



Accueil Conception Java .NET Dév. Web EDI Langages SGBD Office Solutions d'entreprise Applications Systèmes

Langages Assembleur C C++ C# Pascal Perl Python Visual Basic 6 Visual Basic.NET XML Autres

Forum C++ FAQs C++ Tutoriels C++ Livres C++ Outils & compilateurs C++ Bibliothèques C++ Sources C++ Blog C++ Qt   

# Démarrer avec les Winforms en C++/CLI avec le framework .Net 2.0 de Visual C++ 2005

Date de publication : 16/02/2006 , Date de mise à jour : 17/02/2009

Par Nico-pyright(c) (Page d'accueil)

A travers ce tutoriel, vous trouverez une introduction à la programmation des Winforms grâce au framework .net 2.0 à travers l'utilisation du langage C++/CLI avec Visual C++ 2005

- 1.Introduction
- 2.A qui s'adresse ce tutoriel
- 3.Définition des Winforms et Hello World
  - 3.1.Définition des Winforms
  - 3.2.Hello World
  - 3.3.Rentrons dans le code de l'Hello World
- 4.Les contrôles
  - 4.1.Généralités
  - 4.2.La form
  - 4.3.Le bouton
  - 4.4.CheckBox
  - 4.5.CheckedListBox
  - 4.6.ComboBox
  - 4.7.DateTimePicker
  - 4.8.Label
  - 4.9.LinkLabel
  - 4.10.ListBox
  - 4.11.ListView
  - 4.12.MaskedTextBox
  - 4.13.MonthCalendar
  - 4.14.NotifyIcon
  - 4.15.NumericUpDown
  - 4.16.PictureBox
  - 4.17.ProgressBar
  - 4.18.RadioButton
  - 4.19.RichTextBox
  - 4.20.TextBox
  - 4.21.ToolTip
  - 4.22.TreeView
  - 4.23.WebBrowser
  - 4.24.Les autres contrôles ...
- 5.Le programme d'exemple
  - 5.1.Présentation
  - 5.2.Elaboration de l'application
    - 5.2.1.Chargement de l'application, WinForm d'attente avec une barre de progression.
    - 5.2.2.Construction de la fenêtre principale.
    - 5.2.3.La classe CMyNetSend.
    - 5.2.4.L'envoi.
    - 5.2.5.Conclusion et téléchargement.
- Suite
- Remerciements
- Contact

## 1.Introduction

Vous êtes un développeur, vous avez des bases dans le langage C/C++ et vous souhaitez acquérir des connaissances en programmation Windows dans le domaine du développement d'interfaces homme machine en .Net grâce au framework 2.0 et le C++/CLI ? Alors ce tutoriel est pour vous.

Vous trouverez dans ce cours une introduction à la programmation des Winforms (Windows Forms) et un détail des principaux contrôles .Net.

J'illustrerai ces notions avec un programme exemple utilisant plusieurs contrôles .Net.

A travers ce tutoriel, j'utiliserai le logiciel Visual Studio 2005 de Microsoft et plus particulièrement **Visual C++ 2005** ainsi que le framework dotnet 2.0.

## 2.A qui s'adresse ce tutoriel

Avant toutes choses, il est important de comprendre que :

- Ce tutoriel s'adresse en particulier à des développeurs ayant des bases en C++ et qui ne connaissent pas (ou peu) le développement d'IHM sous Windows par l'intermédiaire des MFC ou de l'API Win32.
- Ce tutoriel n'est pas recommandé à des personnes n'ayant pas de bases en C++. Je conseillerais à des personnes qui veulent se lancer dans la programmation des Winforms sans bases de C++ d'utiliser plutôt le C#. En effet, je trouve qu'au premier abord, la syntaxe du C++ est plus déroutante.
- Le C++ est intéressant car il permet d'avoir plus de souplesse et il permet notamment d'intégrer du code non managé dans son application pour l'utiliser par exemple lors de section critiques qui ont besoin d'être d'un niveau le plus bas possible.
- De plus, le C++ dispose d'une gestion de l'interop plus efficace que les autres langages .Net. L'optimisation est en général meilleure, même au niveau de la génération du MSIL (Microsoft Intermediate Language). En plus, le C++ permet d'avoir une productivité accrue de développement, en utilisant la STL ou les templates et dispose de la destruction déterministe, même pour les types managés.
- Pour des personnes qui ne connaissent pas la programmation Windows avec l'API Win32 ou les MFC, il est beaucoup plus simple d'apprendre les Winforms que l'API Win32 ou MFC. Cependant, pour des personnes connaissant déjà la programmation Windows ou voulant disposer d'un framework complet et performant, je conseille d'utiliser les MFC plutôt que de vouloir apprendre les Winforms.
- Si vous utilisez les MFC, vous pourrez apprendre à cet emplacement comment intégrer du code managé dans des applications Win32 ou MFC existantes.

Les avantages des winforms sont :

- Très simple à prendre en main, grâce à l'outil de design inclus dans l'IDE. Il suffit de glisser-déposer des contrôles sur la form, de cliquer sur ces contrôles pour générer les événements, etc ...
- Le code est plus simple à faire et est plus propre qu'avec les MFC.
- Le code est très orienté objet et propose un très haut niveau d'abstraction.

Si vous vous êtes retrouvés dans ces raisons, ou si vous êtes simplement curieux de connaître la programmation des Winforms, suivez moi au prochain chapitre.

Vous pouvez lire en préambule cette introduction au monde du C++/CLI. Cet article précise les concepts inhérents au monde du C++/CLI.

## 3.Définition des Winforms et Hello World

### 3.1.Définition des Winforms

Microsoft Windows Forms (ou Winforms) est le nom donné à la partie du framework .net responsable de l'interface graphique (GUI), donnant accès aux contrôles windows managés.

Les Winforms se rapprochent beaucoup des forms comme on peut les connaître en VB. Microsoft s'est basé sur le modèle de développement des interfaces graphiques en VB, des forms, des contrôles et des propriétés pour créer un nouveau langage équivalent pour le framework.net.

Ainsi, il est très facile de créer une fenêtre, des boutons, d'écrire du texte, d'associer un événement à un click sur un bouton, etc ... Tout ceci est sans effort grâce à la volonté de Microsoft de nous fournir un IDE puissant et performant où il suffit de cliquer, de glisser/déposer pour créer une interface graphique.

