partir d'un fichier image.
<code>FromStream</code>	Retourne un objet <code>Image</code> , chargé à partir d'un flux de données.
<code>FromHbitmap</code>	Retourne un objet <code>Bitmap</code> à partir d'un gestionnaire de fenêtre.
<code>GetBounds</code>	Retourne les limites maximales de l'image.
<code>GetPixelFormatSize</code>	Retourne le nombre de bits par pixels utilisé dans l'image.
<code>Save</code>	Sauvegarde les données de l'image dans le fichier spécifié.
<code>Height</code>	Indique la hauteur de l'image en nombre de pixel.
<code>PixelFormat</code>	Obtient le format de pixel utilisé dans l'image.
<code>Size</code>	Obtient un objet <code>Size</code> indiquant la largeur et la hauteur de l'image.
<code>Width</code>	Indique la largeur de l'image en nombre de pixel.
<code>HorizontalResolution</code>	Indique la résolution horizontale de l'image en pixels par pouces.
<code>VerticalResolution</code>	Indique la résolution verticale de l'image en pixels par pouces.

Il y a également les classes `Bitmap` (pour dessiner dans une image) et `Metafile` (pour les images animées) qui héritent toutes les deux de la classe `Image`.

Du fait de sa simplicité, la classe `Bitmap` est la plus utilisée. Elle peut être instanciée aussi bien en partant d'un fichier qu'en partant d'un objet `Image` directement. Elle propose les méthodes `SetPixel` et `GetPixel` qui permettent respectivement de définir un pixel et d'obtenir un pixel de l'image.

Pour afficher une image, nous allons redéfinir le constructeur de notre Winform. Dans ce nouveau constructeur, nous chargeons une image que nous définissons comme fond de notre fenêtre. Sur cette même image, nous dessinons une suite de pixels et nous enregistrons l'image finale dans un fichier (Plus bas, nous verrons comment dessiner des formes de façon simple) :

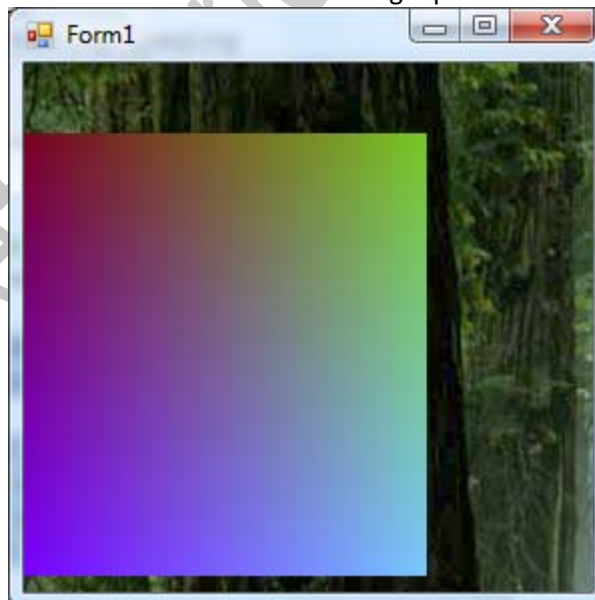
```
'VB
Sub New()
    InitializeComponent()

    Dim image = New Bitmap("forest.jpg")
    Me.BackgroundImage = image
    For j As Integer = 35 To 255 Step 1
        For i As Integer = 0 To 200 Step 1
            image.SetPixel(i, j, Color.FromArgb(255, 120, i, j))
        Next
    Next
    image.Save("pict.bmp", ImageFormat.Bmp)
End Sub

//C#
public Partiel()
{
    InitializeComponent();

    Bitmap image = new Bitmap("forest.jpg");
    this.BackgroundImage = image;
    for (int j = 35 ; j < 255; j++)
        for(int i = 0; i < 200; i++)
            image.SetPixel(i, j, Color.FromArgb(255,120,i,j));
    image.Save("pict.bmp", ImageFormat.Bmp);
}
```

A la compilation, nous avons une fenêtre avec l'image qui est affichée en fond :













Le .NET Framework peut également gérer des images bitmap aux spécificités particulières : Les icônes. En effet, les icônes sont de taille réduite et possède un canal de transparence.

Ces icônes sont représentées par la classe `Icon` dont voici les principaux membres:

Membre	Description
<code>ExtractAssociatedIcon</code>	Retourne une icône basée sur les données de l'image passée en paramètre.
<code>Save</code>	Sauvegarde l'icône dans un fichier.
<code>ToBitmap</code>	Convertit l'icône en image Bitmap.
<code>Height</code>	Retourne la hauteur de l'icône.
<code>Size</code>	Retourne les dimensions de l'icône.
<code>Width</code>	Indique la largeur de l'icône.

Le constructeur de cette classe permet de charger un fichier icône.

