

# Utilisation de la classe NotifyIcon dans Delphi 8

par [Nono40](#)

Date de publication : 16/02/2004

Dernière mise à jour : 16/02/2004

Ce document présente l'utilisation du composant Winform NotifyIcon dans une application Delphi.NET. Cette présentation sera appliquée aux applications Winform et VCL.NET.

## Introduction

### I - Principes communs

I-A - Classe NotifyIcon

I-B - Utilisation de l'icône.

### II - Utilisation dans une application Winform

II-A - Mise en place des composants

II-B - Ajout du code en réponse aux événements

II-C - Exécution

### III - Utilisation dans une application VCL.NET

III-A - Création d'une instance de la classe NotifyIcon

III-B - Ajout des autres composants

III-C - Ajout du code en réponse aux événements

III-D - Exécution

## Conclusion

## Introduction

La classe NotifyIcon de l'assembly System.Windows.Forms permet de gérer très simplement une icône dans le systray. Elle remplace la fonction API Win32 **Shell\_NotifyIcon** en proposant une solution sous forme de composant.

Cet article va présenter son utilisation dans Delphi.NET pour des applications Winform et VCL.NET au travers d'une application simple.

## I - Principes communs

### I-A - Classe NotifyIcon

Le composant Winform NotifyIcon permet une gestion simple d'une icône dans la barre des tâches. La gestion de l'affichage est simplement effectuée par la mise à jour des propriétés :

- Text : Texte affiché sous forme de Hint quand la souris est au dessus de l'icône.
- Icon : Objet System.Drawing.Icon contenant le dessin de l'icône.
- ContextMenu : Objet System.Windows.Forms.ContextMenu contenant le menu contextuel.
- Visible : Tout simplement indique si l'icône doit être affichée ou non.

Pour la gestion des actions de l'utilisateur le composant possède des événements :

- Click : Déclenché quand l'utilisateur clique sur l'icône.
- DoubleClick : Déclenché quand l'utilisateur double-clique sur l'icône.
- MouseDown : Déclenché quand l'utilisateur appuie sur un bouton de la souris au-dessus de l'icône.
- MouseUp : Déclenché quand l'utilisateur relache un bouton de la souris au-dessus de l'icône.
- MouseMove : Déclenché quand l'utilisateur déplace la souris au-dessus de l'icône.

### I-B - Utilisation de l'icône.

Le programme d'exemple décrit plus loin va présenter les fonctions habituelles d'une icône dans le systray :

- Affichage d'une icône.
- Gestion des trois types de click : gauche, droit et double
- Gestion d'un menu contextuel

Le click droit va afficher le menu contexte contextuel contenant les options restaurer, réduire et fermer.

Le double clic va restaurer la fenêtre de l'application.

Le click gauche, non suivi d'un double clic affichera le menu contextuel.

La différenciation entre un click simple et le premier click d'un double click sera effectuée à l'aide d'un Timer. L'affichage du menu dans le cas d'un simple click gauche est retardé d'une demi-seconde pour attendre le deuxième click éventuel du double click.

## II - Utilisation dans une application Winform

### II-A - Mise en place des composants

Créer une nouvelle application Winform, et placer sur la fenêtre un composant NotifyIcon, un composant ContextMenu et un composant Timer. Tous ces composants sont inclus dans la palette "components".

Notez que dans le cas d'un fiche Winform les composants visuels ne sont pas dessinés sur la fiche mais placés dans un bandeau en dessous. Il suffit ensuite de cliquer sur le composant voulu pour en modifier les propriétés.

Cliquer sur le composant NotifyIcon1 et ajuster les propriétés voulues. Pour que l'icône soit affichée il faut ajouter une icône au composant ( ce qui est fait très simplement par le bouton '...' ) et mettre sa propriété visible à True.

Remplir de même les propriétés Text et ContextMenu en sélectionnant ContextMenu1.

Renseigner ensuite le menu contextuel, ceci est réalisé en cliquant sur le composant dans le bas pour faire apparaître un début de menu dans la fiche. Il faut alors remplir les éléments de menu comme pour une application VCL classique.

Remarque : renommer les éléments de menu avant d'y associer des événements. Car le nom des événements n'est pas modifié automatiquement ensuite. Il est tout de même possible de les changer manuellement.

Renseigner enfin les propriétés du timer : interval =500 et enabled à False.

