

Résumé : Applications Windows C#/.NET

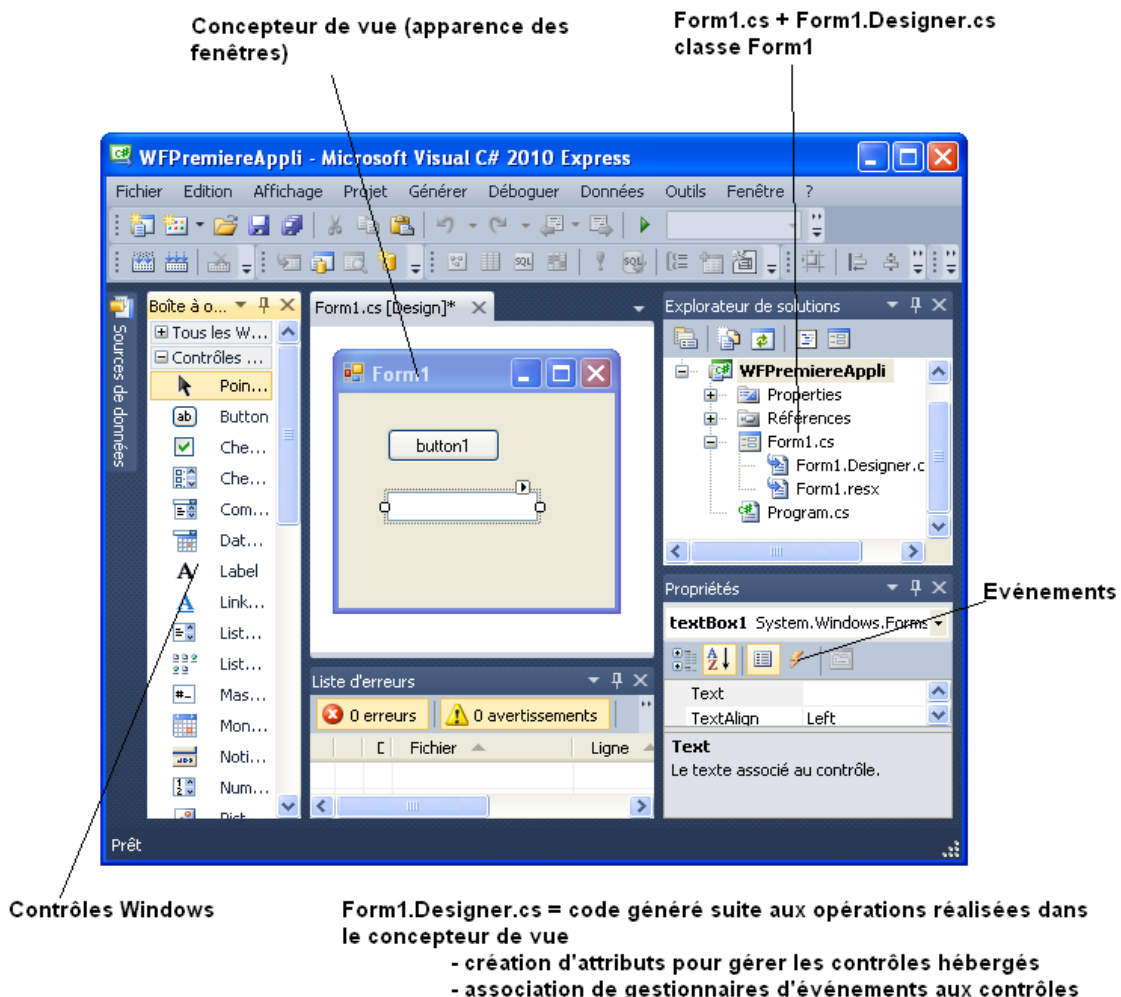
Ce document fournit des indications sur la création d'applications Windows (avec fenêtres) en C# pour .NET. Pour cela, on peut utiliser Visual Studio ou la version Visual C# Express. Dans ces IDE, la création du projet est effectuée par :

Fichier / Nouveau Projet / Visual C# / Application Windows Forms
Nom :
Emplacement :

Important : Le nom de projet sert également de nom de **namespace**.

1. Structure d'une application Windows

Suite à la création du projet, les différentes fenêtres de l'IDE vous permettent d'atteindre du code (explorateur de solution), de modifier l'apparence des fenêtres (concepteur de vue), de visualiser les propriétés et événements des contrôles.