Le Framework propose quelques icônes que vous avez sûrement déjà vu lors de vos utilisations de produits Microsoft. Ces icônes sont stockées dans l'énumération `SystemIcons` :

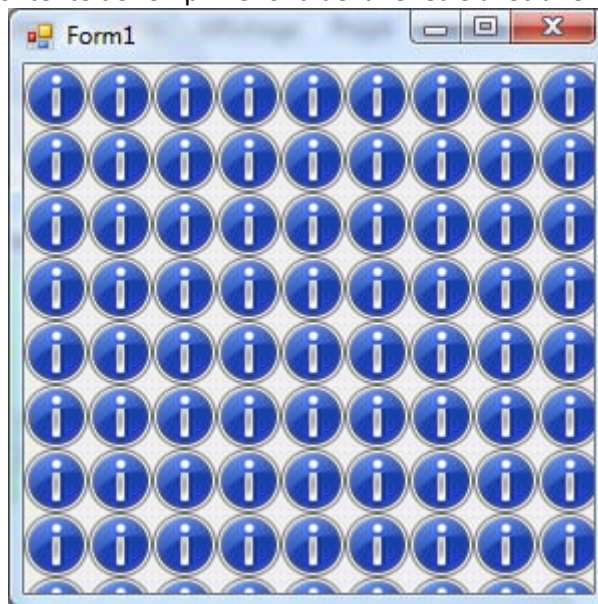
Valeur	Icone
<code>Application</code>	
<code>Asterisk</code>	
<code>Error</code>	
<code>Exclamation</code>	
<code>Hand</code>	
<code>Information</code>	
<code>Question</code>	
<code>Shield</code>	
<code>Warning</code>	
<code>WinLogo</code>	

Note : Ces icônes sont dépendantes de l'environnement Windows dans lequel vous exécutez l'application. Il se peut qu'elles soient différentes de celles-ci.

```
'VB
Sub New()
    InitializeComponent()
    Me.BackgroundImage = SystemIcons.Information.ToBitmap()
End Sub

//C#
public Partiel()
{
    InitializeComponent();
    this.BackgroundImage = SystemIcons.Information.ToBitmap();
}
```

Le code ci-dessus se contente de remplir le fond de la fenêtre avec une icône système:



4 Dessiner des formes

4.1 La classe Graphics

Nous allons maintenant voir comment dessiner des formes géométriques. Tout d'abord, nous allons devoir surveiller l'évènement `Paint` de notre fenêtre. C'est dans la méthode qui sera exécutée à l'évènement `Paint` que nous placerons nos codes de dessin.

Pour créer des formes géométriques nous devons tout d'abord récupérer un objet `Graphics`. Celui-ci nous offre plusieurs méthodes afin de dessiner nos formes géométriques, voici la liste de quelques méthodes :

Méthodes	Description
<code>Clear</code>	Vide la zone de dessin et la remplit par une couleur définie en paramètre.
<code>DrawEllipse</code>	Permet de dessiner une ellipse. L'ellipse doit forcément être contenue dans un rectangle pour être dessinée. Vérifiez bien les différentes surcharges afin de trouver celle qui vous correspond le mieux.
<code>DrawIcon / DrawIconUnstretched</code>	Dessine l'image associée à l'objet <code>Icon</code> . <code>DrawIconUnstretched</code> désactive la mise à l'échelle de l'image.
<code>DrawImage / DrawImageUnscaled / DrawImageUnscaledAndClipped</code>	Dessine l'image associée à l'objet <code>Image</code> . <code>DrawImageUnscaled</code> désactive la mise à l'échelle de l'image. <code>DrawImageUnscaledAndClipped</code> désactive la mise à l'échelle et rogne l'image.
<code>DrawLine</code>	Permet de dessiner une ligne.
<code>DrawLines</code>	Dessine plusieurs lignes grâce à un tableau de structures <code>Point</code> .
<code>DrawPath</code>	Dessine une série de lignes reliées entre elles.
<code>DrawPie</code>	Permet de dessiner des arcs de cercles ayant l'allure d'une part de tarte. <code>DrawPie</code> prend en paramètre un rectangle, et les angles de départ de d'arrivée (en radians) Vérifiez les différentes surcharges.
<code>DrawPolygon</code>	Permet de dessiner une forme d'au moins trois côtés. Prend en paramètre un tableau de structures <code>Point</code> .
<code>DrawRectangle</code>	Permet de dessiner un rectangle.
<code>DrawRectangles</code>	Permet de dessiner plusieurs rectangles grâce à un tableau de structures <code>Rectangle</code> .
<code>DrawString</code>	Permet de dessiner une chaîne de caractère.

Chacune de ces méthodes s'utilise avec un objet `Pen`, vous devez donc avant d'appeler la méthode instancier un objet `Pen`, en définissant par exemple sa couleur et la largeur de son trait.

Afin de comprendre un peu mieux comment dessiner des formes, nous allons vous présenter une petite suite d'exemple. Voici le plus simple, dessiner deux lignes (pour ce premier exemple nous avons copié le code entier de la classe `Form1`, pour la suite, nous n'indiquerons plus que le code contenu par la méthode `Form1_Paint`) :

```
'VB
Public Class Partie2
    Sub New()
        InitializeComponent()
    End Sub

    Private Sub Partie2_Paint(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        Dim g As Graphics = e.Graphics
        Dim p As Pen = New Pen(Color.Green, 2)
        g.DrawLine(p, 25, 25, 100, 100)
        g.DrawLine(p, 25, 150, 100, 50)
    End Sub
End Class

//C#
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Paint(object sender, PaintEventArgs e)
    {
        Graphics g = e.Graphics;
        Pen p = new Pen(Color.Green, 2);
        g.DrawLine(p, 25, 25, 100, 100);
        g.DrawLine(p, 25, 150, 100, 50);
    }
}
```

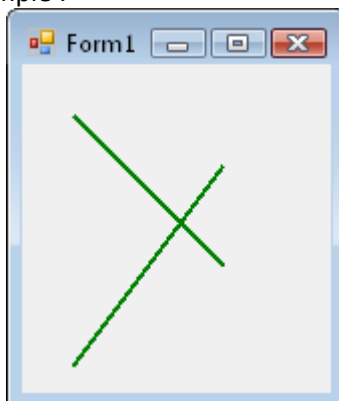
Nous récupérons un objet Graphics grâce à PaintEventArgs. Nous créons ensuite un Pen de couleur verte et de largeur 2. Enfin nous dessinons deux lignes avec des coordonnées différentes et utilisant le même Pen.