### II-B - Ajout du code en réponse aux événements

En premier lieu renseigner les événements du menu contextuel :

```
procedure TWinForm.MenuReduire_Click(sender: System.Object; e: System.EventArgs);
begin
  // L'option réduire permet de ... réduire la fenêtre !
  WindowState:=FormWindowState.Minimized;
end;

procedure TWinForm.MenuRestaurer_Click(sender: System.Object; e: System.EventArgs);
begin
  // L'option restaurer permet de ... restaurer la fenêtre !
  If WindowState=FormWindowState.Minimized
  Then WindowState:=FormWindowState.Normal;
  Show;
end;

procedure TWinForm.MenuFermer_Click(sender: System.Object; e: System.EventArgs);
```

```
begin
  // Fermer la fenêtre
  Close;
end;
```

Renseigner ensuite l'événement MouseDown du NotifyIcon1 :

```
procedure TWinForm.NotifyIcon1_MouseDown(sender: System.Object; e: System.Windows.Forms.MouseEventArgs);
begin
  // Dans le cas d'un click gauche sur l'icone, on test si c'est le premier
  // click puis on lance le timer de retard d'affichage du menu.
  // Cette méthode permet de laisser le temps de double cliquer sur l'icone
  // avant que le menu n'apparaisse.
  If (e.Button=System.Windows.Forms.MouseButtons.Left) And
    (e.Clicks=1)
  Then
    Timer1.Enabled:=True;
end;
```

Notez que la structure des événements dans les objets .NET est toujours sur le même modèle. Deux objets sont passés en paramètres, le premier ( Sender ) est l'objet ayant reçu/provoqué l'événement. Le deuxième ( e ) contient des informations complémentaire. La classe de e dépend du type d'événement.

Dans le cas ci-dessus, e est de type **MouseEventArgs**. Les propriétés de cette classe permettent de différencier le type de bouton et le style d'appui. Il n'y a pas de propriété DoubleClick ou de type de bouton DoubleClick comme dans les événements VCL. Ici, c'est le nombre de click qu'il faut comparer.

Le bouton droit ne sera pas testé, car le menu contextuel est géré automatiquement par le composant lors du click droit.

Ajouter ensuite le code pour le double click :

```
procedure TWinForm.NotifyIcon1_DoubleClick(sender: System.Object; e: System.EventArgs);
begin
  // Dans le cas d'un double-click, on arrête le timer lancé par le premier
  // click et on effectue l'action par défaut.
  Timer1.Enabled:=False;
  MenuRestaurer_Click(Sender,e);
end;
```

Et enfin celui de l'événement Tick du Timer :

```
procedure TWinForm.Timer1_Tick(sender: System.Object; e: System.EventArgs);
begin
  // Si le timer s'est écoulé, c'est qu'il n'y a pas eu de double-click.
  // Dans ce cas on affiche le menu là où est la souris.
  Timer1.Enabled:=False;
  ContextMenu1.Show(Self,Control.MousePosition);
end;
```

## II-C - Exécution

L'application est maintenant prête à être essayée.

Le source complet de cet exemple en disponible ici : [Source0078](#)

### III - Utilisation dans une application VCL.NET

Dans le une applicaton VCL.NET le composant NotifyIcon ne peut pas être utilisé visuellement. L'instance sera donc créée et gérée dynamiquement.

#### III-A - Création d'une instance de la classe NotifyIcon

Pour pouvoir utiliser le composant NotifyIcon le projet à besoin de référencer l'assemblage qui le contient : System.Windows.Forms.dll . Il faudra aussi référencer System.Drawing.dll afin de pouvoir créer une instance de la classe System.Drawing.Icon.

Pour ajouter une référence aller dans le menu *Project* puis sélectionner *Add référence...*

Ajouter les références pour System.Windows.Forms.dll, System.Drawing.dll.

Il faut ensuite ajouter dans le uses les assembly référencées :

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, System.Windows.Forms, System.Drawing;
```

Ajouter ensuite la déclaration dans Form1 manuellement comme habituellement pour la création dynamique d'une composant. Il faut aussi ajouter les méthodes qui seront ajoutés aux événements.

```
TForm1 = class(TForm)
  ...
private
  { Private declarations }
  // Il faut déclarer manuellement le composant WinForm
  NotifyIcon1: NotifyIcon;
  // Ainsi que les événement associés
  procedure NotifyIcon1_DoubleClick(sender: System.Object; e: System.EventArgs);
  procedure NotifyIcon1_MouseDown(sender: System.Object; e: System.Windows.Forms.MouseEventHandler);
public
  { Public declarations }
end;
```

La création du composant sera effectuée dans l'événement OnCreate de la fiche. Pour créer un composant Winform il faut procéder comme les composants VCL, en appelant son constructeur.

```
// Redéclaration de la fonction ExtractIco, celle de Windows.pas n'étant pas correcte.
[DllImport('Shell32.dll', CharSet = CharSet.Auto, SetLastError = True, EntryPoint = 'ExtractIconEx')]
function ExtractIco(lpzFile: string; nIconIndex: Integer;
  out phiconLarge: IntPtr; out phiconSmall: IntPtr; nIcons: UInt): UInt; external;

procedure TForm1.FormCreate(Sender: TObject);
begin
  // L'instance de la classe NotifyIcon doit être créée dynamiquement
  // car les composants WinForm ne peuvent être déposés sur des fiches
  // VCLForm.
  NotifyIcon1 := System.Windows.Forms.NotifyIcon.Create;
```

```

NotifyIcon1.Text:='Nono40 - Source 79';

