

Réalisation d'un écran de veille avec Delphi

par [Nono40](#)

Date de publication : 23/09/2004

Dernière mise à jour :

Cet article présente la méthode pour la réalisation d'un écran de veille simple avec Delphi.

I - Principe

- I-A - Ecran de veille, simple exécutable
- I-B - Affichage normal de l'écran de veille
- I-C - Configuration de l'écran de veille
- I-D - Prévisualisation de l'écran de veille
- I-E - Modification du mot de passe
- I-F - Signaler au système qu'il est en veille
- I-G - Application mono-instance, quoi-que...

II - Réalisation

- II-A - Données globales
- II-B - Fenêtre de visualisation de l'écran de veille
- II-C - Modification du source du projet
- II-D - Ajout d'une fenêtre de configuration
- II-E - Gestion du mot de passe

III - Conclusion

I - Principe

I-A - Ecran de veille, simple exécutable

Un écran de veille pour Windows est en fait un simple exécutable dont l'extension .EXE est renommée en .SCR.

Bien sur afin que celui-ci fonctionne comme un écran de veille il doit répondre aux demandes de windows passées en paramètres lors de son exécution. Le détail de ces paramètres est donné dans les paragraphes suivants.

Windows passe en premier paramètre le type d'action que l'écran de veille doit réaliser, comme la visualisation, configuration, etc.

Le deuxième paramètre est optionnel et dépend du premier.

I-B - Affichage normal de l'écran de veille

Pour afficher l'écran de veille Windows lance l'application en lui passant **-S**, **/S** ou **S** en premier paramètre.

Le second paramètre est vide dans ce cas.

Sur la commande **S** il faut afficher l'écran de veille normalement, le mieux est de le faire dans une fenêtre sans bord (**bsNone**), en avant-plan (**fsStayOnTop**) et en plein écran (**wsMaximized**).

C'est à l'écran de veille de gérer la sortie quand on bouge la souris, on appuie sur une touche, etc... De même c'est à l'application de vérifier que l'écran de veille est éventuellement protégé par un mot de passe et de demander celui-ci pour se fermer.

Si l'écran de veille est protégé par mot de passe, la clef *HKEY_CURRENT_USER\Control Panel\Desktop* doit contenir une valeur *ScreenSaveUsePassword* non nulle.

I-C - Configuration de l'écran de veille

Pour afficher la fenêtre de configuration de l'écran de veille, Windows lance l'application avec le paramètre **-C**, **/C** ou **C**.

Le second paramètre est vide dans ce cas.

Sur la commande **C** il faut afficher la fenêtre de configuration de l'écran de veille. Les valeurs de paramétrage de l'écran de veille peuvent simplement être stockées dans un fichier INI ou dans la base de registre. Vous pouvez prendre la méthode que vous préférez. La fenêtre doit lire les infos de configuration, les afficher, et les sauvegarder si l'utilisateur valide les modifications.

Il n'y a pas de spécification particulière sur la fenêtre de configuration.

I-D - Prévisualisation de l'écran de veille

Pour prévisualiser l'écran de veille Windows lance l'application en lui passant **-P**, **/P** ou **P** en premier paramètre.

Le second paramètre est le Handle de la fenêtre dans laquelle doit s'afficher l'écran de veille.



*Attention, la prévisualisation est l'affichage dans le petit moniteur de la fenêtre des propriétés d'affichage du bureau. Le bouton "tester" de cette même fenêtre ne fait que lancer l'écran de veille normalement avec la commande **S**.*

Si la fenêtre utilisée par la commande **S** est définie comme ci-dessus, alors il suffit d'assigner le **ParentHandle** de cette fenêtre avec la valeur reçue dans le deuxième paramètre.

Il ne faut pas gérer le mot de passe dans ce cas, ni la sortie par les mouvements de souris ou les évènements clavier. Windows se chargera de fermer l'écran de veille en envoyant un message WM_CLOSE à l'application. Rien ne doit donc empêcher sa fermeture en mode prévisualisation.



