

**Utilisez la barre des tâches Windows 7 dans vos applications Delphi 2010**

Par Franck SORIANO

Date de publication : 22 Mai 2011

Dernière mise à jour :

|  |
| --- |
| Avec **Windows 7**, Microsoft a introduit une nouvelle barre des tâches. Cette dernière apporte un certain nombre d'innovations dont on peut profiter dans nos applications **Delphi 2010**.  Dans ce tutoriel, nous verrons comment mettre en oeuvre ces fonctionnalités en utilisant notamment la nouvelle interface **ITaskBarList3**. |

**I - Introduction** 3

**Télécharger les sources de l'article** 3

**II - Programmer avec la barre des tâches** 3

**II-A - Initialisation préalable** 3

**II-B - Utilisation de la JumpList** 3

**II-B-1 - Présentation générale** 3

**II-B-2 - Epinglé, Fréquent, Récent** 3

**II-B-3 - Tâches personnalisées** 3

**II-B-3-a - Initialisation d'une nouvelle liste** 3

**II-B-3-b - Reprise des éléments récents et fréquents** 3

**II-B-3-c - Ajout des tâches applicatives** 3

**II-C - Barre de progression** 3

**II-D - Overlay Icons** 3

**II-E - Regrouper plusieurs applications dans la barre des tâches** 3

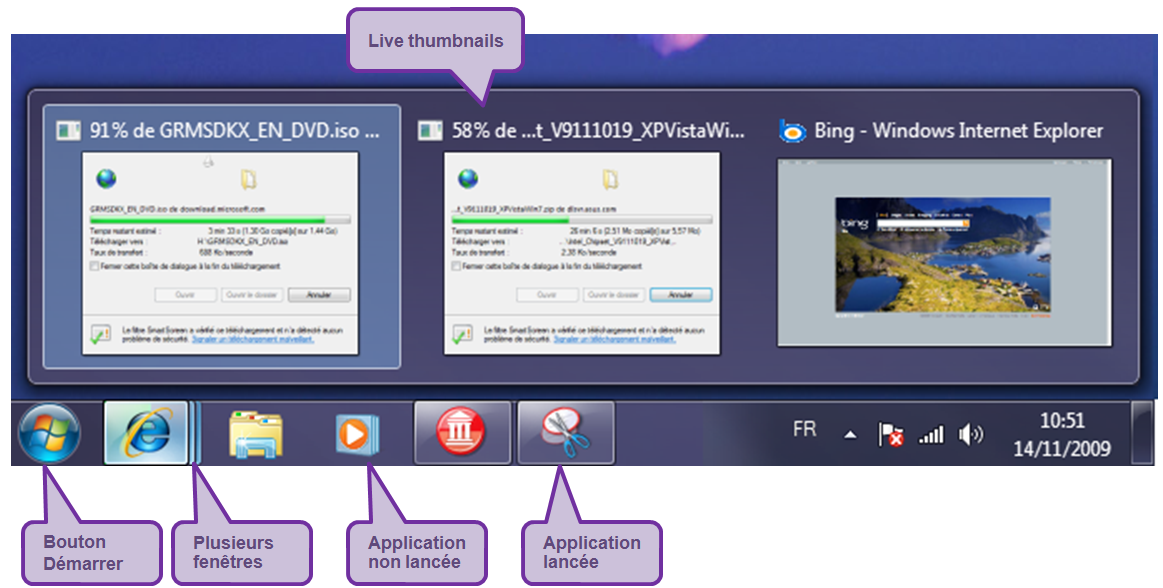
**III - Conclusion** 3

**IV - Références** 3

**V - Remerciements** 3

**I - Introduction**

Dans Windows 7 et Windows 2008, Microsoft a introduit une nouvelle barre des tâches :



Dans cette nouvelle barre, le nom des applications a disparu pour ne laisser la place qu'à leurs icônes. Windows utilise le fond de l'icône pour traduire un certain nombre d'informations : La couleur de fond et la bordure indiquent si l'application est lancée, si elle est active, si plusieurs programmes ont été regroupés derrière un seul bouton...

Le fond du bouton peut également se transformer en barre de progression : Le bouton se colore alors progressivement en vert en fonction de l'état d'avancement du traitement.

La barre de lancement rapide a également disparu. A présent, il est possible d'épingler une application en cours d'exécution pour qu'elle reste présente dans la barre une fois l'application terminée. Un simple clic sur son bouton suffira alors à la relancer.

*Windows Vista* avait déjà introduit le *Live thumbnails*, qui permetttait d'afficher un aperçu de l'écran principale d'une application avant de basculer dessus. Cependant ce dernier était limité à une seule application à la fois. A présent, le Live thumbnails est capable de travailler avec un groupe d'applications, permettant d'identifier ainsi rapidement celle sur laquelle on veut basculer.

Enfin, lorsqu'on fait un click droit sur une application, *Windows* ouvre un mini menu démarrer dédié à l'application concernée : La *Jump List*. Cette dernière permet alors de réouvrir facilement les derniers documents utilisés.