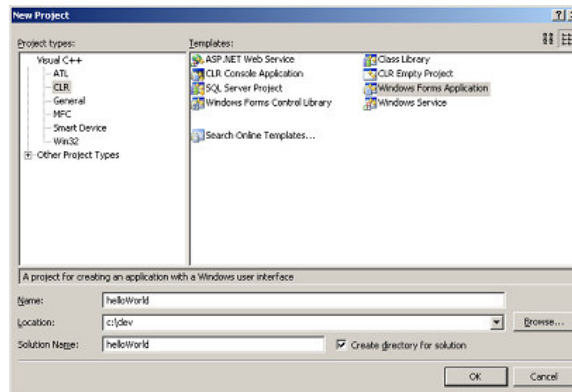
Les winforms sont basées sur GDI+, un ensemble de classes et de fonctionnalités qui permettent de créer et de gérer les IHMs.

Les classes disponibles pour utiliser les Windows Forms se trouvent dans le CLR (Common Language Runtime). La classe fondamentale est `System::Windows::Forms::Form`. Et comme une Winform fait partie intégrante du CLR, elle est sujette à l'héritage, c'est à dire que l'on peut construire une hiérarchie de classes dérivées de Winforms dans une optique très orientée objet.

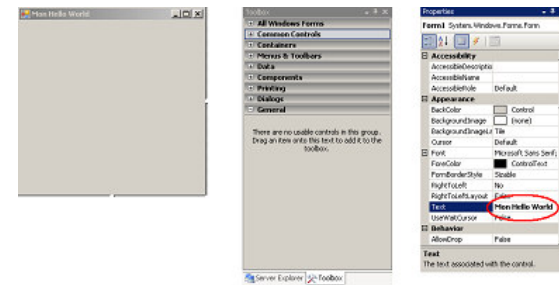
### 3.2.Hello World

Revenons tout de suite dans le vif du sujet avec le traditionnel "Hello World".

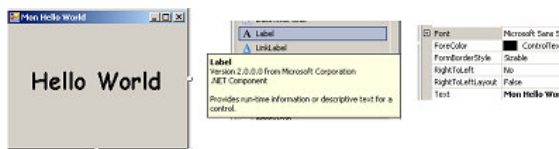
Tout d'abord, créez un nouveau projet de type CLR / Windows Form Application.



On se retrouve dans l'éditeur de ressources où nous voyons notre form, la barre d'outils d'ajout de contrôles et la fenêtre de propriétés. Cliquer sur la form pour visualiser la fenêtre de propriétés associée à la form, et modifier la propriété **text** de celle-ci. Nous changeons ainsi le texte de la barre de titre. Profitez-en pour mettre la propriété **autosize** à true, qui permet comme son nom l'indique à ce que la form se redimensionne toute seule.



Déployez la barre d'outils **common controls** et dessinez un label comme ci-dessous. Vous le placez sur la form et positionnez sa propriété **text** ainsi que la propriété **font** (dans l'exemple : comic sans MS, 24 pt en gras).



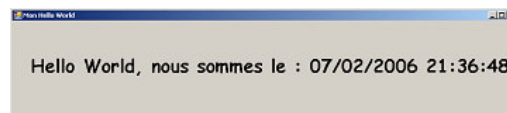
Vous pouvez lancer la compilation et l'exécution du projet, vous verrez alors apparaître la form comme dessinée.

Pour aller un peu plus loin et introduire la notion d'évènement, double-cliquez dans l'éditeur de form sur le label, cela aura pour effet de générer du code qui correspond à un évènement d'un click sur le label. Rajoutez le code suivant dans la fonction générée.

```
private: System::Void label1_Click(System::Object^ sender, System::EventArgs^ e)
{
    System::String ^ chaine;
    System::DateTime ^ date = System::DateTime::Now;
    chaine = "Hello World, nous sommes le : " + date->ToString();
    label1->Text = chaine;
}

```

Ici, nous créons une chaîne de caractère et un objet managé date contenant la date du jour. Puis nous créons la chaîne avec la date du jour et l'affectons à la propriété text de l'objet label1. Exécutez le programme et cliquez sur le label Hello world, vous verrez apparaître la fenêtre comme suit.



Félicitations, vous avez créé votre première application Windows Forms.

### 3.3. Revenons dans le code de l'Hello World

Le wizard nous a généré un certain nombre de fichiers, dont le fichier helloworld.cpp. Nous y trouvons entre autres des initialisations, le point d'entrée de l'application **main** ainsi que le lancement (et l'instanciation) de notre form. Le code parle de lui-même.

```
int main(array<System::String ^> ^args)
{
    // Enabling Windows XP visual effects before any controls are created
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);

    // Create the main window and run it
    Application::Run(gcnew Form1());
    return 0;
}
```

Nous avons aussi un fichier Form1.h qui contient du code (pour le visualiser : click droit sur le fichier --> view code).

```

#pragma once

namespace helloWorld {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    /// <summary>
    /// Summary for Form1
    ///
    /// WARNING: If you change the name of this class, you will need to change the
    /// 'Resource File Name' property for the managed resource compiler tool
    /// associated with all .resx files this class depends on. Otherwise,
    /// the designers will not be able to interact properly with localized
    /// resources associated with this form.
    /// </summary>
    public ref class Form1 : public System::Windows::Forms::Form
    {
    public:
        Form1(void)
        {
            InitializeComponent();
            //
            //TODO: Add the constructor code here
            //
        }

    protected:
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        ~Form1()
        {
            if (components)
            {
                delete components;
            }
        }
    private: System::Windows::Forms::Label^ labell;
    protected:

    private:
        /// <summary>
        /// Required designer variable.
        /// </summary>
        System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        void InitializeComponent(void)
        {
            this->labell = (gcnew System::Windows::Forms::Label());
            this->SuspendLayout();
            //
            // labell
            //
            this->labell->AutoSize = true;
            this->labell->Font = (gcnew System::Drawing::Font(L"Comic Sans MS", 24,
                System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Po-
                static_cast<System::Byte>(0)));
            this->labell->Location = System::Drawing::Point(25, 55);
            this->labell->Name = L"labell";
            this->labell->Size = System::Drawing::Size(205, 45);
            this->labell->TabIndex = 0;
            this->labell->Text = L"Hello World";
            this->labell->Click += gcnew System::EventHandler(this, &Form1::labell_Click);
            //
            // Form1
            //
            this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
            this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
            this->AutoSize = true;
            this->ClientSize = System::Drawing::Size(253, 172);
            this->Controls->Add(this->labell);
            this->Name = L"Form1";
            this->Text = L"Mon Hello World";
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
    private: System::Void labell_Click(System::Object^ sender, System::EventArgs^ e)
    {
        System::String ^ chaine;
        System::DateTime ^ date = System::DateTime::Now;
        chaine = "Hello World, nous sommes le : " + date->ToString();
        labell->Text = chaine;
    }
};
}

```

Nous observons tout un bout de code qui a été généré lors de nos divers click et glisser/déplacer sur la form lors de la phase de design.