Vous pouvez vous-aussi afficher dans une fenêtre Delphi n'importe quel écran de veille, voir [Prévisualisation d'un écran de veille](#).

I-E - Modification du mot de passe

Pour modifier le mot de passe de l'écran de veille Windows lance l'application en lui passant **-A**, **/A** ou **A** en premier paramètre.

Le second paramètre est le Handle de la fenêtre pouvant servir de parent à la boîte de dialogue de saisie du mot de passe.

Vous pouvez gérer vous-même les fenêtres de saisie de mot de passe, mais le mieux est d'utiliser celles de Windows.

Pour appeler la fenêtre de modification du mot de passe de l'écran de veille il faut utiliser la fonction **PwdChangePasswordA** de la librairie *MPR.DLL*.

Pour appeler la fenêtre de vérification du mot de passe (pour sortir de l'écran de veille) il faut utiliser la fonction **VerifyScreenSavePwd** de la librairie *PASSWORD.CPL*.

L'appel de ces deux fonctions sera détaillé un peu plus loin.

I-F - Signaler au système qu'il est en veille

L'écran de veille, quand il est en mode normal, (commande **S**) doit signaler au système qu'il est en veille. Pour cela il faut utiliser la fonction **SystemParametersInfo** avec le paramètre

SPI_SCREENSAVERRUNNING.

Au delà du fait que le système sait qu'il est en mode veille, ceci simplifie beaucoup la gestion des accès externes. En effet en mode veille Alt-TAB, Ctrl-ESC, etc sont désactivés. Il n'y donc plus aucune gestion à faire pour bloquer le basculement de tâche quand l'écran de veille est lancé.

I-G - Application mono-instance, quoi-que...

Une application SCR doit être une application mono-instance, il faut s'en assurer par l'utilisation d'un mutex, voir [Instancier une seule fois l'application](#).

Mais bien sur... Il y a une exception à la règle. Dans le cas de la modification du mot de passe, Windows ne ferme pas l'instance en cours de prévisualisation. Donc pour la modification du mot de passe il faut pouvoir lancer une deuxième instance de l'application. Ceci peut se faire facilement dans le source du projet comme nous allons le voir plus loin.

II - Réalisation

Pour présenter la réalisation des principes proposés précédemment nous allons réaliser un simple écran de veille ou une petite image rebondi sur les bords de l'écran. C'est bien sur simpliste, mais permet de ne garder dans le code que les principes de base d'un écran de veille.

Cet écran de veille aura un seul paramètre de configuration : la vitesse de déplacement de l'image.

II-A - Données globales

Ajouter une unité au projet afin d'y stocker les variables globales et les procédures de lecture/écriture des paramètres.

Le mode de stockage sera ici un fichier INI situé là où est l'exécutable. C'est la méthode la plus simple dans ce cas.

Le source de cette unité est le suivant :

```

unit Declare;

interface

Type TssMode = ( ssAffiche , ssConfig , ssMotDePasse , ssPrevisu );

Var
// Mode de fonctionnement de l'application
ssMode    : TssMode = ssAffiche;

// Valeur des paramètres passés au programme
Param1    : String;
Param2    : String;

// Valeur de déplacement de l'image
DecX      : Integer = 3;
DecY      : Integer = 3;

// Option contenant la vitesse en cours.
optVitesse : Integer = 100;

Procedure LitIni;
Procedure EcriIni;

implementation

Uses IniFiles, SysUtils;

Procedure LitIni;
Var Ini:TIniFile;
Begin
Ini:=TIniFile.Create(ExtractFilePath(ParamStr(0))+'.CONFIG.INI');
Try
optVitesse := Ini.ReadInteger('ECRAN','Vitesse',optVitesse);
Finally
Ini.Free;
End;
End;

```

```

Procedure EcritIni;
Var Ini:TIniFile;
Begin
Ini:=TIniFile.Create(ExtractFilePath(ParamStr(0))+'CONFIG.INI');
Try
Ini.WriteInteger('ECRAN','Vitesse',optVitesse);
Finally
Ini.Free;
End;
End;

end.

