

# Création d'un contrôle Winform avec Delphi 8 .NET

par [Nono40](#)

Date de publication : 01/06/2004

Dernière mise à jour : 01/06/2004

Ce document présente la création d'un contrôle WinForm personnalisé.

## Introduction

I - Création de l'assemblage

II - Ajout de la propriété AiguilleSec

III - Ajout du code de dessin

IV - Surcharge de la taille par défaut

V - Ajout d'un événement

VI - Utilisation du composant dans un projet Delphi .NET

VII - Utilisation du composant dans un projet C#

Conclusion

## Introduction

Cet article va présenter la création d'un composant WinForm personnalisé dérivant du type **System.Windows.Forms.UserControl**. Ce composant est une horloge analogique avec ou sans l'aiguille des secondes :

La méthode proposée ici est l'adaptation à Delphi 8 de l'article de [CGI](#) : [Création d'un contrôle WinForm pour .Net](#)

## I - Création de l'assemblage

La notion de paquet en tant que tel n'existe plus dans Delphi 8. Ceux-ci ont été remplacés par la définition d'assemblage introduite par .NET. Pour créer un composant WinForm utilisable il faut créer un nouvel assemblage contenant ce composant.

Pour créer un nouvel assemblage pour le composant, utilisez Fichier->Nouveau->Paquet. Enregistrez le projet comme PHorloge.bdsprj.

Ensuite ajouter un contrôle Winform avec Fichier->Nouveau->Contrôle utilisateur. Enregistrer le code comme UHorloge.pas. Dans l'éditeur de propriété qui vient de s'ouvrir renommez ( propriété Name ) le composant en THorloge.

Notez que Delphi fait alors directement dériver le composant de la classe **System.Windows.Forms.UserControl** et crée dans l'unité tout le code minimum de gestion du composant.

Le gestionnaire de projet doit alors vous donner les informations suivantes :

## II - Ajout de la propriété AiguilleSec

La propriété AiguilleSec va définir si l'aiguille des secondes est affichée ou non. La définition d'une nouvelle propriété est identique à la méthode utilisée dans la VCL. Il faut ajouter une variable privée contenant la valeur de la propriété et définir les méthodes éventuelles de lecture et écriture de la propriété.

Pour ceci il faut ajoutez dans la définition du composant les ligne suivantes :

```
private
{ Private Declarations }
FAiguilleSec:Boolean;
Procedure SetAiguilleSec(Value:Boolean);
public
constructor Create;
published
[Category('Appearance'),
Description('Affiche l'aiguille des secondes.'),
DefaultValue(true)]
Property AiguilleSec:Boolean Read FAiguilleSec Write SetAiguilleSec;
end;
```

Notez ici la définition des attributs de la propriété permettant de définir la catégorie d'affichage ainsi que le texte d'aide associé dans l'inspecteur d'objet.

Dans le constructeur du composant, ajouter la valeur initiale de la propriété :

```
constructor THorloge.Create;
begin
inherited Create;
//
// Required for Windows Form Designer support
//
InitializeComponent;
//
// TODO: Add any constructor code after InitializeComponent call
//
FAiguilleSec := True;
end;
```

Et pour terminer, définir le code de la méthode d'écriture de la propriété, l'appel d'invalidate va demander au contrôle de se redessiner.

```
procedure THorloge.SetAiguilleSec(Value: Boolean);
begin
FAiguilleSec:=Value;
Invalidate;
end;
```

### III - Ajout du code de dessin

Le dessin devant être effectué toutes les secondes, nous allons ajouter un Timer au contrôle afin d'obtenir un événement régulier. Le dessin en lui-même sera assuré par la surcharge de la méthode OnPaint du contrôle.