Nous reconnaissons en l'occurrence la construction de la classe Form1 qui hérite de System::Windows::Forms::Form. C'est cet objet qui est instancié dans le main et qui correspond à notre form désignée.

```
public ref class Form1 : public System::Windows::Forms::Form
```

On devine aussi la construction du label et l'affectation des propriétés (autosize, font et text notamment) :

```
this->label1->AutoSize = true;
this->label1->Font = (gcnew System::Drawing::Font(L"Comic Sans MS", 24,
    System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
    static_cast<System::Byte>(0)));
this->label1->Location = System::Drawing::Point(25, 55);
this->label1->Name = L"label1";
this->label1->Size = System::Drawing::Size(205, 45);
this->label1->TabIndex = 0;
this->label1->Text = L"Hello World";
this->label1->Click += gcnew System::EventHandler(this, &Form1::label1_Click);
```

La dernière ligne correspond à l'ajout d'un handler pour capter l'évènement sur le click du bouton, associé à la méthode **label1\_Click**. (Plus de précisions sur MSDN)

Enfin, on observe ce qui correspond à l'initialisation de la form :

```
this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
this->AutoSize = true;
this->ClientSize = System::Drawing::Size(253, 172);
this->Controls->Add(this->label1);
this->Name = L"Form1";
this->Text = L"Mon Hello World";
this->ResumeLayout(false);
this->PerformLayout();
```

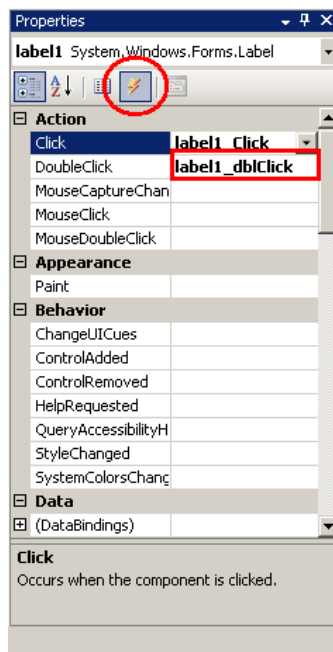
Notez notamment la ligne où l'on ajoute le label à la form.

```
this->Controls->Add(this->label1);
```

Le tout dans un namespace Hello World.

Comprendre ce code n'est pas primordial dans un premier temps, et bien souvent vous n'aurez même pas à vous soucier du code généré (simplement de retrouver la fonction correspondant à un évènement, dans notre cas label1\_Click pour le click sur le label), mais le comprendre permettra une meilleure utilisation future dans des cas particuliers.

Remarque : Si nous avions voulu intercepter le double click plutôt que le click (qui est l'évènement géré par défaut), il aura fallu cliquer sur le bouton "**events**" dans la fenêtre de propriété et taper le nom d'une fonction dans la zone correspondant à l'évènement du double click, comme indiqué sur cette image :



Visual Studio vous aurait alors généré cette fonction :

```
private: System::Void label1_dblClick(System::Object^ sender, System::EventArgs^ e) {
}
```

Il aurait de même ajouté l'évènement ainsi :

```
this->label1->DoubleClick += gcnew System::EventHandler(this, &Form1::label1_dbClick);
```

Vous pouvez télécharger les sources du projet Hello World à cette adresse : [Télécharger ici \(10 ko\)](#)

## 4. Les contrôles

### 4.1. Généralités

A chaque contrôle, que ce soit la form ou bien ce qui la compose, est associée une liste de propriétés. Les plus communes sont les propriétés qui permettent de définir la taille ou le positionnement du contrôle (**location (x,y), Size (width, height), etc ...**), le texte affiché d'un contrôle (**text, value**), si le contrôle est visible, activé, etc ... (**visible, enabled**), ou encore les couleurs et la font du contrôle (**font, forecolor, backcolor, etc ...**), et d'autres ...

Chaque contrôle dispose aussi de propriétés qui sont propres à son fonctionnement (par exemple la propriété **step** de la barre de progression qui définit le pas de progression).

Détailler toutes les propriétés de chaque contrôle de manière exhaustive serait laborieux et peu intéressant dans ce tutoriel. Dans la suite du paragraphe, je vais donc vous décrire brièvement l'utilité des contrôles standards.

Pour connaître en détail la liste des propriétés d'un contrôle, je vous invite à consulter la fenêtre de propriétés (bien souvent, le nom suffit à comprendre l'utilité de la propriété) ou à consulter la MSDN.

Chaque contrôle réagit aussi à un certain nombre d'évènements (click de la souris (droit ou gauche), double click, appui sur une touche, notification de changement de valeur, etc ...). Je ne vous les présenterai pas non plus de manière exhaustive. Sachez que vous les retrouverez dans la fenêtre de propriétés en cliquant sur le bouton "event", comme je l'ai montré un peu plus haut. Vous trouverez de même plus de précisions dans la MSDN. Le fonctionnement global est le même que celui que je vous ai décrit en particulier dans le programme hello world.

### 4.2. La form

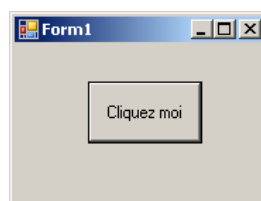
La form, c'est tout simplement la fenêtre. C'est sur celle-ci que vous allez placer les différents contrôles. Elle pourra éventuellement être redimensionnable ou disposer des boutons minimiser et maximiser.

Lorsqu'on a beaucoup programmé avec l'API Win32 ou les MFC, on parle beaucoup de boîte de dialogue. Il s'agit tout simplement d'une form. Ce sont simplement les conditions d'appel et l'apparence qui vont changer (par exemple le `borderStyle` à `FixedToolWindow`). Il vous faudra gérer le passage de paramètre et la récupération de résultat.