Exemple : application avec deux contrôles Button + TextBox

Form1 = classe de la fenêtre principale

Une instance de cette classe est générée dans la fonction Main() (Program.cs)

Form1 est écrite sur 2 fichiers (**Form1.cs + Form1.Designer.cs**)

Form1.cs : fichier modifié par nos soins (gestionnaires d'événements)

Form1.Designer.cs : modifié par IDE suite aux opérations faites dans le concepteur de vue.

QUAND ON AJOUTE UN CONTRÔLE :

- attribut supplémentaire dans la classe **Form1** (voir Form1.Designer.cs)
- instantiation de la classe du contrôle dans la méthode **Form1.InitializeComponent()**;

//Form1.cs (Une partie de la classe Form1)

```
namespace WFPremiereAppli
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent(); // Cette méthode crée les objets Contrôles
                                // Voir fichier Form1.Designer.cs
        }
    }
}
```

//Form1.Designer.cs (Autre partie de la classe Form1)

// Extraits du fichier

```
namespace WFPremiereAppli
{
    partial class Form1
    {
        ...

        private void InitializeComponent()
        {
            this.button1 = new System.Windows.Forms.Button(); //instanciation
            this.textBox1 = new System.Windows.Forms.TextBox();

            ...
            this.button1.Text = "button1";
            this.button1.UseVisualStyleBackColor = true;
            this.textBox1.Name = "textBox1";

            ...
            this.Controls.Add(this.textBox1);
            this.Controls.Add(this.button1);

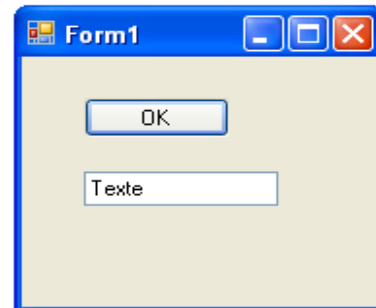
            ...
        }

        private System.Windows.Forms.Button button1; // attribut de type Button
        private System.Windows.Forms.TextBox textBox1; // attribut de type TextBox
    }
}
```

2. Accès aux contrôles dans le code

Depuis n'importe quelle méthode de la classe Form1, les contrôles hébergés par la fenêtre principale sont atteignables grâce à leurs noms d'attributs.

```
namespace WFPremiereAppli
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            button1.Text = "OK";
            textBox1.Text = "Texte";
        }
    }
}
```



Quelques propriétés communes aux contrôles

Text = (type string) définit le texte d'un label, d'un textbox, d'un bouton ...

Location = (type Point) position X,Y du contrôle [(0,0) en haut à gauche]

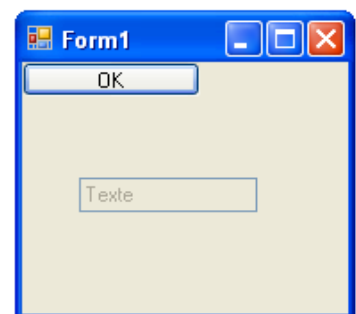
Size = (type Size) taille X,Y du contrôle

Enabled = (type bool) activation du contrôle

Tag = (type object) n'importe quelle variable/objet que l'on "attache" au contrôle

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        button1.Text = "OK";
        button1.Location = new System.Drawing.Point(0, 0);
        button1.Size = new System.Drawing.Size(100, 20);
        button1.Enabled = true;
        button1.Tag = 1;


        textBox1.Text = "Texte";
        textBox1.Enabled = false;
        textBox1.Tag = 2;
    }
}
```



3. Gestionnaires d'événements

Les contrôles déclenchent des événements auxquels on peut abonner des méthodes (gestionnaires d'événements).

Pour associer un gestionnaire d'événements à l'événement d'un contrôle :

- 1) sélectionner le contrôle (concepteur de vue)
- 2) Fenêtre Propriétés | Onglet  → Double Click sur l'événement

Exemple : Gestionnaire de l'événement **Click** d'un bouton

// Form1.cs

```
public partial class Form1 : Form{
    public Form1()
    {
        InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        MessageBox.Show("Appui sur le bouton");
    }
}
```

// Form1.Designer.cs

```
namespace WFPremiereAppli
{
    partial class Form1
    {
        ...

        private void InitializeComponent()
        {
            this.button1 = new System.Windows.Forms.Button(); //instanciation
            ...
            this.button1.Click += new System.EventHandler(this.button1_Click);
            ...
        }

        private System.Windows.Forms.Button button1; // attribut de type Button
        private System.Windows.Forms.TextBox textBox1;
    }
}
```



Signature des événements (la délégation associée)