La définition de la classe devient donc :

```

THorloge = class(System.Windows.Forms.UserControl)
//....
strict protected
  /// <summary>
  /// Clean up any resources being used.
  /// </summary>
  procedure Dispose(Disposing: Boolean); override;
  procedure OnPaint(e:PaintEventArgs); Override;
private
  { Private Declarations }
  T:Timer;
  FAiguilleSec:Boolean;
  Procedure SetAiguilleSec(Value:Boolean);
  Procedure Tempo(myObject:System.Object; myEventArgs:System.EventArgs );
public
  constructor Create;
published
  [Category('Appearance'),
  Description('Affiche l'aiguille des secondes.'),
  DefaultValue(true)]
  Property AiguilleSec:Boolean Read FAiguilleSec Write SetAiguilleSec;
end;

```

Comme d'habitude le composant enfant sera créé dans le constructeur de l'objet.

```

constructor THorloge.Create;
begin
  inherited Create;
  //
  // Required for Windows Form Designer support
  //
  InitializeComponent;
  //
  // TODO: Add any constructor code after InitializeComponent call
  //
  FAiguilleSec := True;
  SetStyle(ControlStyles.DoubleBuffer Or ControlStyles.UserPaint Or
  ControlStyles.AllPaintingInWmPaint, true);
  T:=Timer.Create;
  Include(T.Tick,Tempo);
  T.Interval := 1000;
  T.Start;
end;

```

Notez ici l'ajout d'un gestionnaire d'événement ( Tempo dans notre exemple ) à la liste des gestionnaire associés à T.Tick. Cet ajout est réalisé à l'aide de la procédure Include, ce point est différent de la surcharge habituelle d'un événement d'un contrôle VCL.

Notez aussi la modification du style du contrôle afin de définir un dessin personnalisé et un double-buffer pour éviter le scintillement lors du tracé.

Le code de la procédure Tempo ne fera qu'invalider la zone d'affichage du composant :

```
procedure THorloge.Tempo(myObject: TObject;  
  myEventArgs: System.EventArgs);  
begin  
  Invalidate;  
end;
```