Note : Nous aurions pu utiliser à la place de e.Graphics la méthode CreateGraphics du contrôle. Pour cela il faut la faire précéder par `this` ou `Me`:

```
'VB
Dim g As Graphics = Me.CreateGraphics()

//C#
Graphics g = this.CreateGraphics();
```

Voici ce que donne notre exemple :



Ensuite, nous décidons de créer un smiley. Nous allons réutiliser nos deux droites, et rajouter d'autres éléments afin de le dessiner.

```
'VB
Dim g As Graphics = Me.CreateGraphics()

Dim contour As Pen = New Pen(Color.Yellow, 5)
Dim bouche As Pen = New Pen(Color.Red, 3)
Dim yeux As Pen = New Pen(Color.Blue, 2)

Dim points() As Point = New Point() { _
    New Point(10, 10), _
    New Point(80, 5), _
    New Point(200, 5), _
    New Point(250, 10), _
    New Point(255, 150), _
    New Point(250, 250), _
    New Point(200, 255), _
    New Point(80, 255), _
    New Point(10, 250), _
    New Point(5, 150) _
}

g.DrawLine(yeux, 25, 25, 100, 100)
g.DrawLine(yeux, 25, 150, 100, 50)
g.DrawRectangle(yeux, New Rectangle(160, 25, 80, 85))
g.DrawPie(bouche, New Rectangle(80, 80, 120, 150), 0, 180)
g.DrawPolygon(contour, points)

//C#
Graphics g = e.Graphics;
Pen contour = new Pen(Color.Yellow, 5);
Pen bouche = new Pen(Color.Red, 3);
Pen yeux = new Pen(Color.Blue, 2);

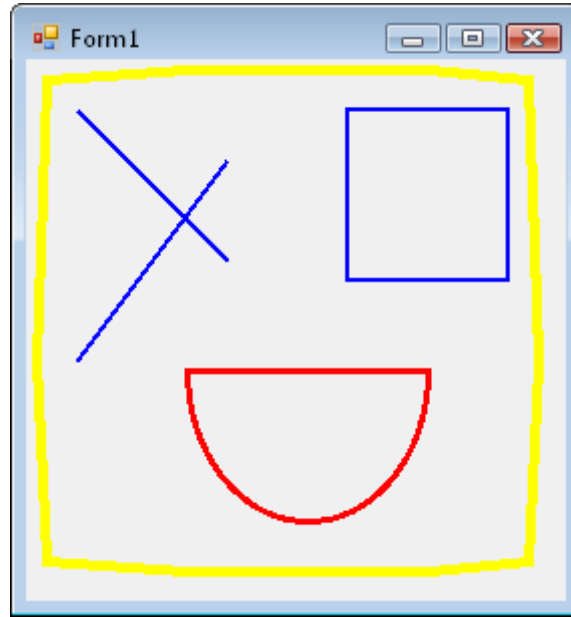
Point[] points = new Point[]
{
    new Point(10,10),
    new Point(80,5),
    new Point(200,5),
    new Point(250,10),
    new Point(255,150),
    new Point(250,250),
    new Point(200,255),
    new Point(80,255),
    new Point(10,250),
    new Point(5,150)
};

g.DrawLine(yeux, 25, 25, 100, 100);
g.DrawLine(yeux, 25, 150, 100, 50);
g.DrawRectangle(yeux, new Rectangle(160, 25, 80, 85));
g.DrawPie(bouche, new Rectangle(80, 80, 120, 150),0,180);
g.DrawPolygon(contour, points);
```

Nous récupérons donc notre objet Graphics, nous instancions trois Pen de couleur et tailles différentes, puis nous instancions un tableau de Point qui comporte 10 points différents.

Enfin nous dessinons les différentes parties : d'abord deux droites qui se croise, un rectangle, que nous instancions dans la méthode car nous n'avons pas besoin de le réutiliser ensuite, ensuite un arc de cercle de 180°, et enfin, un polygone comportant 10 arrêtes définies dans le tableau de Point.

Voici le résultat :



Dotnet-France Association

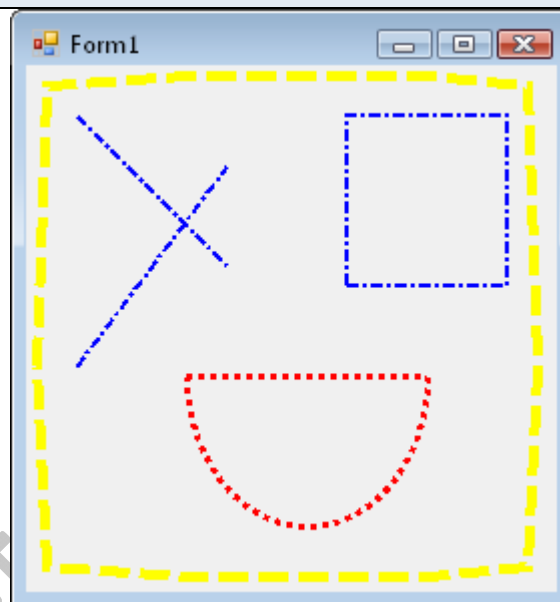
4.2 La classe Pen

Nous avons vu jusqu'à présent que nous pouvons modifier la couleur des objets Pen, mais nous pouvons également en modifier les tracés afin qu'ils soient en pointillés, ou bien qu'ils se terminent par une flèche. Pour cela nous utilisons les propriétés suivantes qui peuvent prendre les valeurs d'énumération se trouvant dans l'espace de nom `System.Drawing.Drawing2D` :