Cette barre des tâches apporte pas mal de nouvelles fonctionnalités intéressantes. Certaines sont automatiques et ne nécessitent pas d'action de la part de l'application (comme la gestion des documents récents), d'autres au contraire nécessitent un petit peu de programmation.

Dans ce tutoriel, nous allons voir quelques unes de ces nouveautés. Nous verrons comment utiliser les nouvelles API de Windows 7 prises en charge par Delphi 2010 à travers un petit programme de démonstration.

**Télécharger les sources de l'article**

Tout au long de cet article, nous allons développer un programme de démo mettant en oeuvre quelques fonctionnalités de la barre des tâches.

|  |  |
| --- | --- |
| Description | Emplacement |
| Demo\_barre\_taches\_win7 | [**taskbar.zip**](ftp://ftp-developpez.com/fsoriano/archives/windows7/delphi/utilisez-barre-taches-windows-7-dans-vos-applications-delphi-2010/fichiers/taskbar.zip) |
| L'article au format docx | [**barre\_taches\_w7**](ftp://ftp-developpez.com/fsoriano/archives/windows7/delphi/utilisez-barre-taches-windows-7-dans-vos-applications-delphi-2010/fichiers/barre_taches_w7.docx) |

Le projet de démo nécessite **Delphi 2010** pour compiler.

**II - Programmer avec la barre des tâches**

**II-A - Initialisation préalable**

La barre des tâches de Windows 7 ne peut pas être manipulée tant que Windows n'a pas créé le bouton correspondant à notre application.

Aussi, avant d'utiliser ses fonctionnalités, on doit attendre que le bouton ait été créé. Pour cela, on doit écouter le message utilisateur **TaskbarButtonCreated**. Comme il s'agit d'un message utilisateur et pas d'un message windows standard, on ne connait pas son identifiant à priori. On doit le demander à Windows avec la fonction **RegisterWindowMessage**.

Le plus simple, c'est de faire cette opération sur la fiche principale de l'application, dans son **FormCreate** :

procedure TfrmMain.FormCreate(Sender: TObject);

begin

// On commence par demander à Windows quel est l'identifiant du message

// TaskbarButtonCreated. Ce dernier sera envoyé par Windows à l'application

// lorsque son bouton aura été créé dans la barre des tâches.

msgTaskbarButtonCreated := RegisterWindowMessage('TaskbarButtonCreated');

Nous devons ensuite surcharger la méthode **WndProc** de la fiche afin de guetter la réception du message :

protected

procedure WndProc(var Message: TMessage); override;

Et nous l'implémentons de la façon suivante :

procedure TfrmMain.WndProc(var Message: TMessage);

begin

// On regarde si le message reçu correspond au message TaskbarButtonCreated

if Message.Msg = msgTaskbarButtonCreated

then begin

// On a reçu le message qu'on attendait, à présent on peut demander l'interface

// ITaskBarList3

TaskBar := CreateComObject(CLSID\_TaskbarList) as ITaskBarList3;

end

else inherited WndProc(Message);

end;

|  |  |
| --- | --- |
| kitwarning.gif | **WndProc** est la procédure de message de la fiche. Tous les messages windows qui sont envoyés à la fiche sont traités par cette méthode. Elle est fréquemment appelée, aussi lorsqu'on la surcharge, il faut faire très attention à bien appeler la méthode héritée et à ne pas effectuer de traitements trop long qui risqueraient de pénaliser les performances. |

L'implémentation est très simple : Une fois le message **TaskbarButtonCreated** reçu, on peut commencer à manipuler la barre des tâches. Comme on accède à la plupart de ses fonctionnalités avec l'interface **ITaskBarList3**, on en profite pour initialiser cette dernière. Il s'agit en fait d'un objet COM qu'on instancie à partir de son CLSID **CLSID\_TaskbarList**.

**TaskBar** et **msgTaskbarButtonCreated** sont deux attributs de la fiche déclarés de la façon suivante :

private

TaskBar : ITaskBarList3;

msgTaskbarButtonCreated : cardinal;

|  |  |
| --- | --- |
| kitinfo.gif | L'interface **ITaskBarList3** est définie dans l'unité **ShlObj**. |

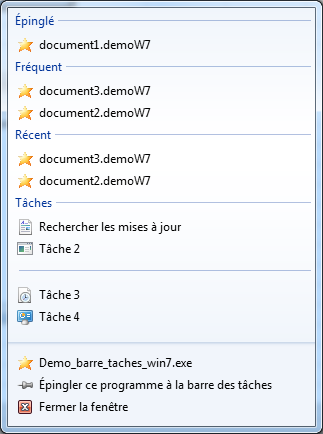
A présent, nous somme prêt à utiliser les nouvelles fonctionnalités de la barre des tâches.

**II-B - Utilisation de la JumpList**

**II-B-1 - Présentation générale**

La nouvelle barre des tâches offre un mini menu Démarrer propre à chaque application.

