

Créez des applications Delphi compatibles Windows Vista

par [Nathanial Woolls](#) Traduction par [Olivier Lance](#)

Date de publication : 05/12/06

Dernière mise à jour : 05/12/06

Windows Vista sera bientôt disponible au grand public, et vous aimeriez fournir à vos clients des applications qui en respectent à la fois le nouveau design et les nouvelles fonctionnalités ? Lisez cette traduction de Creating Windows Vista Ready Applications afin de découvrir comment procéder dès maintenant, sans attendre mi-2007 et la future version de Delphi.

- I - Donc, vos applications "tournent"
- II - L'application exemple
- III - Oh non... Pas encore (ou, nouvelles polices)
- IV - Où est mon intuition ? (ou, Pourquoi la fenêtre secrète ?)
- V - Faites briller vos applications
- VI - Quand une invite n'en est pas une (ou, Dialogues de Tâche)
- VI - Tout est dans l'image
- VIII - Nouvelles boîtes de dialogues
- IX - Conclusion

I - Donc, vos applications "tournent"

Aujourd'hui il faut l'espérer vous avez, en développeur responsable, téléchargé [l'une des versions publiques](#) du prochain système d'exploitation de Microsoft, [Windows Vista](#), et vous êtes assuré que vos applications fonctionnent sous le nouvel OS.

Bien que 99% des applications Delphi correctement codée n'aient aucun problème sous Vista (la considération des droits d'administration, de l'élévation du rang de l'utilisateur et de l'UAC [contrôle des comptes utilisateur] sort du cadre de cet article), elles n'apporteront sûrement pas les nouvelles améliorations dans l'expérience de l'utilisateur que l'on peut trouver dans Windows Vista. En réalité plus vous approfondirez vos recherches au sujet des nouveautés concernant l'expérience utilisateur dans Vista, plus vous découvrirez que vos applications n'y répondent que très peu.

Cependant, pas de panique ! Il est complètement possible, et relativement aisé, de rectifier ces écarts et de rendre vos applications compatibles à 100% avec la nouvelle expérience fournie par Windows Vista. Examinons quelques imperfections trouvées dans les applications générées par Delphi, et voyons comment les corriger ainsi que quelques étapes supplémentaires pour intégrer encore mieux nos applications.

II - L'application exemple

L'application qui servira d'exemple tout au long de ce tutoriel est un simple programme similaire au bloc-notes, intégrant deux fiches et quelques actions. Elle permet de charger et sauver du texte, et d'afficher une boîte de dialogue "A propos". Elle offrira également la possibilité de sauvegarder vos modifications lors de l'ouverture d'un fichier ou de la sortie du programme, si modifications il y a eu.

Application finale

[Téléchargez l'application exemple - Etape 1 \(miroir\)](#) Maintenant, bien que notre application d'exemple fonctionne correctement sous Windows Vista, quelques problèmes apparaissent lorsque l'on s'intéresse à son intégration aux éléments clés l'expérience de l'utilisateur sous Windows Vista.

III - Oh non... Pas encore (ou, nouvelles polices)

Le premier problème, le plus facile à résoudre -et le plus long dans des projets de grande nature- est la police de caractères. Windows Vista est accompagné d'une nouvelle police par défaut pour l'interface utilisateur : [Segoe UI](#). Cela n'est que la moitié du problème. Le point important est que, non seulement une nouvelle police est utilisée, mais de plus sa taille par défaut a été augmentée. Ordinairement les applications Windows XP utilisent la police Tahoma, taille 8. Les applications (qui voudraient être) correctement intégrée à Windows Vista utilisent toutefois Segoe UI, taille 9. Cela signifie qu'il faut d'une part modifier nos polices (si l'application est lancée sous Windows Vista), et d'autre part vérifier chaque écran pour s'assurer que tous les textes sont toujours visibles, en ajustant çà et là le cas échéant.