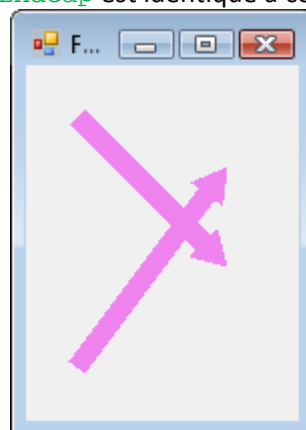
- `DashStyle` qui va nous permettre de rendre nos tracés pointillés, et qui acceptent les valeurs de l'énumération `System.Drawing.Drawing2D.DashStyle` :

```
'VB
contour.DashStyle = System.Drawing.Drawing2D.DashStyle.Dash
bouche.DashStyle = System.Drawing.Drawing2D.DashStyle.Dot
yeux.DashStyle = System.Drawing.Drawing2D.DashStyle.DashDot

//C#
contour.DashStyle = System.Drawing.Drawing2D.DashStyle.Dash;
bouche.DashStyle = System.Drawing.Drawing2D.DashStyle.Dot;
yeux.DashStyle = System.Drawing.Drawing2D.DashStyle.DashDot;
```



- `StartCap` et `EndCap` qui permettent de définir la forme de début ou de fin d'un tracé et qui acceptent les valeurs de l'énumération `System.Drawing.Drawing2D.LineCap`. L'utilisation de `StartCap` et `Endcap` est identique à celle de `DashStyle`.



4.3 L'outil Brush

L'objet `Graphics` peut non seulement dessiner des formes géométriques, mais aussi remplir ces formes de couleurs. Pour cela nous pouvons utiliser des classes contenues dans l'espace de nom `System.Drawing.Drawing2D` et `System.Drawing`, qui héritent toutes de la classe abstraite `Brush`.

Classes	Description
<code>HatchBrush</code>	Définit un pinceau de forme rectangulaire qui dessine avec un hachurage de la zone.
<code>LinearGradientBrush</code>	Définit un pinceau qui dessine un dégradé linéaire dans la zone de dessin.
<code>PathGradientBrush</code>	Définit un pinceau similaire à <code>LinearGradientBrush</code> mais qui permet des dégradés plus complexes.
<code>SolidBrush</code>	Permet de créer un Brush d'une seule couleur.
<code>TextureBrush</code>	Permet de créer un brush qui remplit avec une image.

Voici un exemple d'utilisation des Brush. Nous avons rempli les différentes formes par des couleurs, des dégradés ou des hachures. Voici le code :

```
'VB
Dim g As Graphics = e.Graphics
Dim contour As Pen = New Pen(Color.Yellow, 5)
Dim bouche As Pen = New Pen(Color.Red, 3)
Dim yeux As Pen = New Pen(Color.Blue, 2)

Dim bouche2 As Brush = New SolidBrush(Color.Red)
Dim contour2 As Brush = New LinearGradientBrush(New Point(0, 0), New
Point(255, 255), Color.Yellow, Color.YellowGreen)
Dim oeil As Brush = New HatchBrush(HatchStyle.DiagonalBrick, Color.Blue)

contour.DashStyle = System.Drawing.Drawing2D.DashStyle.Dash
bouche.DashStyle = System.Drawing.Drawing2D.DashStyle.Dot
yeux.DashStyle = System.Drawing.Drawing2D.DashStyle.DashDot

Dim points() As Point = New Point() { _
    New Point(10, 10), _
    New Point(80, 5), _
    New Point(200, 5), _
    New Point(250, 10), _
    New Point(255, 150), _
    New Point(250, 250), _
    New Point(200, 255), _
    New Point(80, 255), _
    New Point(10, 250), _
    New Point(5, 150) _
}

g.DrawPolygon(contour, points)
g.FillPolygon(contour2, points)
g.DrawLine(yeux, 25, 25, 100, 100)
g.DrawLine(yeux, 25, 150, 100, 50)
g.DrawRectangle(yeux, New Rectangle(160, 25, 80, 85))
g.FillRectangle(oeil, New Rectangle(160, 25, 80, 85))
g.DrawPie(bouche, New Rectangle(80, 80, 120, 150), 0, 180)
g.FillPie(bouche2, New Rectangle(80, 80, 120, 150), 0, 180)
```

```
//C#
Graphics g = e.Graphics;
Pen contour = new Pen(Color.Yellow, 5);
Pen bouche = new Pen(Color.Red, 3);
Pen yeux = new Pen(Color.Blue, 2);
Brush bouche2 = new SolidBrush(Color.Red);
Brush contour2 = new LinearGradientBrush(new Point(0, 0), new Point(255,
255), Color.Yellow, Color.YellowGreen);
Brush oeil = new HatchBrush(HatchStyle.DiagonalBrick, Color.Blue);

contour.DashStyle = System.Drawing.Drawing2D.DashStyle.Dash;
bouche.DashStyle = System.Drawing.Drawing2D.DashStyle.Dot;
yeux.DashStyle = System.Drawing.Drawing2D.DashStyle.DashDot;

Point[] points = new Point[]
{
    new Point(10,10),
    new Point(80,5),
    new Point(200,5),
    new Point(250,10),
    new Point(255,150),
    new Point(250,250),
    new Point(200,255),
    new Point(80,255),
    new Point(10,250),
    new Point(5,150)
};

g.DrawPolygon(contour, points);
g.FillPolygon(contour2, points);

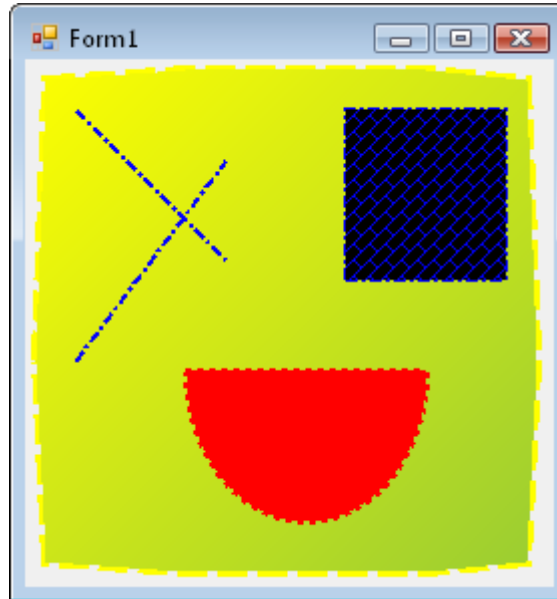
g.DrawLine(yeux, 25, 25, 100, 100);
g.DrawLine(yeux, 25, 150, 100, 50);
g.DrawRectangle(yeux, new Rectangle(160, 25, 80, 85));
g.FillRectangle(oeil, new Rectangle(160, 25, 80, 85));

g.DrawPie(bouche, new Rectangle(80, 80, 120, 150), 0, 180);
g.FillPie(bouche2, new Rectangle(80, 80, 120, 150), 0, 180);
```