Celui-ci est accessible en faisant un clic droit sur le bouton de l'application :



* **Epinglé :** La JumpList affiche ici les documents récemment ouverts dans l'application et qui ont été volontairement épinglés par l'utilisateur. Il lui suffira ensuite de cliquer sur le document, et l'application recevra l'ordre d'ouvrir le document en question (sous la forme d'un **ShellExecute** avec le verbe *OPEN*).
* **Fréquent :** Cette liste affiche les documents fréquemment ouverts par l'utilisateur. Comme pour *Epinglé*, il s'agit d'une liste de raccourcis sur des documents.
* **Récent :** Cette liste affiche les derniers documents ouverts par l'application. Elle est semblable aux documents fréquents.
* **Tâches :** Cette liste est destinée à contenir une liste d'opérations courantes dans l'application. Le principe est de donner à l'utilisateur un moyen simple et rapide pour les lancer. Par exemple, Internet Explorer propose une tâche dans sa JumpList pour ouvrir un nouvel onglet à l'intérieur du navigateur. Il peut s'agir de commandes propres à l'application, tout comme des appels à d'autres exécutables (un progiciel pourrait ainsi prévoir une tâche pour accéder au site Web de l'éditeur). En fait, dans la pratique, la liste des tâches n'est rien d'autre qu'une liste de raccourcis, tels qu'on aurait pu les mettre dans le menu démarrer.

Conceptuellement parlant, la JumpList est une liste de raccourcis (au sens Windows du terme), organisés en différentes catégories qu'il est possible de personnaliser.

C'est ce que nous allons voir immédiatement.

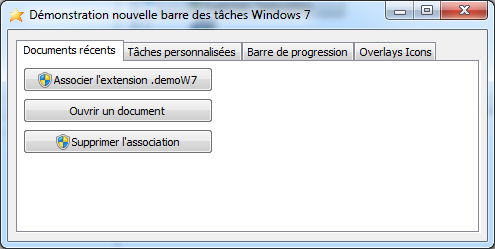
**II-B-2 - Epinglé, Fréquent, Récent**

Commençons par le plus simple : Les éléments Epinglés, Fréquents et Récents. C'est très simple pour une bonne raison : Ils sont gérés automatiquement par Windows.

Ainsi, il n'est pas nécessaire de faire quoi que ce soit pour pouvoir en bénéficier. La seule "contrainte" est qu'il faut avoir défini quels sont les types de fichier utilisés par l'application. Cette opération peut s'effectuer soit manuellement (l'utilisateur lance une commande "ouvrir avec..." depuis l'explorateur Windows) soit par programme en configurant la base de registre.

Ensuite, lorsqu'un document est ouvert dans l'application, il doit être mémorisé dans la liste des documents récents avec la fonction du Shell **SHAddToRecentDocs**. Cependant, on n'a pas besoin de se préoccuper de ce dernier aspect puisque Delphi mémorise automatiquement les documents récents si on passe par les composants standards pour les boîtes de dialogues : **TOpenDialog**, **TSaveDialog**.

Pour s'en convaincre, on va réaliser un petit programme de demo :



Le bouton *"Associer l'extension .demoW7"* va enregistrer l'application avec l'extension de fichier ".demoW7".

L'association d'une application à un type fichier s'effectue en écrivant dans la base de registre. Depuis *Windows Vista*, il est possible d'associer plusieurs applications pour traiter un même type de fichier. Aussi l'enregistrement de cette association est un petit peu plus complexe qu'avant et sort du cadre de ce tutoriel.

Je vais faire appel à la fonction **RegisterFileExtensions** définie dans la classe **TFileRegistration**. Les sources de cette classe se trouve dans l'unité **uFileRegistration.pas** que vous trouverez avec les sources du projet. Cependant je ne détaillerai pas son fonctionnement.

Comme je l'ai dit, pour définir cette association, il faut modifier la base de registres Windows. Or c'est une opération qui va être interdite par l'UAC.

Pour la réaliser, il faut passer par une élévation de privilèges. C'est à dire, qu'il faut exécuter un autre processus Windows, qui sera lancé avec les droits administrateurs.

Par souci de simplicité, je me suis servi de la même application (le même exe) :

Lorsqu'on clique sur le bouton "*Associer l'extension .demoW7*", on exécute le code suivant :

procedure TfrmMain.btAssocierFichierClick(Sender: TObject);

begin

RunAsAdmin(0, Application.ExeName, '/regfiles');

end;

C'est-à-dire on lance une nouvelle instance de l'application en tant qu'administrateur, en lui passant le paramètre "/regfiles" en ligne de commande.

Au démarrage de l'application, on teste les paramètres de la ligne de commande. Si on détecte l'ordre d'enregistrement, on effectue l'association et on sort immédiatement :

begin

// Si l'application est démarrée en ligne de commande avec le paramètre /regfiles

// on se contente de définir l'association avec les types de fichier.

if (ParamCount = 1) and ((ParamStr(1) = '/regfiles') or (ParamStr(1) = '/unregfiles'))

then begin

if ParamStr(1) = '/regfiles'

then TFileRegistration.RegisterFileExtensions('Demo\_barre\_W7', 'Demo barre des tâches Windows 7', ['.demoW7'])

else TFileRegistration.UnregisterFileExtensions('Demo\_barre\_W7', ['.demoW7']);

end

else begin

Application.Initialize;

Application.MainFormOnTaskbar := True;

Application.CreateForm(TfrmMain, frmMain);

Application.Run;

end;

end.