Plus de détails sur les Forms

### 4.3. Le bouton

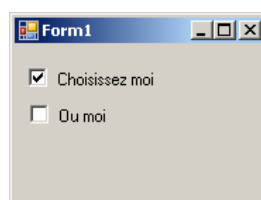
Un grand classique : le bouton. Très utilisé car très intuitif, il permet de déclencher une action, généralement par un click avec le bouton gauche.



Plus de détails sur les boutons

### 4.4. CheckBox

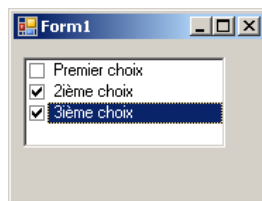
La case à cocher dérive du bouton (plus précisément de la classe `ButtonBase`) et est similaire en beaucoup de points. Elle permet en général de faire un choix, simple ou multiple avec le click gauche de la souris. Ils sont souvent grisés lorsque le choix est inaccessible.



Plus de détails sur les CheckBox

## 4.5.CheckedListBox

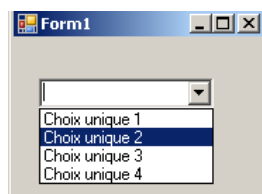
La liste à cocher permet une sélection multiple facilitée, et proposant les mêmes fonctionnalités qu'une listbox. Un point intéressant par rapport au checkbox classique, c'est que ce contrôle est scrollable.



Plus de détails sur les CheckedListBox

## 4.6.ComboBox

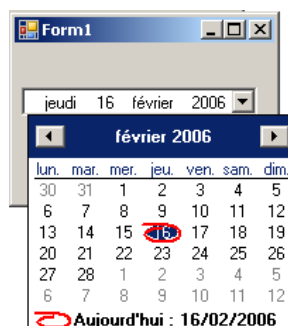
La liste déroulante est une combinaison d'une listbox et d'un textbox. Très utilisée en cas de choix unique dans une liste de choix, son fonctionnement est élégant et rapide.



Plus de détails sur les ComboBox

## 4.7.DateTimePicker

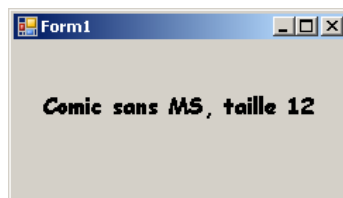
Le contrôle DateTimePicker est utilisé pour permettre à l'utilisateur de sélectionner une date et une heure, et pour afficher cette valeur date et heure au format spécifié.



Plus de détails sur le DateTimePicker

## 4.8.Label

Le label est le contrôle par excellence pour afficher du texte. On pourrait croire à son nom qu'il ne peut afficher que du texte statique. Au contraire, il sait aussi bien afficher du texte dynamique. Il sert couramment à donner un intitulé ou un titre à un autre contrôle.



Plus de détails sur les Labels

## 4.9.LinkLabel

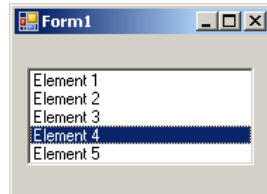
Le contrôle LinkLabel est analogue à un contrôle Label, à la différence qu'il peut afficher un lien hypertexte.



Plus de détails sur les LinkLabels

#### 4.10. ListBox

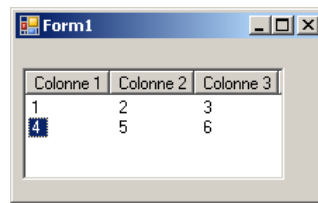
Le contrôle ListBox permet d'afficher une liste d'éléments dans laquelle l'utilisateur peut sélectionner un ou plusieurs éléments avec la souris.



Plus de détails sur les ListBox

#### 4.11. ListView

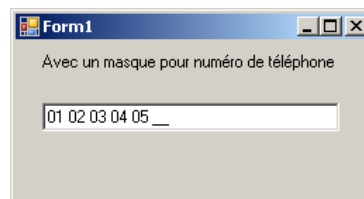
Le listview est un contrôle très connu, utilisé pour afficher des listes d'éléments. Très connu car c'est le premier qu'on voit lorsqu'on utilise l'explorateur de windows. Il permet aussi d'afficher des icônes.



Plus de détails sur les ListView

#### 4.12. MaskedTextBox

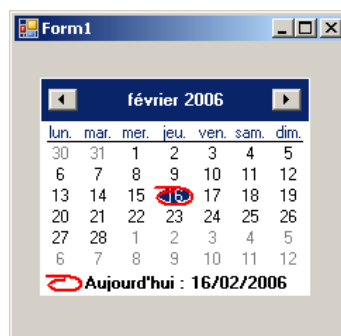
Le MaskedTextBox est un contrôle d'édition qui permet à des développeurs de définir des masques de chaînes pour contrôler le format de sortie de la valeur saisie.



Plus de détails sur les MaskedTextBox (en anglais)

#### 4.13. MonthCalendar

Le contrôle MonthCalendar permet à l'utilisateur de sélectionner une date et une heure à l'aide d'un affichage visuel.



Plus de détails sur les MonthCalendar

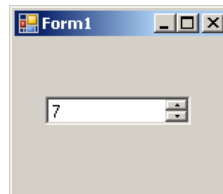
## 4.14.NotifyIcon

La classe NotifyIcon permet de programmer des processus, non dotés d'interfaces avec lesquels on interagit depuis la barre système.

Plus de détails sur les NotifyIcon

## 4.15.NumericUpDown

Le contrôle NumericUpDown permet d'incrémenter ou de décrémenter une valeur numérique à l'aide de boutons (flèche vers le haut ou vers le bas).



Plus de détails sur les NumericUpDown

## 4.16.PictureBox

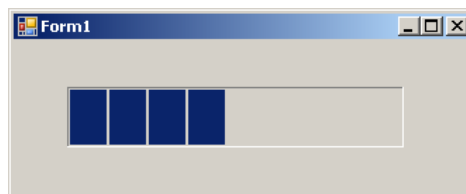
On se sert principalement du contrôle PictureBox pour afficher des images bitmap, icône, JPEG, GIF, PNG.



Plus de détails sur les PictureBox

## 4.17.ProgressBar

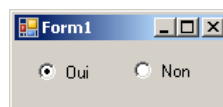
Il s'agit là de la barre de progression classique et bien connue des logiciels qui ont besoin de charger des informations.