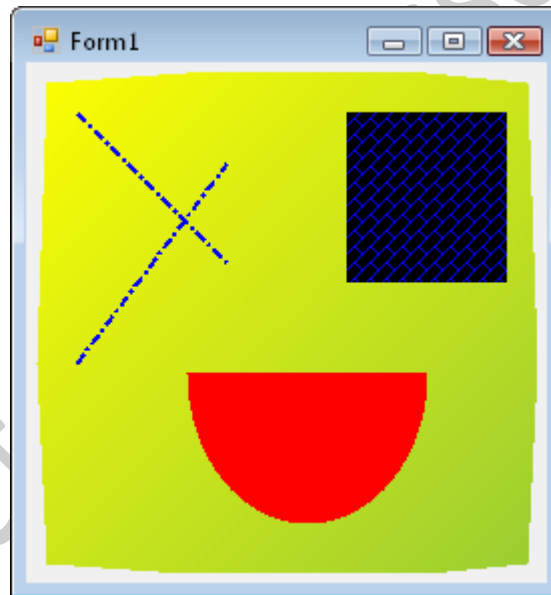
Nous avons donc instanciées des objets Brush afin de remplir les formes : un objet SolidBrush de couleur rouge, puis un objet LinearGradientBrush afin de dessiner un dégradé oblique allant du jaune au jaune vert, et enfin un objet HatchBrush bleu en forme de briquettes. (L'énumération HatchStyle vous propose un grand nombre de hachures différentes).

Ensuite, nous utilisons les méthodes FillXXX afin de remplir nos différentes formes, le principe est le même qu'avec les méthodes DrawXXX.

Voici le résultat final :



Bien sûr, vous pouvez remplir une forme géométrique n'ayant pas de contour dessiné à l'écran tant que son contour est défini dans le code :



4.4 Dessiner du texte

Afin de dessiner du texte dans une image, vous devez disposer d'une instance de la classe **Graphics**. Ensuite, il vous faut avoir une police d'écriture chargée, optionnellement les couleurs de textes et vous pourrez utiliser la méthode **DrawString** afin de dessiner le texte désiré.

Une police de texte est représentée par la classe **Font** ainsi que quelques classes annexes : La classe **FontConverter** et **FontFamily**.

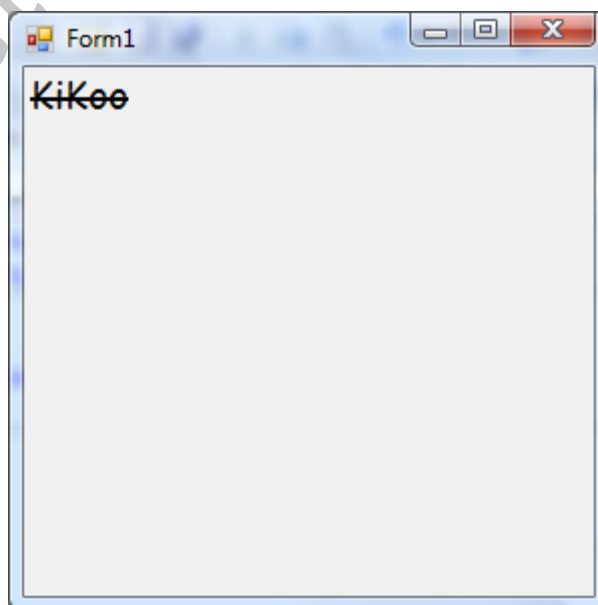
Du reste, dessiner du texte reste sensiblement identique à toutes les autres méthodes de dessin :

```
'VB
Dim f As Font
Sub New()
    InitializeComponent()
    Dim fa As FontFamily = New FontFamily("Comic Sans MS")
    f = New Font(fa, 14, FontStyle.Strikeout)
End Sub

Private Sub partie2_Paint(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
    e.Graphics.DrawString("KiKoo", f, Brushes.Black, 0, 0)
End Sub

//C#
Font f;
public Partiel()
{
    InitializeComponent();
    FontFamily fa = new FontFamily("Comic Sans MS");
    f = new Font(fa, 14, FontStyle.Strikeout);
}
private void Partiel_Paint(object sender, PaintEventArgs e)
{
    e.Graphics.DrawString("Kikoo", f, Brushes.Black, 0, 0);
}
```

Le code ci-dessus se contente de charger une police d'écriture (Comic Sans MS, rayé de taille 14). Ensuite, il va écrire la chaîne "KiKoo" en utilisant la police chargée et en mettant le texte de couleur noir.