Vous remarquerez, qu'on procède de la même façon pour supprimer l'association avec le paramètre "/unregfiles".

La procédure **RunAsAdmin** est définie de la façon suivante :

procedure RunAsAdmin(hWnd : HWND; app, params : string);

var

sei : TShellExecuteInfo;

begin

Fillchar(sei,SizeOf(sei),0);

sei.cbSize := SizeOf(sei);

sei.Wnd := hWnd;

sei.fMask := SEE\_MASK\_FLAG\_DDEWAIT or SEE\_MASK\_FLAG\_NO\_UI;

sei.lpfile := PChar(app);

sei.lpVerb := 'runas';

sei.lpParameters := PChar(params);

sei.nShow := SW\_SHOWNORMAL;

// Attention ! si l'utilisateur n'accepte pas l'élévation de privillèges,

// on déclenche une exception !

if not ShellExecuteEx(@sei)

then RaiseLastOSError;

end;

Je vous invite à consulter le tutoriel de [**Pascal Fonteneau**](http://www.developpez.net/forums/u142033/pascal-fonteneau/) sur [**l'UAC**](http://pascalfonteneau.developpez.com/articles/delphi/vista/uac/) pour de plus amples explications sur la prise en compte de l'UAC.

|  |  |
| --- | --- |
| kitinfo.gif | Dans Delphi 2010, la VCL a été enrichie au sujet de l'UAC. A présent, pour afficher l'icône du bouclier sur les boutons lançant des opérations qui nécessitent une élévation de privilèges, il suffit de définir la propriété **ElevationRequired** du **TButton** à **true**. |

A présent, on peut utiliser le bouton *"Ouvrir un document"* pour appeler le composant **TOpenDialg** afin de sélectionner un fichier .demoW7 (on peut pour cela créer un simple fichier texte portant l'extension .demoW7, peu importe le fichier, on ne fait rien avec...) :

procedure TfrmMain.btOpenDialogClick(Sender: TObject);

begin

// La liste des documents récents affiche tous les documenhts enregistrés avec

// SHAddToRecentDocs.

// Lorsqu'on utilise une boîte de dialogue standard pour ouvrir un fichier,

// SHAddToRecentDocs est appelée automatiquement pour nous.

opDlg.Execute(Handle);

end;

On remarquera aussitôt que les derniers fichiers .demoW7 sélectionnés apparaissent dans la liste des documents récents tout seul. C'est ensuite à l'utilisateur d'épingler les fichiers qu'il veut conserver de façon permanente dans la liste *Epinglé*.

Si on clique sur un document récent dans la JumpList, on remarquera qu'une nouvelle instance de l'application est lancée. En effet, Windows exécute alors le verbe OPEN qui a été mémorisé pour le fichier, ce qui se traduit par le lancement d'une nouvelle instance de l'application.

**II-B-3 - Tâches personnalisées**

**II-B-3-a - Initialisation d'une nouvelle liste**

Voyons à présent comment définir une tâche dans la JumpList.

La JumpList se manipule grâce à l'interface **ICustomDestinationList**, définie dans l'unité **ShlObj**. Pour l'obtenir une référence sur cette interface, on doit instancier un objet COM fourni par Windows : **CLSID\_DestinationList**.

procedure TfrmMain.btConfigureTasksClick(Sender: TObject);

var

JumpList : ICustomDestinationList;

List : IObjectCollection;

pcMaxSlots : cardinal;

Link1 : IShellLink;

begin

// Pour définir des tâches, on doit remplacer la JumpList par une liste

// personnalisée. Pour celà, on passe par un objet COM CLSID\_DestinationList

// et son interface ICustomDestinationList.

JumpList := CreateComObject(CLSID\_DestinationList) as ICustomDestinationList;

On va remplacer la JumpList par défaut par une liste personnalisée. On débute la définition d'une nouvelle liste en appelant la méthode **BeginList**.

// On va définir une nouvelle liste. On demarre la définition de la liste par

// un appel à BeginList.

OleCheck(JumpList.BeginList(pcMaxSlots, IID\_IObjectArray, RemovedDestination));

**BeginList** est une méthode COM qui attend trois paramètres :

* **pcMaxSlots :** En sortie, ce paramètre indiquera le nombre maximum d'éléments qui peuvent être affichés dans la JumpList. Il s'agit du nombre d'éléments qui a été paramétrés dans les propriétés de la barre des tâches. La valeur par défaut est de 10. On peut définir davantage d'éléments dans la liste, mais Windows n'affichera que **pcMaxSlots** éléments, toute catégorie confondue.
* **IID\_IObjectArray :** Le deuxième paramètre permet d'indiquer le type d'interface désirée pour le troisième paramètre. En principe, on indique la valeur **IID\_IObjectArray** pour obtenir une interface **IObjectArray**.
* **RemovedDestination :** En sorti, le troisième paramètre contient une interface décrivant la liste des destinations (les éléments) qui ont été supprimés manuellement par l'utilisateur dans la JumpList. Le principe étant qu'on doit respecter les choix de l'utilisateur. Si ce dernier décide de supprimer un élément de la liste, on ne doit pas le ré-insérer.