Commençons par créer une nouvelle unité nommée `uVistaFuncs.pas`. Nous y ajouterons nos méthodes spécifiques à Vista. A cette unité, nous allons ajouter des fonctions pour vérifier que notre application est lancée ou non sous Windows Vista et pour appliquer la police et taille de caractères adéquates à une fiche donnée. Ainsi, dans le `FormCreate` de notre fiche principale et de la fenêtre "A propos", nous n'aurons qu'à appeler `SetVistaFonts(Self)`, en prenant garde que `uVistaFuncs` est présente dans la clause `uses`.

Vous remarquerez que le titre de la boîte de dialogue "A propos" utilise la police par défaut, Tahoma. Pour cause, nous avons changé la police en mode design et la propriété `ParentFont` est maintenant à `False`. Vous trouverez cela dans beaucoup d'applications. Le meilleur moyen de vérifier que ce détail a été géré est d'éditer `uVistaFuncs` et de changer "Segoe UI" en "Segoe Script". Cela rendra plus évident, dans l'application, les endroits où des modifications doivent être apportées. Dans notre cas, il nous suffit d'assigner `TitleLabel.Font.Name` après avoir appelé `SetVistaFonts`.

Nouvelle police

[Téléchargez l'application exemple - Etape 2 \(miroir\)](#)

IV - Où est mon intuition ? (ou, Pourquoi la fenêtre secrète ?)

Windows Vista présente plusieurs éléments qui améliorent l'intuition dans l'expérience de l'utilisateur. Vous pouvez lire un article (bien qu'un peu dépassé) sur les interfaces utilisateur intuitives [ici](#). Basiquement, une interface intuitive assiste activement l'utilisateur dans la compréhension de ce qui est affiché à l'écran.

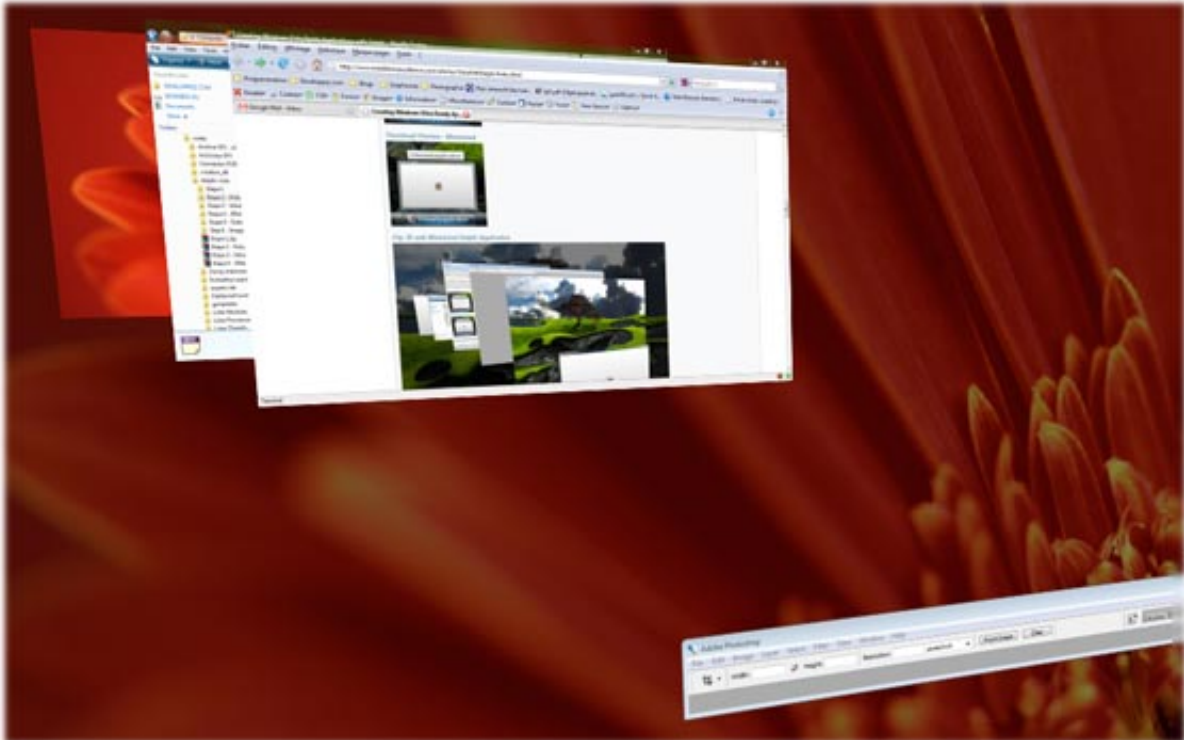
Quelques exemples d'une telle interface dans Windows Vista incluent les nouvelles animations de réduction et d'agrandissement d'une application. Comme le mentionne Ales Holecek dans son interview [ici](#), nombreux sont les utilisateurs qui ne comprennent pas le concept d'une fenêtre et de son bouton correspondant dans la barre des tâches. Ils ne saisissent pas où la fenêtre se trouve lorsqu'elle est minimisée et qu'elle est associée au bouton de la barre des tâches. Pour aider à remédier à ce problème (et pour exhiber l'interface 3D, [Aero](#)) dans Vista, les fenêtres sont doucement animées en 3D vers, et à partir de leur bouton de la barre des tâches lorsqu'elles sont réduites ou agrandies.