Cette technique peut-être utile si vous souhaitez enregistrer des images avec du texte ajouté par le logiciel.

4.5 Formater le texte

Ici, nous avons vu comment dessiner du texte de façon simple. Sachez toutefois que la méthode `DrawString` vous permet également de modifier l'organisation du texte au moment du dessin. En effet, l'une de ses surcharges prend aussi un objet `StringFormat` en paramètre.

Cette classe permet de modifier plusieurs paramètres de présentation du texte. Voici les différentes propriétés dont dispose cette classe ainsi que les valeurs qu'elles peuvent prendre :

Valeur	Description
<code>Alignment</code>	– Prend l'une des valeurs de <code>StringAlignment</code> .
<code>Near</code>	Aligne le texte sur la gauche.
<code>Far</code>	Aligne le texte sur la droite.
<code>Center</code>	Centre horizontalement le texte.
<code>FormatFlags</code>	– Prend l'une des valeurs de <code>StringFormatFlags</code> .
<code>DirectionRightToLeft</code>	Le texte est orienté de droite à gauche.
<code>DirectionVertical</code>	Le texte est orienté selon un axe vertical.
<code>DisplayFormatControl</code>	Active ou non l'affichage des caractères de contrôles.
<code>FitBlackBox</code>	Les caractères en dehors des limites d'affichage peuvent être placés au dessus du cadre. Par défaut, ils sont réorganisés pour éviter le surplombage.
<code>LineLimit</code>	Seules des lignes complètes sont affichées dans le cadre de destination du texte. Par défaut, les mots sont affichés même si ceux-ci font partis d'une ligne tronquée.
<code>MeasureTrailingSpaces</code>	Prend en compte les fins de ligne dans le calcul des dimensions de la chaîne de caractère grâce à la méthode <code>MeasureString</code> .
<code>NoClip</code>	Les parties du texte qui sortent du rectangle d'affichage seront affichées.
<code>NotFontFallback</code>	Autorise ou non l'utilisation d'une autre police de texte si des caractères à afficher ne sont pas pris en charge par la police actuelle. Généralement, un carré sera utilisé à la place du caractère non-affichable.
<code>NoWrap</code>	Annule tout habillage de texte. Si des caractères sont en dehors du cadre d'affichage, ils pourront être affichés mais seront rognés.
<code>LineAlignment</code>	– Prend l'une des valeurs de <code>StringAlignment</code> .
<code>Center</code>	Centre le texte verticalement
<code>Near</code>	Aligne le texte sur le haut.
<code>Far</code>	Aligne le texte sur le bas.
<code>Trimming</code>	– Prend l'une des valeurs de <code>StringTrimming</code>
<code>Character</code>	Réduit la chaîne de caractères au dernier caractère affichable.
<code>EllipsisCharacter</code>	Réduit la chaîne de caractères au dernier caractère affichable et remplace le reste par "...".
<code>EllipsisPath</code>	Supprime le centre de la chaîne de caractères en le remplaçant par "...".
<code>EllipsisWord</code>	Réduit la chaîne de caractères au dernier mot affichable et affiche "...".
<code>None</code>	Supprime tout rognage du texte.
<code>Word</code>	Réduit la chaîne de caractère au dernier mot affichable.