```

II-B - Fenêtre de visualisation de l'écran de veille

Ajouter une fenêtre à votre projet servant d'affichage de l'écran de veille. Sur celle-ci définissez les propriétés suivantes :

- **BorderStyle** : **bsNone** (obligatoire)
- **FormStyle** : **fsStayOnTop** (obligatoire)
- **WindowState** : **wsMaximized** (obligatoire)
- **Color** : **clBlack**

Dans cette fenêtre placez en haut à gauche un **TImage** dont la propriété **AutoSize** est à True. Chargez une image de petite taille (car elle doit rebondir aussi dans la petite fenêtre de prévisualisation).

Placer aussi un **TTimer** sur la fiche pour gérer le déplacement de l'image.

Dans le code de création de la fiche, il faut lire le fichier INI et utiliser les valeurs qu'il contient.

```

procedure TFenEcran.FormCreate(Sender: TObject);
begin
// Lecture des paramètres de configuration
LitIni;
Timer.Interval := optVitesse;
// Suppression du curseur
cursor:=crNone;
end;

```

A l'affichage de la fenêtre, on masque l'application dans la barre des tâches et on signale au système que l'écran de veille est actif.

La fonction **SystemParametersInfo** attend quatre paramètres :

- **uiAction** : Identificateur du paramètre à modifier, ici SPI_SCREENSAVERRUNNING
- **uiParam** : Valeur du paramètre : 1 si l'écran de veille est actif, 0 sinon
- **pvParam** : Pointeur vers un entier (inutilisé dans ce cas)
- **fWinIni** : Mettre 1 pour modifier le fichier WinIni, dans notre cas il faut mettre 0

```

procedure TFenEcran.FormShow(Sender: TObject);

```

```

Var i:Integer;
begin
  // On masque l'application dans la barre des tâches
  ShowWindow(Application.Handle,SW_HIDE);
  // On dit au système que l'on est en mode veille
  SystemParametersInfo(SPI_SCREENSAVERUNNING,1,@i,0);
end;

```

Le code du timer est le suivant (c'est simpliste, je sais...)

```

procedure TFenEcran.TimerTimer(Sender: TObject);
begin
  // Code de déplacement de l'image
  Image.Left := Image.Left + DecX;
  Image.Top := Image.Top + DecY;
  If (Image.Left + Image.Width)>=ClientWidth Then DecX := -3;
  If (Image.Top + Image.Height)>=ClientHeight Then DecY := -3;
  If Image.Left <= 0 Then DecX := 3;
  If Image.Top <= 0 Then DecY := 3;
end;

```

L'écran de veille doit gérer les causes de sa fermeture, ici nous allons simplement gérer le click de la souris. Mais d'autres évènements peuvent facilement être ajoutés.

```

procedure TFenEcran.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  // Sur un click on ferme l'écran de veille
  Close;
end;

```

Lors de la fermeture de la fiche il faut dire aussi au système que l'on sort de l'écran de veille :

```

procedure TFenEcran.FormClose(Sender: TObject; var Action: TCloseAction);
Var i:Integer;
begin
  // On dit au système qu'il n'est plus en mode veille
  SystemParametersInfo(SPI_SCREENSAVERUNNING,0,@i,0);
end;

```

II-C - Modification du source du projet

La gestion des paramètres sera effectuée dans le source du projet. C'est de loin la méthode la plus simple pour gérer le nombre d'instances et la création de différentes fenêtres suivant les commandes.