Allez-y, testez vous-même... pas avec une application Delphi ! Si vous réduisez ou agrandissez une application Delphi sous Vista, les animations sont celles de la fermeture ou de l'affichage d'une fenêtre : celle-ci s'efface petit à petit en se fermant ou apparaît en fondu lorsqu'elle est affichée. La raison de ce comportement est que toutes les applications Delphi ont une "fiche d'application", cachée et secrète. Cette fiche cachée est la propriétaire du bouton de la barre des tâches que vous pouvez voir dans vos applications Delphi (et c'est aussi pourquoi les menus systèmes de vos applications Delphi sont différents des autres applications). Lorsque vous réduisez ou restaurez vos fiches, la magie interne de Delphi outrepassé les messages et, à la place, affiche ou cache simplement les fiches actives. Vos fiches ne sont jamais réduites ou agrandies au sens propre du terme.

Autorisons-nous un moment de digression sur une autre nouveauté de Windows Vista. Nous reviendrons à notre fenêtre secrète bientôt. Cette nouvelle fonctionnalité dans Vista, qui prend également part à l'interface utilisateur intuitive, nous permet de voir un aperçu de nos fenêtre en passant simplement le curseur de la souris sur le bouton de notre application dans la barre des tâches. Cela facilite la recherche du bon bouton à cliquer dans la barre des tâches et est -chose discutable- plutôt sympathique. La même technique est également utilisée pour afficher le tout nouveau [Flip 3D](#), une représentation en trois dimensions du classique écran Alt+Tab. Si vous essayez cela avec une application Delphi normale et non réduite, tout fonctionne correctement. Cependant si vous essayez avec une application Delphi minimisée, vous ne verrez qu'une fenêtre vide avec l'icône de votre application plutôt que l'aperçu en temps réel.

Aperçu - fenêtre restaurée

Aperçu - fenêtre minimisée



Flip3D avec l'application minimisée

Ces impressions d'écran ne correspondent pas tout à fait à celles de l'article original. En effet, sur la version RC1 de Windows Vista que j'ai utilisée pour mes propres tests, la fenêtre une fois réduite n'apparaît pas du tout en aperçu ou dans Flip3D.

Pourquoi nos applications Delphi se comportent-elles différemment ? Serait-ce l'oeuvre d'une force diabolique ? Non ! C'est la fenêtre secrète ! Lorsque vous survolez le bouton de la barre de tâches, et qu'une fiche est agrandie, la fiche active de l'application est affichée dans l'aperçu. Toutefois, quand toutes les fiches sont réduites, la "fiche d'application" cachée est montrée dans l'aperçu.