Plus de détails sur les ProgressBar

## 4.18.RadioButton

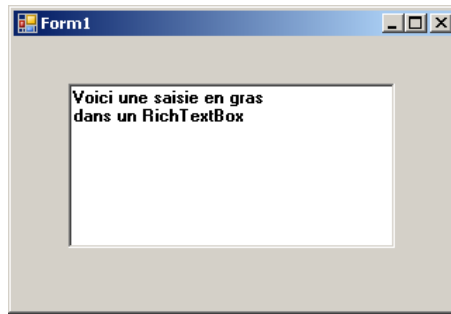
Le bouton radio est très utilisé lorsqu'il s'agit de faire une sélection unique parmi plusieurs choix.



Plus de détails sur les RadioButtons

## 4.19.RichTextBox

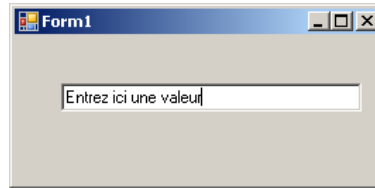
Le contrôle RichTextBox est une extension du textbox, qui permet une saisie avancée avec une gestion du format RTF (pour avoir du gras, souligné, etc ...).



Plus de détails sur les RichTextBox

## 4.20. TextBox

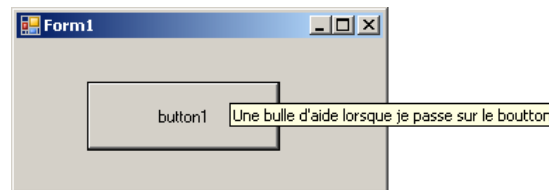
Le contrôle TextBox est le contrôle par excellence qui permet la saisie, sur une seule ligne ou multiligne pour des longs textes.



Plus de détails sur les TextBox

## 4.21. ToolTip

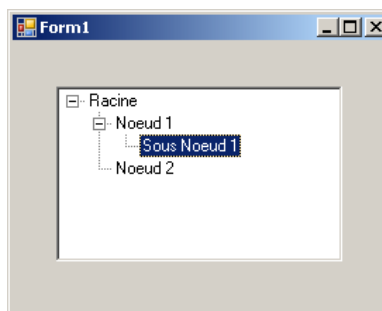
On se sert du contrôle ToolTip lorsqu'on veut afficher des bulles d'aides, afin d'afficher des renseignements supplémentaires lors du passage de la souris sur une zone.



Plus de détails sur les ToolTips

## 4.22. TreeView

Le contrôle TreeView permet d'afficher des éléments sous une forme arborescente. Très utilisé, avec ses noeuds cliquables et déroulables, il est au centre de beaucoup d'application, comme dans l'explorateur de fichiers de windows.



Plus de détails sur les TreeView

## 4.23. WebBrowser

Le contrôle WebBrowser, comme son nom l'indique, va vous permettre d'afficher très facilement des pages html.



Plus de détails sur les WebBrowser

## 4.24. Les autres contrôles ...

Nous avons fait le tour des "commons controls", sachez qu'il en existe d'autres par défaut et que bien sur, vous pouvez ajouter vos propres contrôles.

En vrac, les plus utilisés sont :

- les groupbox pour grouper des contrôles.
- Les panels pour regrouper des collections de contrôles.
- Les splitters pour séparer votre form en plusieurs parties redimensionnables.
- Les tabcontrols pour faire des onglets.
- Les timers.
- Les boites de dialogue pour ouvrir ou enregistrer des fichiers.
- etc ...

## 5. Le programme d'exemple

### 5.1. Présentation

Pour illustrer certains de ces contrôles, je vais vous accompagner dans la création d'une petite application.

Le principe de cette application est simple, il s'agit de proposer une interface utilisateur pour envoyer des "net send", ces messages visuels gérés par windows, qu'on a tous utilisé en salle d'informatique...

Dans le réseau de mon entreprise, faire la liste des ordinateurs du réseau et permettre la sélection aisée d'un ou de plusieurs d'entre eux pour envoi. Nous saisisons de même le message ainsi que le nom de l'expéditeur, c'est à dire nous ...

Enfin, nous aurons un récapitulatif de la réussite ou de l'échec de l'envoi du net send.

### 5.2. Elaboration de l'application

#### 5.2.1. Chargement de l'application, WinForm d'attente avec une barre de progression.

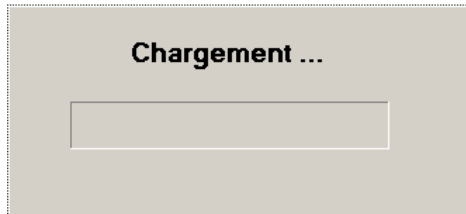
Commençons par charger nos informations. En l'occurrence, ici nous allons récupérer la liste des ordinateurs de notre groupe de travail. Et quoi de mieux pour faire patienter notre utilisateur qu'une barre de progression ?

Dans le réseau de mon entreprise, faire la liste des ordinateurs du groupe de travail peut s'avérer assez long... Donc, visuellement, l'effet de la barre de progression est satisfaisant. Si vous faites l'essai chez vous, il est probable que ça n'ai pas autant d'effet (sauf si vous avez un parc informatique impressionnant...). N'hésitez pas à rajouter un petit Sleep() pour avoir un effet visuel plus intense ... juste pour le plaisir des yeux !

Créons alors une nouvelle classe dérivée de Winforms. Add --> New Item --> UI / Windows Form, que nous appelons frm\_loading.

Réglez les propriétés de la form : StartPosition à CenterScreen et FormBorderStyle à None.

Placez-y une progressBar dessus et un label, comme ci dessous (modifiez les propriétés font et text du label).