Grâce à ces formats, vous pouvez disposer le texte à votre convenance. Par exemple, le code ci-dessous se contente d'afficher les caractères saisis au clavier. Si on appui sur la touche tabulation, cela efface la chaîne de caractère remplie et si on appui sur Entrée, ça change le formatage de texte :

```
'VB
Dim police As Font
Dim format As StringFormat
Dim chaine As StringBuilder
Dim r As Rectangle

Sub New()
    InitializeComponent()
    police = New Font("Comic Sans MS", 14, FontStyle.Regular)

    format = New StringFormat()
    format.FormatFlags = StringFormatFlags.DirectionVertical
    format.LineAlignment = StringAlignment.Center
    format.Trimming = StringTrimming.EllipsisCharacter

    r = New Rectangle(0, 0, 120, 120)
    chaine = New StringBuilder()
End Sub

Private Sub partie2_Paint(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
    e.Graphics.Clear(Color.Gray)
    e.Graphics.DrawString(chaine.ToString(), police, Brushes.Black, r,
format)
End Sub

Private Sub partie2_KeyPress(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.KeyPressEventArgs) Handles MyBase.KeyPress
    If (e.KeyChar = Char.Parse(vbTab)) Then
        chaine.Remove(0, chaine.Length)
    ElseIf (e.KeyChar = Char.Parse(vbCr)) Then
        Select Case (format.FormatFlags)
            Case StringFormatFlags.DirectionRightToLeft
                format.FormatFlags =
StringFormatFlags.DirectionVertical
                Exit Select
            Case StringFormatFlags.DirectionVertical
                format.FormatFlags =
StringFormatFlags.DisplayFormatControl
                Exit Select
            Case StringFormatFlags.DisplayFormatControl
                format.FormatFlags =
StringFormatFlags.DirectionRightToLeft
                Exit Select
        End Select
    Else
        chaine.Append(e.KeyChar)
    End If
    Me.InvokePaint(Me, New PaintEventArgs(Me.CreateGraphics(), r))
End Sub
```

```
//C#
Font police;
Font f;
StringFormat format;
StringBuilder chaine;
Rectangle r;

public Partiel()
{
    InitializeComponent();
    police = new Font("Comic Sans MS", 14, FontStyle.Regular);

    format = new StringFormat();
    format.FormatFlags = StringFormatFlags.DirectionVertical;
    format.LineAlignment = StringAlignment.Center;
    format.Trimming = StringTrimming.EllipsisCharacter;

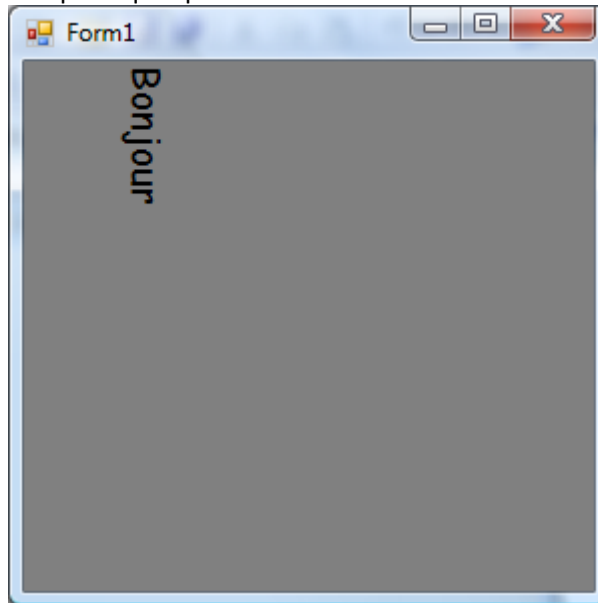
    r = new Rectangle(0, 0, 120, 120);
    chaine = new StringBuilder();
}

private void Partiel_Paint(object sender, PaintEventArgs e)
{
    e.Graphics.Clear(Color.Gray);
    e.Graphics.DrawString(chaine.ToString(), police, Brushes.Black, r,
format);
}

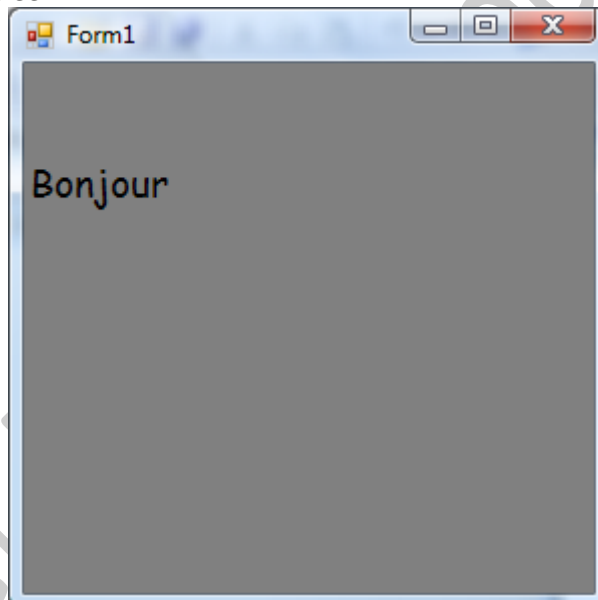
private void Partiel_KeyPress(object sender, KeyPressEventArgs e)
{
    if(e.KeyChar == char.Parse("\t"))
    {
        chaine.Remove(0, chaine.Length);
    }
    else if (e.KeyChar == char.Parse("\r"))
    {
        switch(format.FormatFlags)
        {
            case StringFormatFlags.DirectionRightToLeft:
                format.FormatFlags = StringFormatFlags.DirectionVertical;
                break;
            case StringFormatFlags.DirectionVertical:
                format.FormatFlags =
StringFormatFlags.DisplayFormatControl;
                break;
            case StringFormatFlags.DisplayFormatControl:
                format.FormatFlags =
StringFormatFlags.DirectionRightToLeft;
                break;
        }
    }
    else
        chaine.Append(e.KeyChar);

    this.InvokePaint(this, new PaintEventArgs(this.CreateGraphics(), r));
}
```

A la compilation, si nous tapons quelques lettres :



Et si on appui sur "Entrée" :