**BeginList** est une méthode COM déclarée dans l'interface avec la convention d'appel STDCALL. Aussi, elle retourne un HRESULT en guise de code de retour qu'il faut tester pour savoir si l'appel s'est bien déroulé. On utilise pour cela la procédure **OleCheck** de l'unité **ComObj** qui déclenchera une exception en cas d'erreur.

**II-B-3-b - Reprise des éléments récents et fréquents**

On est en train de définir une nouvelle JumpList qui remplacera complètement la liste existante. Par défaut, les documents récents et fréquents seront supprimés de la liste.

Si on veut qu'ils continuent à apparaitre, il faut les ajouter explicitement dans la JumpList en appelant la méthode **AppendKnownCategory** :

// Ajout des documents fréquents

OleCheck(JumpList.AppendKnownCategory(KDC\_FREQUENT));

// Ajout des documents récents.

OleCheck(JumpList.AppendKnownCategory(KDC\_RECENT));

**KDC\_FREQUENT** et **KDC\_RECENT** sont de simples constantes également définit dans l'unité **ShlObj**.

**II-B-3-c - Ajout des tâches applicatives**

Pour ajouter nos tâches applicatives, il faut appeler la méthode **AddUserTasks** en lui donnant la liste des tâches à ajouter.

On doit donc commencer par créer une liste vide et la remplir avec les différentes tâches. Commençons par la liste vide :

// A présent, on prépare une nouvelle liste de tâches à ajouter dans la

// JumpList.

TaskList := CreateComObject(CLSID\_EnumerableObjectCollection) as IObjectCollection;

**AddUserTasks** attend qu'on lui fournisse une interface **IObjectArray** avec la liste des tâches. Cependant cette interface décrit un tableau statique et ne permet pas de lui ajouter de nouvel élément.

Pour construire notre liste de tâches, on utilise l'interface **IObjectCollection** qui hérite de **IObjectArray** en l'enrichissant pour qu'on puisse la manipuler comme une collection dynamique. La liste elle-même est un objet COM **CLSID\_EnumerableObjectCollection**.

|  |  |
| --- | --- |
| kitinfo.gif | **IObjectCollection** et **IObjectArray** sont définies dans l'unité **ObjectArray**. **CLSID\_EnumerableObjectCollection** est définie dans **ShlObj**. |

Maintenant voyons comment définir une tâche. En fait, il s'agit d'un banal raccourci tel qu'on pourrait le créer dans le menu démarrer. La seule différence, c'est qu'au lieu d'enregistrer le raccourci sur le disque, on va l'ajouter à la liste des tâches de la JumpList.

Nous devons donc créer un élément **IShellLink** (ou **IShellItem** d'après la MSDN) et veiller à renseigner :

* **Le chemin de l'application :** C'est la cible qui sera lancée à l'exécution de la tâche.
* **Les arguments de la ligne de commande :** Attention, c'est obligatoire. Il n'est pas possible de créer de tâche qui appelle une application sans argument.
* **L'emplacement de l'icône associée à la tâche :** La définition des tâches est persistante une fois configurée. Les tâches restent disponibles même lorsque l'application n'est pas lancée. Aussi, il faut bien indiquer l'emplacement vers une ressource de type ICONE. On ne peut pas se contenter de fournir une image quelconque.
* **Titre de la tâche :** C'est le nom de la tâche, tel qu'il devra apparaitre dans la liste. On peut définir soit directement un libellé, soit une référence vers une ressource de type string. Dans ce dernier cas, le libellé pourra être localisé automatiquement en fonction de la langue de l'utilisateur.

Pour nous simplifier la création des tâches, nous allons définir une fonction utilitaire qui retourne un élément **IShellLink** :

function CreateShellLink(Path, Args, Title, IconLocation : string; IconIndex : integer) : IShellLink;

var

PropertyStore : IPropertyStore;

TitleValue : PROPVARIANT;

begin

// Création d'une nouvelle instance IShellLink

result := CreateComObject(CLSID\_ShellLink) as IShellLink;

// On définit le chemin vers l'application à lancer.

OleCheck(result.SetPath(PWideChar(Path)));

// On définit les arguments d'appel.

OleCheck(result.SetArguments(PWideChar(Args)));

// On définit l'emplacement de l'icônes

OleCheck(result.SetIconLocation(PWideChar(IconLocation), IconIndex));

// Pour définir le titre du raccourci, il faut passer par l'interface IPropertyStore

// On définit la propriété PKEY\_Title

PropertyStore := result as IPropertyStore;

TitleValue.vt := VT\_LPWSTR;

TitleValue.pwszVal := PWideChar(Title);

OleCheck(PropertyStore.SetValue(PKEY\_Title, TitleValue));

OleCHeck(PropertyStore.Commit); // On valide la définition des propriétés

end;

On peut également insérer des lignes de type séparateur dans la liste des tâches, comme pour un menu. Pour cela, on définit un raccourci vide et on initialise sa propriété **PKEY\_AppUserModel\_IsDestListSeparator** à *true*.

function CreateSeparator : IShellLink;

var

PropertyStore : IPropertyStore;

TitleValue : PROPVARIANT;

begin

// On crée un nouveau raccourci vide.

result := CreateComObject(CLSID\_ShellLink) as IShellLink;

// Au lieu de définir son titre, on définit simplement la propriété

// PKEY\_AppUserModel\_IsDestListSeparator à true.