L'événement OnPaint se chargera du dessin du composant :

```
procedure THorloge.OnPaint(e: PaintEventArgs);  
Var BlackPen :Pen;  
  Rect :Rectangle;  
  d :DateTime;  
  Pinceau :SolidBrush;  
  drawFont :System.Drawing.Font;  
  n,z,u,x0,y0,x1,y1,x2,y2,x3,y3:Single;  
  aigPoints : Array[0..3]Of PointF;  
begin  
  BlackPen := Pen.Create(Color.Black, 1);  
  Rect := Rectangle.Create( 2, 2, 118, 118);  
  e.Graphics.DrawEllipse(blackPen, rect);  
  
  d := DateTime.Now;  
  
  //-----  
  // Aiguille des Secondes (si propriété AiguilleSec à true)  
  if AiguilleSec Then  
  Begin  
    n := d.Second*200/60;  
    z := n/100*3.14159;  
    u := (n+50)/100*3.14159;  
  
    x0 := Math.Sin(z)*50;  
    y0 := -Math.Cos(z)*50;  
  
    x1 := -Math.Sin(z)*10;  
    y1 := Math.Cos(z)*10;  
  
    x2 := Math.Sin(u)*2;  
    y2 := -Math.Cos(u)*2;  
  
    x3 := -Math.Sin(u)*2;  
    y3 := Math.Cos(u)*2;  
  
    Pinceau:=SolidBrush.Create(Color.Red);  
  
    aigPoints[0].X := x1+60;  
    aigPoints[0].Y := y1+60;  
  
    aigPoints[1].X := x2+60;  
    aigPoints[1].Y := y2+60;  
  
    aigPoints[2].X := x0+60;  
    aigPoints[2].Y := y0+60;  
  
    aigPoints[3].X := x3+60;  
    aigPoints[3].Y := y3+60;
```

```
e.Graphics.FillPolygon(Pinceau, aigPoints);
e.Graphics.DrawPolygon(blackPen, aigPoints);
Pinceau.Dispose();
End;
```

```
//-----
// Aiguille des Minutes
```

```
n := d.Minute*200/60;
z := n/100*3.14159;
u := (n+50)/100*3.14159;
```

```
x0 := Math.Sin(z)*50;
y0 := -Math.Cos(z)*50;
```

```
x1 := -Math.Sin(z)*10;
y1 := Math.Cos(z)*10;
```

```
x2 := Math.Sin(u)*4;
y2 := -Math.Cos(u)*4;
```

```
x3 := -Math.Sin(u)*4;
y3 := Math.Cos(u)*4;
```

```
Pinceau:=SolidBrush.Create(Color.Lime);
```

```
aigPoints[0].X := x1+60;
aigPoints[0].Y := y1+60;
```

```
aigPoints[1].X := x2+60;
aigPoints[1].Y := y2+60;
```

```
aigPoints[2].X := x0+60;
aigPoints[2].Y := y0+60;
```

```
aigPoints[3].X := x3+60;
aigPoints[3].Y := y3+60;
```

```
e.Graphics.FillPolygon(Pinceau , aigPoints);
e.Graphics.DrawPolygon(blackPen, aigPoints);
Pinceau.Dispose();
```

```
//-----
// Aiguille des Heures
```

```
n := d.Hour*200/12 + d.Minute*200/60/12;
z := n/100*3.14159;
u := (n+50)/100*3.14159;
```

```
x0 := Math.Sin(z)*35;
y0 := -Math.Cos(z)*35;
```

```
x1 := -Math.Sin(z)*10;
y1 := Math.Cos(z)*10;
```

```
x2 := Math.Sin(u)*4;
y2 := -Math.Cos(u)*4;
```

```
x3 := -Math.Sin(u)*4;
y3 := Math.Cos(u)*4;
```

```
Pinceau:=SolidBrush.Create(Color.Yellow);
```

```
aigPoints[0].X := x1+60;
```

```
aigPoints[0].Y := y1+60;

aigPoints[1].X := x2+60;
aigPoints[1].Y := y2+60;

aigPoints[2].X := x0+60;
aigPoints[2].Y := y0+60;

aigPoints[3].X := x3+60;
aigPoints[3].Y := y3+60;

e.Graphics.FillPolygon(Pinceau , aigPoints);
e.Graphics.DrawPolygon(blackPen, aigPoints);
Pinceau.Dispose();
//-----
// Chiffres
drawFont := System.Drawing.Font.Create('Arial', 8);
Pinceau := SolidBrush.Create(Color.Black);

e.Graphics.DrawString('12', drawFont, Pinceau, 55, 6);
e.Graphics.DrawString( '6', drawFont, Pinceau, 58, 101);
e.Graphics.DrawString( '3', drawFont, Pinceau, 105, 53);
e.Graphics.DrawString( '9', drawFont, Pinceau, 8, 53);
//-----
drawFont.Dispose();
Pinceau.Dispose();
blackPen.Dispose();
inherited;
End;
```

## IV - Surcharge de la taille par défaut

Afin que la taille du composant soit correcte ( 121 x 121 ) lors de sa création, il faut redéfinir la taille par défaut du composant par surcharge de la propriété taille :

```
THorloge = class(System.Windows.Forms.UserControl)
{$REGION 'Designer Managed Code'}
//...
strict protected
/// <summary>
/// Clean up any resources being used.
/// </summary>
procedure Dispose(Disposing: Boolean); override;
procedure OnPaint(e:PaintEventArgs); Override;
private
{ Private Declarations }
T:Timer;
FAiguilleSec:Boolean;
Function GetSize:Size;
Procedure SetAiguilleSec(Value:Boolean);
Procedure Tempo(myObject:System.Object; myEventArgs:System.EventArgs );
public
constructor Create;
published
Property Size Read GetSize;
[Category('Appearance'),
Description('Affiche l" aiguille des secondes.'),
DefaultValue(true)]
Property AiguilleSec:Boolean Read FAiguilleSec Write SetAiguilleSec;
end;
```

La méthode GetSize est alors définie comme suit :

```
function THorloge.GetSize: Size;
begin
Result := System.Drawing.Size.Create(121,121);
end;
```