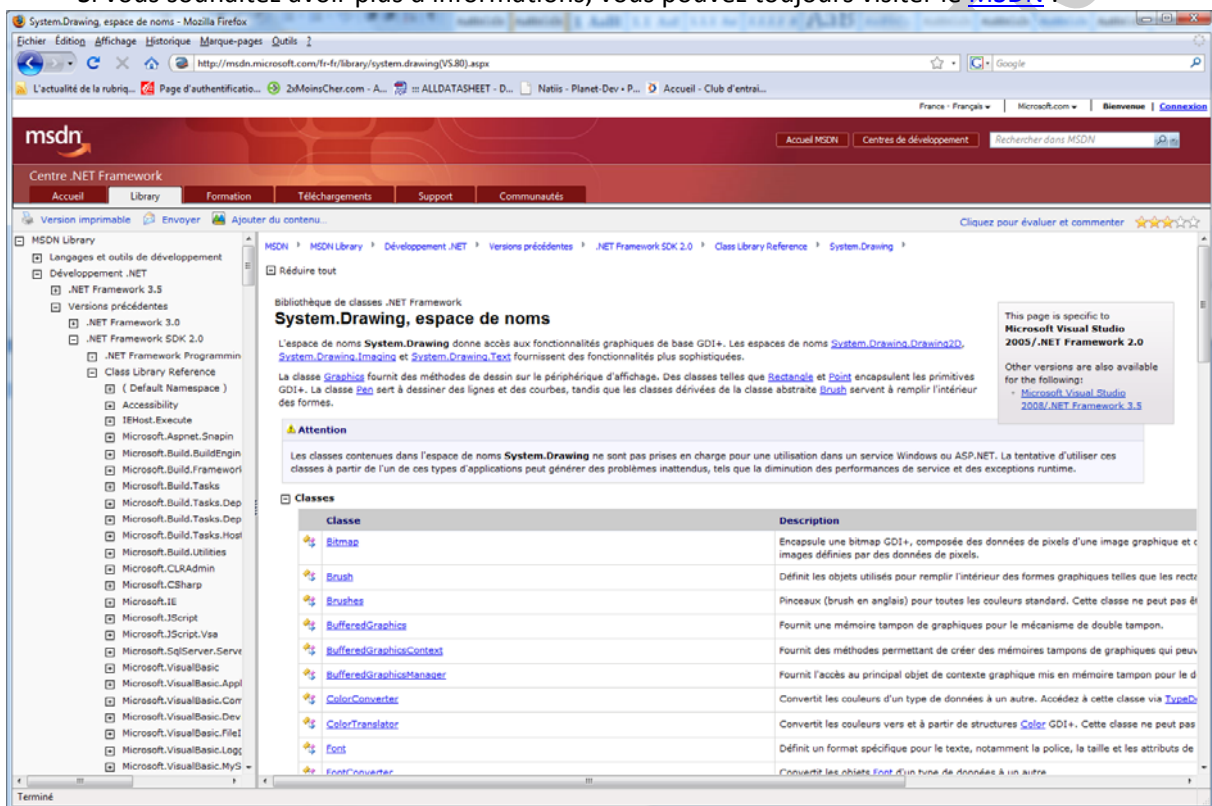
5 Conclusion

Vous avez pu constater qu'il est très facile de manipuler les images avec le .NET Framework. En effet, seules deux ou trois classes sont nécessaires pour charger les données d'une image, les modifier et/ou les afficher puis les enregistrer dans un fichier.

Aussi à la fin de ce chapitre, vous devriez être capable de:

- Charger une image, y apporter des modifications et enregistrer les changements effectués dans un fichier.
- Dessiner des formes simples telles que des cercles, des rectangles, des lignes ...
- Avoir compris la différence entre Pen et Brush et savoir comment utiliser ces deux classes pour personnaliser vos tracés.
- Savoir dessiner du texte sur une image.

Si vous souhaitez avoir plus d'informations, vous pouvez toujours visiter le [MSDN](http://msdn.microsoft.com/fr-fr/library/system.drawing(VS.80).aspx) :



The screenshot shows the MSDN website for the System.Drawing namespace. The page title is "System.Drawing, espace de noms". The content includes an introduction to the namespace, a list of classes, and a table of classes with descriptions.

System.Drawing, espace de noms

L'espace de noms **System.Drawing** donne accès aux fonctionnalités graphiques de base GDI+. Les espaces de noms [System.Drawing.Drawing2D](#), [System.Drawing.Imaging](#) et [System.Drawing.Text](#) fournissent des fonctionnalités plus sophistiquées.

La classe [Graphics](#) fournit des méthodes de dessin sur le périphérique d'affichage. Des classes telles que [Rectangle](#) et [Point](#) encapsulent les primitives GDI+. La classe [Pen](#) sert à dessiner des lignes et des courbes, tandis que les classes dérivées de la classe abstraite [Brush](#) servent à remplir l'intérieur des formes.

Attention

Les classes contenues dans l'espace de noms **System.Drawing** ne sont pas prises en charge pour une utilisation dans un service Windows ou ASP.NET. La tentative d'utiliser ces classes à partir de l'un de ces types d'applications peut générer des problèmes inattendus, tels que la diminution des performances de service et des exceptions runtime.

Classe	Description
Bitmap	Encapsule un bitmap GDI+, composée des données de pixels d'une image graphique et des images définies par des données de pixels.
Brush	Définit les objets utilisés pour remplir l'intérieur des formes graphiques telles que les rectangles.
Brushes	Pinceaux (brush en anglais) pour toutes les couleurs standard. Cette classe ne peut pas être héritée.
BufferedGraphics	Fournit une mémoire tampon de graphiques pour le mécanisme de double tampon.
BufferedGraphicsContext	Fournit des méthodes permettant de créer des mémoires tampons de graphiques qui peuvent être héritées.
BufferedGraphicsManager	Fournit l'accès au principal objet de contexte graphique mis en mémoire tampon pour le double tampon.
ColorConverter	Convertit les couleurs d'un type de données à un autre. Accédez à cette classe via TypeConverter .
ColorTranslator	Convertit les couleurs vers et à partir de structures Color GDI+. Cette classe ne peut pas être héritée.
Font	Définit un format spécifique pour le texte, notamment la police, la taille et les attributs de mise en forme.
FontConverter	Convertit les objets Font d'un type de données à un autre.