Voyons maintenant comment régler le problème ! Nous y arriverons en utilisant les méthodes décrites par Peter Below [ici](#). La première chose que nous avons besoin de faire est de nous débarrasser du bouton dans la barre de tâches pour notre "fiche d'application" cachée. Dans le FormCreate de notre application, nous devons ajuster quelques indicateurs sur notre "fiche d'application" :

```
ShowWindow(Application.Handle, SW_HIDE);
SetWindowLong(Application.Handle, GWL_EXSTYLE,
  GetWindowLong(Application.Handle, GWL_EXSTYLE) and not WS_EX_APPWINDOW
  or WS_EX_TOOLWINDOW);
ShowWindow(Application.Handle, SW_SHOW);
```

Ensuite, pour chaque fiche pour laquelle nous voulons un bouton dans la barre de tâches (au moins la

fiche principale), nous devons surcharger CreateParams :

```
procedure TMainForm.CreateParams(var Params: TCreateParams);
begin
  inherited CreateParams(Params);
  Params.ExStyle := Params.ExStyle and not WS_EX_TOOLWINDOW or
    WS_EX_APPWINDOW;
end;
```

Enfin, pour chaque forme possédant son bouton et la possibilité d'être réduite, nous devons gérer le message Windows WM_SYSCOMMAND :

```
interface
...
protected
procedure WMSyscommand(var Message: TWmSysCommand); message WM_SYSCOMMAND;
...
implementation
...
procedure TMainForm.WMSyscommand(var Message: TWmSysCommand);
begin
  case (Message.CmdType and $FFF0) of
    SC_MINIMIZE:
      begin
        ShowWindow(Handle, SW_MINIMIZE);
        Message.Result := 0;
      end;
    SC_RESTORE:
      begin
        ShowWindow(Handle, SW_RESTORE);
        Message.Result := 0;
      end;
    else
      inherited;
  end;
end;
```

Ce code a pour effet d'empêcher aux processus internes de Delphi de surcharger la réduction et l'agrandissement de nos fiches, causant à la place un simple masquage ou affichage. Maintenant, nos fiches devraient se comporter correctement. Elles s'animent comme il se doit lorsqu'elles sont réduites et restaurées, et elles affichent correctement leur aperçu quand elles sont réduites dans la barre de tâches.

Cependant, il nous reste un problème majeur. Si nous lançons notre application exemple et sélectionnons "Aide" puis "A propos", et que finalement nous cliquons sur le bouton de la barre des tâches de la fenêtre, la fiche serait mise en avant-plan, par-dessus notre dialogue "A propos" ! Cela est dû au fait que, par défaut, Delphi définit la "fiche d'application" cachée comme parente des fiches modales. Pour rectifier cela, vous devez changer Form.PopupParent avant chaque appel à Form.ShowModal :

```
procedure TMainForm.AboutActionExecute(Sender: TObject);
begin
  AboutForm := TAboutForm.Create(Self);
  try
    AboutForm.PopupParent := Self;
  end;
```

```
AboutForm.ShowModal;  
finally  
  FreeAndNil(AboutForm);  
end;  
end;
```

[Téléchargez l'application exemple - Etape 3 \(miroir\)](#)

V - Faites briller vos applications

A moins que vous n'ayez vécu dans une grotte, vous savez que les applications tournant sous Windows Vista (sur une machine avec une carte accélératrice 3D) sont dessinées avec une zone non-cliente et une bordure translucides, à la manière de verre. Cet effet est appelé, en toute logique, *Glass* ndt: j'ai préféré ne pas traduire ce terme, qui garde tout son sens en Anglais.. Le Gestionnaire de Fenêtres (*DWM - Desktop Windows Manager*) fournit des API qui vous permettent d'étendre cette surface *Glass* sur la zone cliente de vos applications. Bien que ce ne soit pas nécessaire, cela peut aider votre application à s'intégrer en tant "qu'application Vista".

Nous allons commencer par ajouter quelques nouvelles fonctions à notre fichier *uVistaFuncs.pas* : *CompositingEnabled* et *ExtendGlass*. La source pour ces méthodes est basée sur le code trouvé [ici](#). *CompositingEnabled* indique si les fonctionnalités 3D sont disponibles ou non, et *ExtendGlass* permet d'étendre la surface *Glass* sur la zone cliente de nos fiches. Ajoutons maintenant un Panel au bas de notre fiche. La raison de sa présence est que les API du Gestionnaire de Fenêtres pour étendre la zone *Glass* requiert une surface noire sur laquelle effectuer son rendu. Enfin, nous ajoutons une nouvelle méthode à notre fiche principale qui vérifie si les fonctionnalités d'Aero sont activées et, si c'est le cas, qui initialise le Panel noir et appelle *ExtendGlass* pour dessiner l'effet *Glass* sur notre fiche principale :