// Si l'icone est celle de l'exécutable
ExtractIco(Forms.Application.ExeName,0,Large,Small,1);
NotifyIcon1.Icon:=System.Drawing.Icon.FromHandle(Large);

// Association des méthodes aux événements correspondants. Notez ici
// la différence importante avec Delphi 7. Les événements sont ajoutés
// dans la liste des événements. Ce n'est plus une simple affectation
// d'adresse de méthode.
Include(NotifyIcon1.DoubleClick,NotifyIcon1_DoubleClick);
Include(NotifyIcon1.MouseDown ,NotifyIcon1_MouseDown);
// L'icône est affichée dans la barre de tâches
NotifyIcon1.Visible:=True;
end;

```

Notez dans le code ci-dessus la différence importante pour la création du lien entre les méthodes de la fiche et les événements du composant. Il ne faut plus mettre à jour l'adresse de l'événement en associant la méthode à l'événement mais il faut utiliser la fonction **Include** qui ajoute un gestionnaire d'événement dans la liste des gestionnaires.

Notez aussi que nous n'utiliserons pas `NotifyIcon.ContextMenu`, car il faudrait aussi créer un menu contextuel WinForm, alors qu'il est plus simple ici d'utiliser un menu contextuel VCL. La seule chose qu'il faudra effectuer en plus est l'affichage lors du click droit.

L'icône affichée ici est celle du fichier exécutable récupérée par la procédure `ExtractIco`. Il est possible aussi d'utiliser un fichier séparé ou un `TImage` pour stocker l'icône. Dans ces deux cas le code de création de `NotifyIcon1.Icon` est le suivant :

```

// Si l'icône est lue dans un fichier placé à coté de l'exécutable
NotifyIcon1.Icon:=System.Drawing.Icon.Create('Nono.ico');

// Si l'icone est stockée dans un TImage sur la fiche
Ico:=IntPtr.Create(Image1.Picture.Icon.Handle);
NotifyIcon1.Icon:=System.Drawing.Icon.FromHandle(Ico);

```

Ne pas oublier de détruire l'instance à la destruction de la fiche :

```

procedure TForm1.FormDestroy(Sender: TObject);
begin
// L'instance est créée sans propriétaire, donc ne pas
// oublier de la détruire.
NotifyIcon1.Free;
end;

```

### III-B - Ajout des autres composants

Ajouter ensuite sur la fiche un `PopupMenu` et un `Timer`.

Sur le `PopupMenu` définir trois éléments : "Restaurer", "réduire" et "Fermer".

Sur le `Timer` fixer son `Interval` à 500 et mettre `Enabled` à `False`.

### III-C - Ajout du code en réponse aux événements

En premier lieu renseigner les événements du menu contextuel :

```
procedure TForm1.MenuReduireClick(Sender: TObject);
begin
  // L'option réduire permet de ... réduire la fenêtre !
  WindowState:=wsMinimized;
end;

procedure TForm1.MenuRestaurerClick(Sender: TObject);
begin
  // L'option restaurer permet de ... restaurer la fenêtre !
  WindowState:=wsNormal;
end;

procedure TForm1.MenuFermerClick(Sender: TObject);
begin
  // Fermer la fenêtre
  Close
end;
```

Renseigner ensuite les gestionnaires d'événements liés au NotifyIcon :

```
procedure TForm1.NotifyIcon1_MouseDown(sender: TObject;
e: System.Windows.Forms.MouseEventHandler);
begin
  // Dans le cas d'un click gauche, on lance le timer pour
  // différencier le click, d'un double-click.
  If e.Button=System.Windows.Forms.MouseButtons.Left Then
    Timer1.Enabled:=True;
  // Dans le cas d'un click droit on affiche le menu.
  If e.Button=System.Windows.Forms.MouseButtons.Right Then
    PopUpMenu1.Popup(Mouse.CursorPos.X,Mouse.CursorPos.Y);
end;

procedure TForm1.NotifyIcon1_DoubleClick(sender: TObject;
e: System.EventArgs);
begin
  // Dans le cas d'un double-click, on arrête le timer lancé par le premier
  // click et on effectue l'action par défaut.
  Timer1.Enabled:=False;
  MenuRestaurerClick(nil);
end;
```

Renseigner enfin l'événement Timer du Timer :

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  // Si le timer s'est écoulé, c'est qu'il n'y a pas eu de double-click.
  // Dans ce cas on affiche le menu là où est la souris.
  Timer1.Enabled:=False;
  PopUpMenu1.Popup(Mouse.CursorPos.X,Mouse.CursorPos.Y);
end;
```

### III-D - Exécution

L'application est maintenant prête à être essayée.

Le source complet de cet exemple en disponible ici : [Source0079](#)

## Conclusion

L'utilisation d'une icône dans la barre des tâches est donc devenue plus facile qu'en utilisant les fonctions API dans un application Win32.

Source de l'application WinForm : [Source0078](#)

Source de l'application VCL.NET : [Source0079](#)

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.