A ce stade le code du projet est le suivant :

```

program Project1;

uses
  Windows,
  Forms,
  SysUtils,
  Declare in 'Declare.pas';

```

```

UEcran in 'UEcran.pas' {FenEcran};

// Modification de l'extension
{$E SCR}
{$R *.res}
// Déclaration du nom de l'écran de veille
{$D SCRNSAVE Ecran de veille de Nono}

begin
// On commence par prendre les paramètres passés au programme
Param1 := Copy(UpperCase(ParamStr(1)),1,2);
Param2 := UpperCase(ParamStr(2));

// On filtre le premier afin de ne garder que la lettre
if (Length(Param1)>0)And Not (Param1[1] In ['A'..'Z']) Then
  Param1 := Copy(Param1,2,1);

// On détermine ce qu'il faut faire en fonction de Param1
if Param1='P' Then ssMode := ssPrevisu;
if Param1='C' Then ssMode := ssConfig;
if Param1='S' Then ssMode := ssAffiche;
if Param1='A' Then ssMode := ssMotDePasse;

// Une application écran de veille doit être mono-instance
SetLastError(NO_ERROR);
CreateMutex (nil, False, 'MONSCREENSAVER');
if GetLastError = ERROR_ALREADY_EXISTS Then Exit;

// Traitement en fonction des cas
Case ssMode Of
ssAffiche,ssPrevisu:
  Begin
    Application.Initialize;
    Application.CreateForm(TFenEcran, FenEcran);
    if ssMode=ssPrevisu Then
      FenEcran.ParentWindow := StrToInt(Param2);
    Application.Run;
  End;
End;
end.

```

On notera l'utilisation de la directive \$E afin de modifier l'extension du fichier exécutable en .SCR. Ceci est important afin que Windows le reconnaisse en tant que tel.

Le début du code ne fait que lire et stocker les paramètres passés au programme.

Ensuite, la vérification de la mono-instance de l'application est effectuée.

Enfin l'écran de veille est affiché.

Vous noterez que la différence entre l'affichage normal et la prévisualisation est minime. Dans le cas de la prévisualisation une simple affectation de **ParentWindow** permet à la fenêtre de l'écran de veille de se loger dans le petit moniteur. C'est pour cela que dans le code de l'écran de veille on utilise **ClientWidth/ClientHeight** au lieu de **Screen.Width** et **Screen.Height**. La fenêtre prenant d'elle-même les dimensions de la fenêtre parente (Merci Delphi).

A ce stade, l'écran de veille est fonctionnel. Compilez le projet, puis effectuez un clic droit dans l'explorateur afin d'installer l'écran de veille. Cette méthode permet de le tester sans avoir à le copier

dans le répertoire de base de Windows.

L'écran de veille doit alors apparaître en prévisualisation :

Le bouton "Aperçu" permet de le voir en plein écran, comme en veille normale.

II-D - Ajout d'une fenêtre de configuration

L'ajout d'une fenêtre de configuration est très simple.

Ajoutez une nouvelle fenêtre au projet et donnez lui l'aspect suivant :

Ajouter le code suivant dans le OnShow :

```
procedure TFenConfig.FormShow(Sender: TObject);
begin
  // Lecture et affichage des paramètres
  LitIni;
  seTimer.Value := optVitesse;
  // On masque l'application dans la barre des tâches
  ShowWindow(Application.Handle, SW_HIDE);
end;
```

Et le code suivant sur les boutons :

```
procedure TFenConfig.btnAnnulerClick(Sender: TObject);
begin
  // Pour annuler, on ferme simplement...
  Close;
end;

procedure TFenConfig.btnOKClick(Sender: TObject);
begin
  // En cas de validation il faut sauver les paramtres
  optVitesse := seTimer.Value;
  EcritIni;
  // Puis fermer la fenêtre
  Close;
end;
```

Il ne reste plus qu'à modifier le source du projet pour appeler la fenêtre de configuration quand Windows le demande :

```
// Traitement en fonction des cas
Case ssMode Of
  ssAffiche, ssPrevisu:
  Begin
    Application.Initialize;
```

```

Application.CreateForm(TFenEcran, FenEcran);
if ssMode=ssPrevisu Then
  FenEcran.ParentWindow := StrToInt(Param2);
Application.Run;
End;
ssConfig:
Begin
  Application.CreateForm(TFenConfig,FenConfig);
  Application.Initialize;
  Application.Run;
End;
End;

```

Et voilà, quand vous cliquez sur le bouton "Paramètres" dans les propriétés d'affichage, votre fenêtre de configuration s'affiche.