[Téléchargez l'application exemple - Etape 4 \(miroir\)](#)

VI - Quand une invite n'en est pas une (ou, Dialogues de Tâche)

Windows Vista introduit un nouvel élément dans l'expérience de l'utilisateur, appelé le [Dialogue de Tâche](#). Il apporte de nombreuses invites sous l'égide d'une unique API afin de renforcer la consistance du système. Bien qu'il est possible de créer des invites extrêmement complexes avec l'API des Dialogues de Tâche, nous commencerons par un simple remplacement de MessageDlg. Notre méthode utilisera les Dialogues de Tâche de Vista s'ils sont disponibles, et retournera à des appels de MessageDlg si nécessaire. Il existe un parfait exemple de cette méthode [ici](#).

Une fois que nous avons ajouté la nouvelle méthode pour utiliser les Dialogues de Tâche à notre unité uVistaFuncs.pas, une simple modification de notre méthode SaveHandled permet à notre message à la fois de mieux s'intégrer dans l'expérience utilisateur de Vista, tout en fournissant un meilleur retour d'information à l'utilisateur final.

Dialogue classique

Nouveau Dialogue de Tâche de Windows Vista

[Téléchargez l'application exemple - Etape 5 \(miroir\)](#)

Le code fourni dans l'article original ne fonctionnait pas sur mon système, aucun dialogue ne s'affichait. Afin de le faire fonctionner correctement, j'ai du modifier certaines constantes dans uVistaFuncs.pas J'ai laissé en commentaire les valeurs initiales.

Les valeurs que j'utilise ont été trouvées [ici](#).

VI - Tout est dans l'image

Nous approchons de la fin. Ce changement aurait vraisemblablement dû être fait en portant notre application de Windows 2000 à Windows XP mais, dans cet exemple, notre application est bien loin derrière tout cela. Ce que nous devons faire maintenant, c'est remplacer ces images plates et pixellisées par de belles images, lisses, si possible ombrées, et transparentes. J'ai trouvé que la meilleure manière de faire cela était d'utiliser PngComponents et le composant TPngImageList, que l'on peut trouver [ici](#). Cette simple alternative aux TImageList nous permet de charger des images PNG transparentes dans une liste d'images et de les utiliser au sein de notre application.

[Téléchargez l'application exemple - Etape 5 \(miroir\)](#)

VIII - Nouvelles boîtes de dialogues

Windows Vista introduit de nouvelles boîtes de dialogue, dont des fenêtres d'ouverture et d'enregistrement d'un tout nouveau design. Ces nouvelles boîtes de dialogue incluent plusieurs fonctionnalités nouvelles à Windows Vista, comme la zone des Liens Favoris (ndt: cette traduction ne correspond pas nécessairement à la version française de Windows Vista, ne l'ayant pas à disposition pour vérifier)(personnalisable via le menu contextuel), de nouvelles vues et une fenêtre de sauvegarde repliable.

Dialogue de sauvegarde classique

Nouveau style de dialogue - replié

Nouveau style de dialogue - Déplié

Malheureusement, les applications écrites en Delphi n'utiliseront pas automatiquement ces nouveaux styles pour les dialogues d'ouverture et d'enregistrement de fichier. Cela est dû à des *flags* utilisés dans le code de l'unité Dialogs.pas, OGN_ENABLEHOOK et OFN_ENABLETEMPLATE. Pour plus d'information, lisez le document du Quality Central ici.