## V - Ajout d'un événement

Un événement en dotnet comporte toujours deux paramètres, le premier est l'objet déclenchant (comme en VCL) le deuxième est un objet donnant la liste des valeurs spécifiques à l'événement. On va donc créer une nouvelle classe de description des paramètres de l'événement. Cette classe ne va donner ici que l'heure de déclenchement, ce n'est pas très utile, mais c'est juste pour l'exemple.

```

TReveilArgs = Class
Public
    DateHeureReveil : DateTime;
    Constructor Create(D:DateTime);
End;

TReveilEvent = Procedure ( Sender:TObject ; R : TReveilArgs )Of Object;

```

La date et l'heure de déclenchement de l'événement sera conservé dans une nouvelle propriété Reveil de type System.DateTime. L'écriture d'un gestionnaire d'événement est identique à celle utilisée dans la VCL.

Une propriété de type événement n'est plus déclarée avec les mots clef Read et Write ( bien que ce soit toujours possible par compatibilité ) mais par les mots clefs Add et Remove. Ceci permet de gérer les événements multicast de la CLR .NET.

```

THorloge = class(System.Windows.Forms.UserControl)
//...
private
{ Private Declarations }
    FReveil:DateTime;
    FReveilEvent:TReveilEvent;
//...
published
    [Category('Misc'),
    Description('Date/heure de déclenchement du réveil. '),
    DefaultValue(true)]
    Property Reveil:DateTime Read FReveil Write FReveil;
    [Category('Misc'),
    Description('Événement déclenché si Now=Reveil. '),
    DefaultValue(true)]
    Property OnReveil:TReveilEvent Add FReveilEvent Remove FReveilEvent;
end;

```

Bien sur, le code du timer va être modifié pour tester si l'heure de l'événement correspond et ainsi appeler le gestionnaire d'événement. Notez qu'il faut appeler la variable privée et non l'événement lui-même comme on le faisait en VCL.

```

procedure THorloge.Tempo(myObject: TObject;
myEventArgs: System.EventArgs);
begin
    Invalidate;
    If DateTime.Now.ToString=Reveil.ToString Then
        FReveilEvent(Self,TReveilArgs.Create(Reveil));
end;

```

## VI - Utilisation du composant dans un projet Delphi .NET

Une fois le projet compilé, installez le nouveau composant WinForm dans Delphi avec Composants->Composants .NET intallés. Puis en sélectionnant l'assemblage nouvellement créé et de définir la catégorie dans laquelle il faut ajouter le composant ( général par défaut ).

Après installation le composant apparait dans la liste :

Il suffit alors de poser le composant sur une fiche ( WinForm ) afin de l'utiliser comme n'importe quel autre composant.

## VII - Utilisation du composant dans un projet C#

Pour tester le composant dans une application C# il suffit de compiler l'exemple suivant avec le compilateur en ligne **csc**.

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace MonProjet
{
    public class WinForm : Form
    {
        private UHorloge.THorloge horloge1 = new UHorloge.THorloge();

        public WinForm()
        {
            horloge1.Location = new Point(20, 20);

            Size = new Size(170, 200);
            Text = "WinForm";

            Controls.Add(this.horloge1);
        }

        [STAThread]
        static void Main()
        {
            Application.Run(new WinForm());
        }
    }
}
```

Notez que l'espace de nom n'est pas celui de la dll mais celui de l'unité contenant le composant.

Lors de la compilation il faut donner en référence la dll contenant notre composant :

```
csc /target:winexe /reference:PHorloge.dll WinForm.cs
```

## Conclusion

La création d'un composant WinForm n'est donc pas si différente de la méthode utilisée habituellement pour les composants VCL sous Win32.

Source du composant : [Source0080](#)

Version PDF de cet article :

Miroir 1 : [Version PDF](#)

Dans le cas où le miroir 1 ne fonctionne pas :

Miroir 2 : [Version PDF](#)

Merci à [Pierre Castelain](#) pour la correction orthographique.