PropertyStore := result as IPropertyStore;

TitleValue.vt := VT\_BOOL;

TitleValue.boolVal := true;

OleCheck(PropertyStore.SetValue(PKEY\_AppUserModel\_IsDestListSeparator, TitleValue));

OleCHeck(PropertyStore.Commit);

end;

**PKEY\_Title** et **PKEY\_AppUserModel\_IsDestListSeparator** sont deux constantes définies de la façon suivante :

const

PKEY\_Title : TPropertyKey = (fmtid: '{F29F85E0-4FF9-1068-AB91-08002B27B3D9}'; pid: 2);

PKEY\_AppUserModel\_IsDestListSeparator : TPropertyKey = (fmtid: '{9F4C2855-9F79-4B39-A8D0-E1D42DE1D5F3}'; pid: 6);

Finalement, il ne nous reste plus qu'à définir une liste de tâches. Le code complet pour la définition des tâches est le suivant :

procedure TfrmMain.btConfigureTasksClick(Sender: TObject);

var

JumpList : ICustomDestinationList;

RemovedDestination : IObjectArray;

TaskList : IObjectCollection;

pcMaxSlots : cardinal;

Link1 : IShellLink;

begin

// Pour définir des tâches, on doit remplacer la JumpList par une liste

// personnalisée. Pour celà, on passe par un objet COM CLSID\_DestinationList

// et son interface ICustomDestinationList.

JumpList := CreateComObject(CLSID\_DestinationList) as ICustomDestinationList;

// On va définir une nouvelle liste. On demarre la définition de la liste par

// un appel à BeginList.

OleCheck(JumpList.BeginList(pcMaxSlots, IID\_IObjectArray, RemovedDestination));

try

// La liste va complètement remplacer la liste existante. Elle va également

// supprimer l'affichage des documents récents et des documents fréquents.

// Si on veut qu'ils continuent à s'afficher, on doit les faire apparaitres

// dans la liste en utilisant AppendKnownCategory :

// Ajout des documents fréquents

OleCheck(JumpList.AppendKnownCategory(KDC\_FREQUENT));

// Ajout des documents récents.

OleCheck(JumpList.AppendKnownCategory(KDC\_RECENT));

// A présent, on prépare une nouvelle liste de tâches à ajouter dans la

// JumpList.

TaskList := CreateComObject(CLSID\_EnumerableObjectCollection) as IObjectCollection;

// Création d'une première tâche "Rechercher les mises à jour"

Link1 := CreateShellLink(Application.ExeName, 'Task1', 'Rechercher les mises à jour', 'Shell32.dll', 1);

OleCheck(TaskList.AddObject(Link1));

// On ajoute un séparateur

Link1 := CreateSeparator;

OleCheck(TaskList.AddObject(Link1));

// Création d'une deuxième tâche.

Link1 := CreateShellLink(Application.ExeName, 'Task2', 'Tâche 2', 'Shell32.dll', 2);

OleCheck(TaskList.AddObject(Link1));

// Enfin, on définit la liste des tâches en appelant AddUserTasks.

OleCheck(JumpList.AddUserTasks(TaskList));

except

// En cas d'erreur, il faut annuler la liste en cours de définition

JumpList.AbortList;

raise; // Puis on redéclenche l'exception.

end;

OleCheck(JumpList.CommitList);

end;

Les tâches ici créées vont chercher leurs icônes dans **Shell32.dll**.

Les tâches sont ajoutées à la JumpList par un appel à **AddUserTasks**.

Lorsqu'on a fini de modifier la JumpList, il faut valider les changements en appelant **CommitList**. Inversement, en cas d'erreur, si on veut abandonner la définition de la liste, il faut appeler **AbortList**.

|  |  |
| --- | --- |
| kitinfo.gif | La définition des tâches (en fait la personnalisation de la JumpList) est persistante entre deux exécutions de l'application. Ainsi, il n'est pas nécessaire de les redéfinir à chaque lancement de l'application.  Elle persiste même lorsque l'application est épinglée dans la barre des tâches alors qu'elle n'est pas lancée. |

Pour effacer la liste des tâches, il suffit de redéfinir une JumpList vide :

procedure TfrmMain.btRemoveTasksClick(Sender: TObject);

var

JumpList : ICustomDestinationList;

RemovedDestination : IObjectArray;

pcMaxSlots : cardinal;

begin

JumpList := CreateComObject(CLSID\_DestinationList) as ICustomDestinationList;

OleCheck(JumpList.BeginList(pcMaxSlots, IID\_IObjectArray, RemovedDestination));

// Ajout des documents fréquents

OleCheck(JumpList.AppendKnownCategory(KDC\_FREQUENT));

// Ajout des documents récents.