Pour régler ce problème :

- Premièrement, faites une copie de Dialogs.pas et placez cette copie soit dans le répertoire des sources de votre applications, ou dans un répertoire inclus dans vos chemins de bibliothèques.
- Deuxièmement, dans cette copie de Dialogs.pas, recherchez "OFN_ENABLEHOOK". Vous devriez voir le code "Flags := OFN_ENABLEHOOK;". Commentez cette ligne.
- Ajoutez la ligne "Flags := 0" après la ligne que vous venez de commenter
- Recherchez "OFN_ENABLETEMPLATE". Commentez le code en commençant deux lignes au dessus de la première occurrence de OFN_ENABLETEMPLATE, "if Template <> nil then", et en terminant à "hWndOwner := Application.Handle;". Cela doit représenter une vingtaine de lignes de code.
- Ajoutez la ligne "hWndOwner := ParentWnd;" après le code que vous venez de commenter
- Rendez-vous à la méthode "TCommonDialog.Execute". Commentez le "if", "begin", "else" et "ParentWnd := Application.Handle". Cela devrait vous laisser un code qui essaie toujours d'affecter à ParentWnd le Handle de la fenêtre active, retombant sur Application.Handle.
- Sauvegardez vos changements sur votre copie de Dialogs.pas

Notre application d'exemple devrait maintenant utiliser les nouveaux dialogues d'ouverture et de sauvegarde. La première étape ci-dessus permet d'utiliser une copie modifiée de la VCL sans affecter le fichier original. La seconde étape retire le premier *flag* incriminé, mais signifie aussi que nos dialogues ne seront pas centrés. La troisième étape est juste dans le prolongement de la seconde étape, initialisant le *flag* à zéro. La quatrième étape retire le second *flag* en cause de notre problème, ainsi que du code qui ne fonctionne plus pour centrer les dialogues. Les cinquième et sixième étapes corrigent le problème du centrage des dialogues, donnant en parent à nos dialogues la fiche active.

Etant donné que cette partie implique de modifier les sources de la VCL, je ne peux pas poster la source. Cependant, en suivant les instructions ci-dessus vous devriez pouvoir donner à toutes vos applications la possibilité d'utiliser le nouveau style de dialogues sans avoir à modifier vos sources. Si vous préférez ne pas modifier Dialogs.pas, lisez [cette page](#) pour des instructions utilisant les nouveaux dialogues d'ouverture et de sauvegarde avec des appels à l'API.

*Pour ma part, en plus des étapes ci-dessus, j'ai du renommer ma copie de Dialogs.pas en VistaDialogs.pas (ou tout autre nom différent de l'original) et ajouter VistaDialogs dans ma clause uses, **après** Dialogs.*

[Téléchargez l'EXE final \(miroir\)](#)

IX - Conclusion

Les changements de l'expérience de l'utilisateur dans Windows Vista, bien qu'importantes, n'apporte pas une rupture aussi capitale que celles apportées par Windows XP. Avec ce dernier, pour intégrer nos applications, il nous fallait soit utiliser un [gestionnaire de thème tiers](#), soit attendre Delphi 7. Avec Windows Vista, tant que nos applications étaient développées pour fonctionner avec Windows XP, elles cadreront automatiquement avec le nouveau style visuel de Windows Vista.

Cependant, depuis Windows Vista, Microsoft se concentre beaucoup plus sur tous les petits détails qui englobent l'expérience utilisateur fournie par un ordinateur personnel. Cela signifie que si nous, développeurs, voulons que nos applications s'intègrent dans cette expérience utilisateur consistante, nous devons également nous concentrer sur les détails de l'expérience de nos clients.

Bien que la prochaine version de Delphi abordera sans aucun doute plusieurs des sujets discutés ici, personnellement, je ne souhaite pas être obligé de mettre à jour mon environnement de développement uniquement pour que mes applications s'intègrent à un nouveau système d'exploitation. Les codes ci-dessus nous permettent d'offrir à nos utilisateurs une expérience riche, consistante, fonctionnant sous Windows Vista, et ce dès aujourd'hui.

Ci-dessous vous trouverez quelques articles et tutoriels au sujet de Windows Vista et des interfaces utilisateur :

- [What's New in Windows Vista](#)
- [Top Rules for the Windows Vista User Experience](#)
- [Top Guidelines Violations](#)
- [Top 10 Ways to Light Up Your Windows Vista Apps](#)
- [Using the new Windows Vista Task Dialog](#)
- [The new File Open / Save Dialogs in Windows Vista](#)
- [Taking the New Windows Vista Task Dialog One Step Further](#)