Rajouter les fonction ci dessous :

```

System::Collections::ArrayList ^ getListOfComputer ()
{
    return listOfComputer;
}
private:
void ThreadProcess(void)
{
    for each(System::DirectoryServices::DirectoryEntry ^ entry in entries)
    {
        try
        {
            System::Net::IPHostEntry ^ ip = System::Net::Dns::GetHostEntry(entry->Name);
            listOfComputer->Add(entry->Name);
        }
        catch (System::Exception ^ ex)
        {
        }
        this->progressBar1->PerformStep();
    }
    this->Close();
}

private: System::Void frm_loading_Load(System::Object^ sender, System::EventArgs^ e)
{
    listOfComputer = gcnew System::Collections::ArrayList();

    System::String ^ domainName = "WORKGROUP";
    System::DirectoryServices::DirectoryEntry ^ domainEntry =
    gcnew System::DirectoryServices::DirectoryEntry(System::String::Format("WinNT://{0}", domainName));
    domainEntry->Children->SchemaFilter->Add("computer");
    entries = domainEntry->Children;
    int nb = 0;
    for each(System::DirectoryServices::DirectoryEntry ^ entry in entries)
        nb++;

    this->progressBar1->Minimum = 0;
    this->progressBar1->Maximum = nb;
    this->progressBar1->Value = 0;
    this->progressBar1->Step = 1;

    System::Threading::Thread ^ t =
    gcnew System::Threading::Thread(gcnew System::Threading::ThreadStart(
        this, &NetSendDotNet::frm_loading::ThreadProcess));

    t->Start();
}

```

Comme il n'est pas bon de figer l'application lorsque l'on effectue une tâche longue, nous allons effectuer le traitement dans un thread (je ne vais pas rentrer en détails dans l'élaboration d'un Thread, mais ici le code est assez simple, on lui passe l'adresse de la fonction à paralléliser).

Nous disposons de deux variables membre de la form :

```

private: System::Collections::ArrayList ^ listOfComputer;
private: System::DirectoryServices::DirectoryEntries ^ entries;

```

Un arraylist qui contient la liste des ordinateurs que nous allons retourner et une variable de type DirectoryEntries qui va nous servir pour énumérer les machines du domaine "Workgroup".

Dans le form\_load, on s'occupe de faire les initialisations de la barre de progression et de lancer notre Thread.

On commence par établir le nombre de machines. On boucle sur un for each pour avoir le nombre de machines. Cette fonction n'est bien sur pas optimisée, devoir parcourir deux fois la liste est une mauvaise méthode. Mais le but ici étant de montrer l'utilisation d'une progressbar, vous ne m'en tiendrez pas rigueur ...

Ensuite, nous initialisons la barre de progression, sachant que nous définissons un intervalle de progression entre **minimum** et **maximum** pour un pas (**step**) de 1. La valeur de départ (**value** donne la valeur en cours) étant bien entendu 0.

Le thread s'exécute alors ensuite, nous parcourons une nouvelle fois notre liste, en ajoutant à chaque fois le nom à notre ArrayList, tout en incrémentant la barre de progression (**PerformStep()**). Notez l'utilisation de **GetHostEntry** pour vérifier que la machine est bien connectée au réseau et disponible pour un envoi de NetSend.

Enfin, on ferme la fenêtre. On note aussi la présence de la fonction getListOfComputer qui nous retournera la liste.

Remarque : On utilise **DirectoryEntry** pour lister les machines du domaine : WinNt://WORKGROUP. Pour utiliser DirectoryEntry, il faudra rajouter une référence vers System::DirectoryServices.

### 5.2.2.Construction de la fenêtre principale.

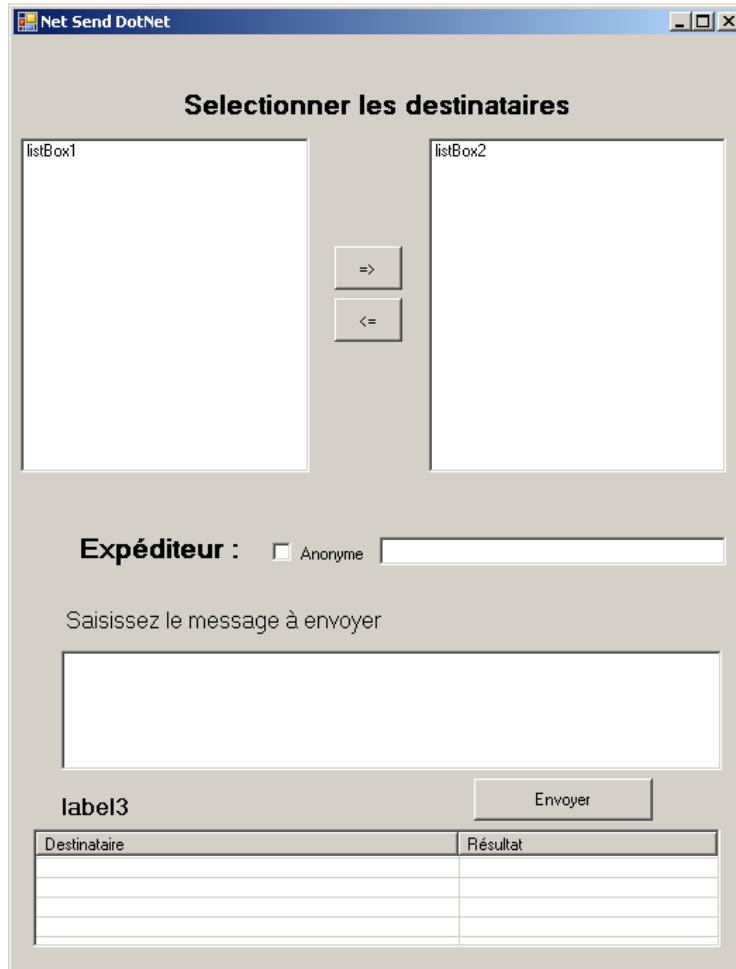
Maintenant, il est temps de passer à la création de la fenêtre principale.

- Modifier la propriété StartPosition de la form à CenterScreen pour que la fenêtre soit centrée par rapport à l'écran. Changer aussi le

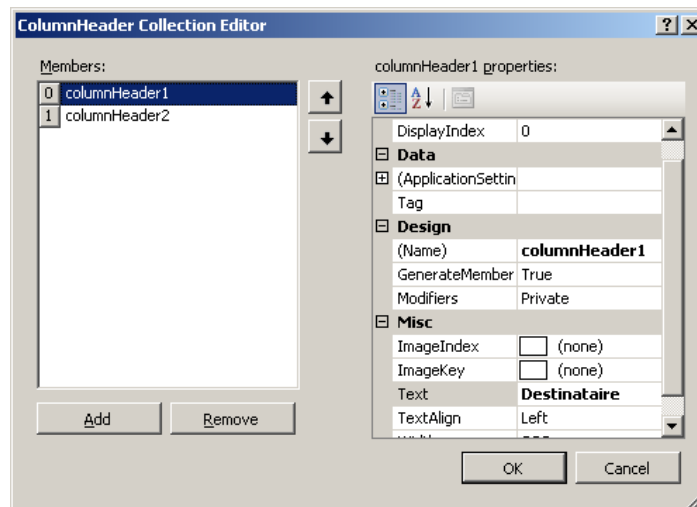
titre (propriété text) pour "NetSend DotNet".