OleCheck(JumpList.AppendKnownCategory(KDC\_RECENT));

OleCheck(JumpList.CommitList);

end;

**II-C - Barre de progression**

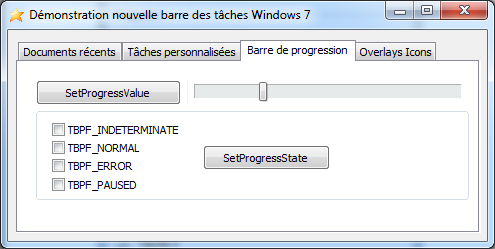
Une autre nouveauté de la barre des tâches consiste en la possibilité d'utiliser l'icône de l'application comme une barre de progression !

progressbar.png

Par exemple, lorsqu'on lance un gros téléchargement depuis IE, ou lorsqu'on fait une copie de fichier dans l'explorateur Windows, l'icône de l'application dans la barre des tâches se transforme en barre de progression indiquant en permanence la progression du traitement.

De cette façon, l'utilisateur peut basculer sur une autre application tout en suivant la progression de son traitement dans un coin de l'écran.

Voilà une fonctionnalité sympa très facile à mettre en oeuvre : Nous allons réaliser une petite demo illustrant le fonctionnement de cette barre de progression :



On définit une TrackBar qui reflètera la barre de progression dans la barre des tâches. Lorsque l'utilisateur déplacera la tirette, la barre de progression sera mise à jour en même temps.

En fait, c'est vraiment très simple à mettre en oeuvre puisqu'on a une seule méthode à appeler : **ITaskBarList3.SetProcessValue**

On a déjà vu comment obtenir et initialiser l'interface **ITaskBarList3**. Aussi le code permettant d'afficher la barre de progression se résume à :

procedure TfrmMain.trackBarChange(Sender: TObject);

begin

TaskBar.SetProgressValue(Handle, trackBar.Position, trackBar.Max);

end;

Rien d'autre !

La méthode **SetProgressValue** attend trois paramètres :

* **Hwnd :** Handle de la fenêtre Windows qui contient la barre de progression à afficher.
* **ullCompleted :** Il s'agit d'un *int64* indiquant la position courante de la barre de progression.
* **ullTotal :** Il s'agit également d'un *int64* indiquant la position maximale de la barre de progression.

Il suffit d'un seul appel à **SetProgressValue** pour faire apparaitre la barre de progression derrière l'icône de l'application.

Il est ensuite possible de personnaliser un peu l'apparence de la barre de progression en utilisant **SetProgressState :**

TaskBar.SetProgressState(Handle, TBF\_INDETERMINATE)

L'état courant de la barre de progression ne peut pas être déterminé (par exemple, on vient de lancer un téléchargement dont on ne connait pas la taille totale...). Au lieu d'afficher une barre de progression, Windows affiche une barre verticale qui se déplace de gauche à droite.

TaskBar.SetProgressState(Handle, TBF\_NORMAL)

La barre de progression s'affiche normalement : La progression s'affiche en vert.

SetProgressState(Handle, TBF\_ERROR)

Le traitement vient de rencontrer une erreur. En principe, l'application devrait stopper la barre de progression, et la progression courante s'affiche en rouge.

TaskBar.SetProgressState(Handle, TBF\_PAUSED)

Le traitement a été mis en pause. Comme précédemment, la progression devrait être arrêtée. La progression courante s'affiche en jaune.

Pour finir, on peut faire disparaitre la barre de progression par un appel à :

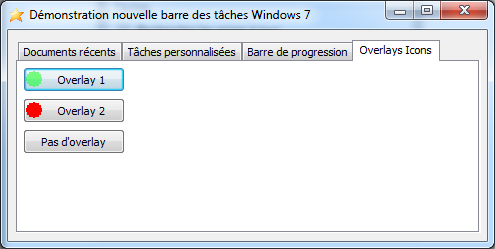
TaskBar.SetProgressState(Handle, 0)

**II-D - Overlay Icons**

Avec Windows 7, Microsoft déconseille totalement l'utilisation de la tray-icon pour afficher des icônes reflétant l'état de l'application. D'ailleurs cette dernière est considérablement réduite et la plupart des icones n'apparaissent même plus.

A la place, Windows 7 introduit une nouvelle manière d'indiquer l'état d'une appli : Les icônes en surimpression. Concrètement on affiche l'icône en surimpression par-dessus l'icône normale de l'application.

Là aussi, cette fonctionnalité est assez simple à mettre en oeuvre :



Ici, lorsqu'on clique sur un bouton, on va afficher l'icône correspondante en overlay dans la barre des tâches.

Une fois encore, cette fonctionnalité se résume en un appel à une seule méthode : **ITaskBarList3.SetOverlayIcon**.

La méthode attend trois paramètres :

* **hwnd :** Handle de la fenêtre de l'application.
* **hIcon :** Handle de l'icône à afficher.
* **pszDescription :** Une description textuelle de l'état où se trouve l'application.