```
delegate void EventHandler(object sender, EventArgs e)
```


Les événements peuvent donc déclencher des appels de méthodes prenant deux paramètres

sender = référence à l'objet ayant déclenché l'événement (souvent un contrôle)

e = arguments (données) fournies au gestionnaire d'événement

3.1. Même gestionnaire pour plusieurs événements

On peut associer le même gestionnaire aux événements de plusieurs contrôles.

- 3) sélectionner le deuxième contrôle (concepteur de vue)
- 4) Fenêtre Propriétés | Onglet  → choisir un gestionnaire existant dans la liste

C#/NET

// Form1.cs

```
public partial class Form1 : Form
{
    public Form1() { InitializeComponent(); }

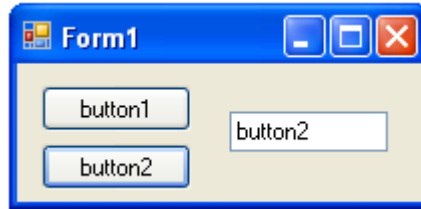
    private void button1_Click(object sender, EventArgs e)
    {
        Button b = sender as Button; // conversion object vers Button
        if (b != null) // si l'émetteur est réellement un Button
        {
            textBox1.Text = b.Name;
        }
    }
}
```

// Form1.Designer.cs

```
namespace WFPremiereAppli
{
    partial class Form1
    {
        ...

        private void InitializeComponent()
        {
            this.button1 = new System.Windows.Forms.Button(); //instanciation
            this.button2 = new System.Windows.Forms.Button(); //instanciation
            ...
            this.button1.Click += new System.EventHandler(this.button1_Click);
            this.button2.Click += new System.EventHandler(this.button1_Click);
            ...
        }

        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.Button button2;
        private System.Windows.Forms.TextBox textBox1;
    }
}
```



4. Formulaire (Fenêtre) Supplémentaire

4.1. MessageBox

Crée une boîte de dialogue pour afficher un message, ou poser une question à l'utilisateur.

```
private void buttonFormulaire_Click(object sender, EventArgs e)
{
    MessageBox.Show("Message seul");
    MessageBox.Show("Message", "Titre de la boîte");

    // MessageBoxButtons = énumération de différentes configurations
    DialogResult result;
    result = MessageBox.Show("Continuer ?", "Attention",
                            MessageBoxButtons.OKCancel);
    if (result == DialogResult.OK)
    {
        MessageBox.Show("Vous avez cliqué su OK");
    }
}
```

4.2. Formulaire en plus

Pour ajouter une nouvelle classe de fenêtre, dans l'explorateur de solutions, click-droit sur nom de projet puis Ajouter/Formulaire Windows. Dans l'explorateur de solution, on voit alors deux nouveaux fichiers pour le formulaire supplémentaire

Ex : FormSupp.cs + FormSupp.Designer.cs

Ce nouveau formulaire peut lui-même contenir des contrôles qui seront vus comme des attributs de la classe, et des gestionnaires associés aux contrôles.

OUVERTURE MODALE

On n'a plus accès au reste de l'application avant la fermeture du formulaire.

```
public partial class Form1 : Form
{
    public Form1 ()
    {
        InitializeComponent();
    }
    private void buttonFormulaire_Click(object sender, EventArgs e)
    {
        FormSupp form = new FormSupp();

        // FORMULAIRE MODAL
        form.ShowDialog();
    }
}
```

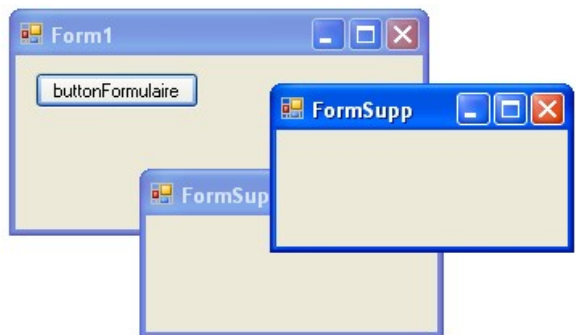


OUVERTURE NON MODALE

```
public partial class Form1 : Form
{
    public Form1 ()
    {
        InitializeComponent();
    }

    private void buttonFormulaire_Click(object sender, EventArgs e)
    {
        FormSupp form = new FormSupp();

        // FORMULAIRE NON MODAL
        form.Show();
    }
}
```



LE PROBLEME PRINCIPAL EST L'ECHANGE D'INFORMATIONS ENTRE LES DIFFERENTS FORMULAIRES DE L'APPLICATION.