II-E - Gestion du mot de passe

Pour ajouter la gestion du mot de passe il faut ajouter deux parties dans le code :

- Répondre à la commande **A** de modification du mot de passe.
- Tester le mot de passe dans l'écran de veille.

Pour modifier le mot de passe, comme énoncé plus haut nous allons utiliser les procédures de Windows. Il faut noter aussi que la demande de modification du mot de passe peut être demandée même si l'écran de veille est déjà en prévisualisation. Il faut donc gérer la modification du mot de passe avant la gestion du nombre d'instances. Le source du projet devient :

```

// Traitement en fonction des cas
Case ssMode Of
ssMotDePasse:Begin
  MdpLib := LoadLibrary('MPR.DLL');
  If MdpLib <> 0 Then
  Begin
    MdpFunc := GetProcAddress(MdpLib,'PwdChangePasswordA');
    If Assigned(MdpFunc) Then
      MdpFunc('SCRSAVE',StrToInt(Param2),0,0);
    FreeLibrary(MdpLib);
  End;
  Exit;
End;
End;

// Une application écran de veille doit être mono-instance
SetLastError(NO_ERROR);
CreateMutex(nil, False, 'MONSCREENSAVER');
if GetLastError = ERROR_ALREADY_EXISTS Then Exit;

// Traitement en fonction des cas
...

```

Pour gérer le mot de passe, nous allons simplement utiliser l'évènement **OnCloseQuery** de la fiche de visualisation :

```

procedure TFenEcran.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
Var Reg:TRegistry;
    MdpFunc : function (Parent : THandle) : Boolean; stdcall;
    MdpLib : THandle;

begin
    // On teste le mot de passe éventuel dans le cas de demande de fermeture de la fiche
    CanClose := True;
    If ssMode=ssAffiche Then
    Try
        // Test seulement en affichage normal
        Reg := TRegistry.Create;
        Try
            // Lecture de la clef
            Reg.RootKey := HKEY_CURRENT_USER;
            If Reg.OpenKey('Control Panel\Desktop',False) Then
                If Reg.ReadInteger('ScreenSaveUsePassword') <> 0 Then
                    Begin
                        MdpLib := LoadLibrary('PASSWORD.CPL');
                        If MdpLib <> 0 Then
                            Begin
                                // Appel de la procédure de demande de mot de passe
                                MdpFunc := GetProcAddress(MdpLib,'VerifyScreenSavePwd');
                                CanClose := MdpFunc(Handle);
                                FreeLibrary(MdpLib);
                            End;
                        End;
                    End;
                Finally
                    Reg.Free;
                End;
            Except
                // Sur erreur, par défaut on ferme.
                CanClose:=True;
            End;
        end;
    end;

```

A noter que par défaut (erreur de lecture BDR etc...) il ne faut pas gérer le mot de passe, sous peine d'avoir à rebooter votre PC.

Notez également que dans le cas de la prévisualisation il ne faut pas demander le mot de passe, car l'écran de veille doit se fermer dès que Windows le demande (fermeture de la fenêtre des propriétés d'affichage).



Sous Windows XP la clef n'existe pas. La gestion du mot de passe est effectuée par Windows directement en affichant la fenêtre d'ouverture de session.

III - Conclusion

La gestion d'un écran de veille n'est donc pas si compliquée que ça, libre à vous maintenant de créer des écrans plus compliqués.

Ici l'écran de prévisualisation est le même que l'écran final, mais ce n'est pas obligatoirement le cas.

Téléchargez ici le source complet du projet : [Source0084.zip](#)

Version PDF de cet article :

Miroir 1 : [Version PDF](#)

Dans le cas où le miroir 1 ne fonctionne pas :

Miroir 2 : [Version PDF](#)

Lien : [Prévisualisation d'un écran de veille](#)

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