Par commodité, j'ai placé chaque image dans une ImageList. Puis j'utilise cette dernière pour obtenir les icônes à afficher :

procedure TfrmMain.Button6Click(Sender: TObject);

var

Icon : HICON;

begin

Icon := ImageList\_GetIcon(imgList.Handle, TButton(Sender).ImageIndex, ILD\_TRANSPARENT);

try

OleCheck(TaskBar.SetOverlayIcon(Handle, icon, PWideChar(TButton(Sender).Hint)));

finally

DestroyIcon(Icon);

end;

end;

Le résultat est immédiat :

|  |  |
| --- | --- |
| overlay1.png | overlay2.png |

**II-E - Regrouper plusieurs applications dans la barre des tâches**

Lorsque nous lançons plusieurs instances de l'application, Windows regroupe automatiquement toutes les instances sous le même bouton dans la barre des tâches.

Lorsqu'il s'agit du même exe, celà semble évident. Si on regarde comment se comportent Internet Explorer, ou l'explorateur de fichier on peut remarquer que parfois, ce sont des fenêtre complètement différentes qui sont regroupées dans la barre des tâches. Par exemple, les fenêtres de téléchargements de fichiers ou les copies de fichiers sont bien regroupées et identifiées comme faisant partie de leur application respective. Pourtant, il s'agit bien de processus différents indépendants. Si on ferme Internet Explorer, les téléchargements en cours ne se ferment pas pour autant !

Alors comment peut-on faire la même chose dans nos applications ? Par exemple, si notre application se compose de plusieurs exe, comment faire en sorte qu'ils apparaissent ensemble dans la barre des tâches ?

En fait c'est très simple. Les applications ne sont pas regroupées ensemble parce qu'elles ont le même exe, mais parce qu'elles partagent le même **Application ID**.

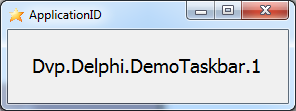
Un **application ID** n'est rien d'autre qu'une chaîne de caractères, d'une longueur maximale de 128 charactères. Par défaut, Windows définit un **Application ID** automatique en se basant sur le nom de l'executable.

Cependant, il est possible de définir explicitement l'application ID d'un processus grâce à la fonction **SetCurrentProcessExplicitAppUserModelID** du Shell. Cette dernière est définie dans l'unité **ShlObj**.

Par convention, l'application ID est une chaîne de caractères sous la forme :

Company.Product.SubProduct.Version

Pour illustrer le principe, nous allons créer une deuxième application composée d'une seule fiche avec un label.



Puis nous ajoutons l'appel à **SetCurrentProcessExplicitAppUserModelID**, à la fin du FormCreate de la fiche principale des deux applications :

procedure TForm1.FormCreate(Sender: TObject);

begin

OleCheck(SetCurrentProcessExplicitAppUserModelID('Dvp.Delphi.DemoTaskbar.1'));

end;

procedure TfrmMain.FormCreate(Sender: TObject);

begin

// On commence par demander à Windows quel est l'identifiant du message

// TaskbarButtonCreated. Ce dernier sera envoyé par Windows à l'application

// lorsque son bouton aura été créé dans la barre des tâches.

// Lorsqu'on aura reçu

msgTaskbarButtonCreated := RegisterWindowMessage('TaskbarButtonCreated');

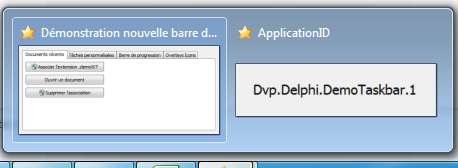
if ParamCount > 0

then Caption := ParamStr(1);

OleCheck(SetCurrentProcessExplicitAppUserModelID('Dvp.Delphi.DemoTaskbar.1'));

end;

Il ne reste plus qu'à lancer les deux applications et a tester :



**III - Conclusion**

Dans ce tutoriel, nous avons vu quelques unes des fonctionnalités de la nouvelle barre des tâches Windows 7 et comment les mettre en oeuvre sous Delphi 2010.

La liste n'est pas exhaustive. D'autres fonctionnalités telles que la Thumbnail toolbar n'ont pas été abordées.

Télécharger le projet complet : [**taskbar.zip**](./fichiers/taskbar.zip)

**IV - Références**

Windows 7 Training Kit for Developer :

kiten.jpg [**http://www.microsoft.com/downloads/en/details.aspx?FamilyID=1c333f06-fadb-4d93-9c80-402621c600e7**](http://www.microsoft.com/downloads/en/details.aspx?FamilyID=1c333f06-fadb-4d93-9c80-402621c600e7)

La documentation de l'interface ITaskbarList3 sur MSDN :

kiten.jpg [**http://msdn.microsoft.com/en-us/library/dd391692(v=vs.85).aspx**](http://msdn.microsoft.com/en-us/library/dd391692(v=vs.85).aspx)

**V - Remerciements**

Je remercie particulièrement [**nono40**](http://www.developpez.net/forums/u1804/nono40/), [**pedro**](http://www.developpez.net/forums/u25538/pedro/) et [**ero-sennin**](http://www.developpez.net/forums/u68681/ero-sennin/) pour leur relecture et conseils avisés !