- Nous allons maintenant reproduire le fonctionnement bien connue de deux listes qui s'alimentent l'une l'autre. Dessiner deux listBox, puis deux boutons comme sur l'image ci-dessous. Profitez-en pour changer la propriété text des boutons pour => et <=.
- Changez la propriété SelectionMode des listBox et passez là à MultiSimple pour autoriser une sélection multiple. Rajoutez de même un label pour faire le titre, en changeant sa propriété text.
- Rajoutez maintenant un label, un checkbox et un textbox pour saisir l'expéditeur.
- Ajoutez un label ainsi qu'un textbox avec la propriété multiline à true pour la saisie du message à envoyer.
- Rajoutez aussi un bouton envoyer et un label pour afficher la progression.
- Ajoutez aussi un tooltip, vous verrez qu'il se place sur une fenêtre en dessous.
- Enfin, nous ajoutons une ListView dans laquelle nous afficherons les résultats de l'envoi. (Positionnez la propriétés view à détails et gridline à true).

Vous devriez obtenir une fenêtre comme ci dessous :



NB : Pour définir les colonnes, j'ai utilisé la propriété Columns, et j'ai défini les deux colonnes comme ci-dessous :



Passons maintenant au code :

Générer l'évènement form\_load pour la form (soit en double-cliquant sur la form, soit dans le panneau "events" de la fenêtre de propriétés). C'est ici que nous allons faire les initialisations de la fenêtre.

```
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
    frm_loading ^ frm = gcnew frm_loading;
    frm->ShowDialog();
    System::Collections::ArrayList ^ listComputer = frm->getListOfComputer();
    label3->Text = "";

    listBox1->Items->Clear();
    listBox2->Items->Clear();
    for each (System::String ^ machine in listComputer)
        listBox1->Items->Add(machine);
    textBox1->Text = System::Environment::UserName;
}

```

Nous instancions en premier lieu notre fenêtre de chargement, nous l'affichons via la méthode ShowDialog().

Notez au passage que nous avons créé une boîte de dialogue comme une form classique, et que c'est l'appel de cette fonction qui provoque son comportement de boîte de dialogue.

Nous ne nous occupons pas de savoir s'il y a une valeur de retour à l'appel de la boîte de dialogue. Nous nous contentons de récupérer la liste des noms des ordinateurs, ceci étant possible car l'objet n'est pas détruit et les ressources non libérées (avec la méthode Dispose()).

Ensuite, on positionne le label d'information de progression à chaîne vide, puis on efface les contenus des deux listboxs. Ensuite, on parcourt la liste et on ajoute les noms à la première liste.

Enfin, on affecte un nom d'expéditeur par défaut.

Maintenant, nous allons gérer le comportement de la double liste, qui fait passer un ou plusieurs éléments de la liste de gauche à la liste de droite quand on appuie sur => et inversement lorsqu'on appuie sur <=

Ajoutons donc un événement sur le click de chacun de ces boutons.

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    System::Collections::ArrayList ^listItem = gcnew System::Collections::ArrayList();
    for each (String ^ item in listBox1->SelectedItems)
        listItem->Add(item);
    for each (String ^ item in listItem)
    {
        listBox1->Items->Remove(item);
        listBox2->Items->Add(item);
    }
}
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e)
{
    System::Collections::ArrayList ^listItem = gcnew System::Collections::ArrayList();
    for each (String ^ item in listBox2->SelectedItems)
        listItem->Add(item);
    for each (String ^ item in listItem)
    {
        listBox2->Items->Remove(item);
        listBox1->Items->Add(item);
    }
}

```

On boucle donc sur listBox->SelectedItems pour avoir la liste des éléments sélectionnés. Puis on les supprime de l'un pour les ajouter à l'autre.

On ne peut pas faire cette opération en un parcours car sinon, en faisant un remove, on modifie le contenu de la liste qu'on est en train de parcourir. Ce qui fait que le programme ne peut plus savoir ce qu'il fait, et sur quel élément il travaille. D'où l'utilisation d'un ArrayList temporaire.

Maintenant, traitons l'évènement CheckedChanged sur la checkbox.

```
private: System::Void checkBox1_CheckedChanged(System::Object^ sender, System::EventArgs^ e)
{
    if (checkBox1->Checked)
        textBox1->Text = "Anonyme";
    else
        textBox1->Text = System::Environment::UserName;
    textBox1->Enabled = !checkBox1->Checked;
}

```

Le code parle de lui même. On positionne le nom "Anonyme" lorsque la checkbox est cochée, sinon au nom de l'utilisateur. On rend en plus le contrôle inactif si la case à cocher est cochée.

Maintenant, on va ajouter une bulle d'aide sur le bouton "envoyer". Générer l'évènement MouseHover sur le bouton, et appelons dedans la méthode Show du tooltip en lui passant en paramètres le message à afficher ainsi que le contrôle où il doit s'afficher, c'est à dire le bouton.

```
private: System::Void button3_MouseHover(System::Object^ sender, System::EventArgs^ e)
{
    tooltip1->Show("Cliquez ici pour envoyer", this->button3);
}

```

Ainsi, nous avons la bulle d'aide qui s'affiche, lorsque la souris passe sur le bouton (MouseHover).

Il ne nous reste plus que l'envoi.

### 5.2.3. La classe CMYNetSend.

Je vais faire une classe managée, qui va me permettre d'envoyer le net send en utilisant une API Win32.

Ajoutons donc une classe C++ (add class -> C++ -> C++ class). Appelons là CMyNetSend.

Rajoutons ensuite une fonction à la classe et les entêtes. Le fichier .h ressemblera ainsi à ça :

```
#pragma once

#include <windows.h>
#include <lm.h>
#include <stdio.h>
#include <stdlib.h>

#pragma comment(lib, "Netapi32.lib")

public ref class CMyNetSend
{
public:
    CMyNetSend(void);
    bool Send(LPCWSTR, LPCWSTR, LPCWSTR);
public:
    ~CMyNetSend(void);
};
```

On inclut les .h du platform SDK nécessaires (windows.h et lm.h), puis on lie par pragma aussi la librairie nécessaire à l'utilisation de l'API "NetSend".

Enfin, dans le .cpp, le code de la fonction, à savoir :

```
bool CMyNetSend::Send(LPCWSTR aQui, LPCWSTR monMessage, LPCWSTR nomSource)
{
    NetMessageNameAdd(NULL, nomSource);

    DWORD rc = NetMessageBufferSend(NULL, aQui, nomSource, (BYTE *)&monMessage[0], wcslen(monMessage) * 2);
    if (rc != NERR_Success) // on réessaie d'envoyer sans nom personnalisé en cas d'échec
        rc = NetMessageBufferSend(NULL, aQui, NULL, (BYTE *)&monMessage[0], wcslen(monMessage) * 2);

    return (rc==NERR_Success);
}
```

Lorsque l'on veut compiler, on rencontre une erreur, qui peut être celle-ci par exemple :

```
'IDataObject' : ambiguous symbol error
```

Ce problème vient de l'include de Windows.h, qui définit lui aussi un IDataObject, qui est une interface COM.

Pour corriger ce problème, on a deux possibilités :

- Soit en précisant explicitement le namespace que l'on utilise, pour éviter que le compilateur ne sache pas quel IDataObject utiliser. Concrètement, cela revient à enlever tous les using des .h pour les mettre dans les .cpp.
- Soit en définissant WIN32\_LEAN\_AND\_MEAN, qui a pour but d'exclure tous les entêtes Win32 qui ne sont pas utiles.

J'ai choisi cette deuxième méthode, je rajoute donc avant mon premier include :

```
#define WIN32_LEAN_AND_MEAN
```

Voilà pour l'objet CMyNetSend. Il ne nous reste plus qu'à gérer le click sur le bouton envoyer :

#### 5.2.4.L'envoi.

```

if (listBox2->Items->Count == 0)
{
    label3->ForeColor = System::Drawing::Color::Red;
    label3->Text = "Veuillez saisir au moins un destinataire";
    return;
}
if (textBox2->Text->Length == 0)
{
    label3->ForeColor = System::Drawing::Color::Red;
    label3->Text = "Veuillez saisir le message";
    return;
}
if (textBox1->Text->Length == 0)
{
    label3->ForeColor = System::Drawing::Color::Red;
    label3->Text = "Veuillez saisir l'expéditeur";
    return;
}
label3->Text = "Envoi en cours...";
Update();
listView1->Items->Clear();

for each (System::String ^ toWhom in listBox2->Items)
{
    CMyNetSend ^ myNetSend = gcnew CMyNetSend();

    System::IntPtr p1 = System::Runtime::InteropServices::Marshal::StringToHGlobalUni(toWhom);
    LPCWSTR str1 = reinterpret_cast<LPCWSTR>(static_cast<void *>(p1));
    System::IntPtr p2 = System::Runtime::InteropServices::Marshal::StringToHGlobalUni(textBox2->Text);
    LPCWSTR str2 = reinterpret_cast<LPCWSTR>(static_cast<void *>(p2));
    System::IntPtr p3 = System::Runtime::InteropServices::Marshal::StringToHGlobalUni(textBox1->Text);
    LPCWSTR str3 = reinterpret_cast<LPCWSTR>(static_cast<void *>(p3));

    System::Windows::Forms::ListItem ^ item = gcnew System::Windows::Forms::ListItem(toWhom);
    if (myNetSend->Send(str1, str2, str3))
        item->SubItems->Add("Ok");
    else
        item->SubItems->Add("Erreur");
    listView1->Items->Add(item);

    System::Runtime::InteropServices::Marshal::FreeHGlobal(p1);
    System::Runtime::InteropServices::Marshal::FreeHGlobal(p2);
    System::Runtime::InteropServices::Marshal::FreeHGlobal(p3);
}
label3->Text = "Envoi terminé";

```

Au début de cette fonction, on vérifie si tous les paramètres sont bien renseignés. Si ce n'est pas le cas, on affiche un message en rouge, dans le label d'information (propriété ForeColor).

On efface la listView de résultat et on boucle sur la liste des machines présentes dans la listBox de destination.

On instancie notre classe managée CMyNetSend (avec gcnew) et on convertit les valeurs à envoyer en unicode grâce l'objet Marshal. (en effet, rappelez vous, on utilise des LPCWSTR en paramètres de l'API NetMessageBufferSend).

Puis on ajoute le résultat dans la listView en fonction du bon retour ou non de la méthode Send de notre objet.

### 5.2.5.Conclusion et téléchargement.

Voilà, nous avons terminé le développement et vous savez tout sur notre petite application. Vous voyez comme le designer de visual studio nous a bien aidé pour gérer les contrôles et les événements et faire la génération du code laborieux.

J'espère que vous aurez bien compris ce petit exemple, et que vous en avez profité pour disséquer le code afin de voir comment fonctionnent les différents contrôles que j'ai utilisé.

Vous pouvez télécharger le programme d'exemple à cette adresse : [Télécharger ici \(21 ko\)](#)

Remarque : vous pouvez constater dans mon code que je n'ai pas changé les noms de contrôles (label1, label2, etc ...). Il est en général conseillé de donner un nom plus explicite à ses contrôles (ex : labelPourAfficherLeResultatIntermediaire).

## Suite

N'hésitez pas à aller consulter la deuxième partie de cet article sur la programmation des winforms.  
[Aller au tutoriel sur les Winforms - partie 2](#)

## Remerciements

Je remercie toute l'équipe C++ pour leur relecture attentive du document.

## Contact

Si vous constatez une erreur dans le tutorial, dans le source, dans la programmation ou pour toutes informations, n'hésitez pas à me contacter par mail, ou par le forum.

---

Les sources présentées sur cette page sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 Nico-pyright(c). Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts. Cette page est déposée à la SACD.

**Responsable bénévole de la rubrique C++ : Philippe Dunki - Contacter par email**

Vos questions techniques : **forum d'entraide C++** - Publiez vos articles, tutoriels et cours  
et rejoignez-nous dans l'équipe de rédaction du club d'entraide des développeurs francophones  
Nous contacter - Hébergement - Participez - Copyright © 2000-2010 www.developpez.com - Legal informations.

