Delphi Firemonkey : Charger un style

Comment charger et appliquer un style sous Windows ?

Par Serge Girard

Date de publication : 12 février 2025

CONFIRMÉ

L'objectif de ce tutoriel est de vous montrer les diverses manières de stockage et de chargement de style pour les applications Delphi multi-plateforme.

Ce tutoriel fait suite à mon **Introduction sur les Styles FMX** et explore plus en détail ce que le **chapitre II** a commencé à montrer.

Cette démonstration est faite à partir de la dernière version gratuite disponible à ce jour (c'est-à-dire la version 12 Community).

VERSION NON DÉFINITIVE

En complément sur Developpez.com

- Introduction aux styles FMX

I - Solution au design de l'application

Dans mon introduction aux styles, j'ai fait la part belle à l'utilisation d'un composant : TStyleBook. Pour charger un fichier de style (**.style** ou **.fsf**) une fois le concepteur de style ouvert (pour rappel : en double cliquant sur le composant) , la seule action à faire est de choisir le bouton de chargement et de sélectionner le fichier que vous voulez utiliser.

Une fois le concepteur fermé, ce fichier, sous forme compressée, est inclus dans le fichier .fmx associé à l'unité .pas concernée.

⚠ Surtout n'oubliez jamais de fermer le concepteur pour que les modifications soient appliquées.

Quand vous ne voulez appliquer qu'un seul style, ne cibler qu'une seule plateforme, c'est l'idéal. Si cette application contient plusieurs fenêtres, indiquer que ce style sera celui qu'utilisera le manager de style (**StyleManager**) via la propriété **UseStyleManager** est la solution la plus simple mais, a priori, cette propriété n'a d'impact qu'en mode design.

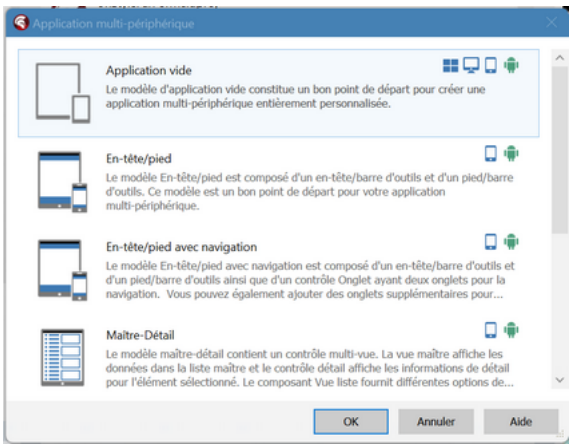
Alors, corsons la chose, comment faire quand l'on veut proposer à l'utilisateur final plusieurs thèmes, a minima, comme cela semble désormais la règle, un thème clair et un thème sombre ?

I-A - Une application Windows simple avec deux thèmes

Si vous avez déjà lu mon introduction, vous en avez déjà les principes.

Je vous propose de créer une application multi-plateforme vide.

Option du menu de l'IDE Delphi *: Fichier/Nouveau/Application multi-périphérique*.



Cet assistant va nous créer le projet et une première unité contenant une fenêtre vide.

Avant même d'aller plus loin, je vous invite à ajouter un module de données au projet.

Option du menu de l'IDE Delphi *: Fichier/Nouveau/Module de données.*

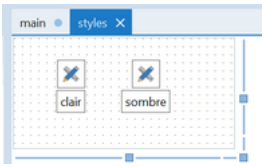
Avant même de sauvegarder ce projet vide :

- Donnez un nom autre que « **form1 ou datamodule*1*»** aux deux éléments créés (en modifiant la propriété **Name**) ;
- Allez dans les options de projet et faites en sorte que le module de données soit créé en premier.

Option du menu de l'IDE : *Projet/Options (Ctrl+MAJ+F11)*
puis, dans arborescence des options : Application/Fiches ;

- Enfin, sauvegardez le tout, de préférence dans un nouveau répertoire et avec des noms différents que ceux proposés par défaut (unit1, unit2, project1).
  Option du menu de l'IDE Delphi *: Fichier/Tout enregistrer (Ctrl+MAJ+S).*

Ouvrez alors, l'unité module de données, dans laquelle vous allez déposer deux composants **TStyleBook**.



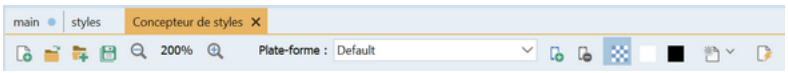
Pour cette petite démonstration, cible Windows, nous allons charger pour le premier **TStyleBook** le fichier **light.style** et le nommer **« clair »,** pour le second, à nommer **« sombre »,** le fichier **dark.style** conviendra parfaitement à notre propos.

Pour rappel, charger un fichier de style au design, se fait en passant par l'onglet « Concepteur de styles ». Le plus simple pour ouvrir ce concepteur est de double-cliquer sur le composant TStyleBook. Pour sélectionner un fichier, utilisez le bouton

Ouvrir

de la barre de menu de l'onglet.

Vous retrouverez ce petit programme en téléchargement **dans cette archive**.



N'oubliez pas de valider ces modifications en fermant le concepteur de styles.

Mon conseil, pour votre tranquillité, faites en sorte qu'il n'y ait toujours qu'un seul onglet de ce type.

Pour rappel, les fichier de style, de type **.style** ou **.fsf** se retrouvent, en cas d'installation « standard » dans les répertoires :

C:\Users\Public\Documents\Embarcadero\Studio\23.0\Styles
ou

C:\Program Files (x86)\Embarcadero\Studio\23.0\Redist\styles\Fmx.

Contrairement à ce que j'ai indiqué au début du chapitre I, je ne renseignerai pas **UseStyleManager** et ce pour aucun des deux composants. En effet il ne peut y avoir qu'un seul **TStyleBook** qui peut avoir cette propriété active.

La question est donc de savoir comment appliquer le bon style à l'exécution.

Votre premier réflexe serait de faire comme on le ferait au moment du design, à savoir indiquer à la forme créé le nom du style à utiliser, c'est-à-dire renseigner la propriété **StyleBook** de la fenêtre, quelque chose comme ceci.

exemple
```
If DataStyle.modesombre
   then StyleBook:=DataStyle.sombre
   else StyleBook:=DataStyle.clair;
```

Pas si idiot le réflexe, au moment du design, pour avoir un aperçu de l'interface utilisateur (WISIWIG).

I-A-1 - Coin du mécano : L'utilisation de la propriété Stylebook

L'instruction SetStyleBook(untstylebook) fait la même assignation.

Inconvénient de l'utilisation de la propriété ***StyleBook***, les fenêtres « enfants » que vous pourriez ouvrir n'utiliseront pas ce style, sauf bien sûr à l'indiquer lors de la création de la fiche enfant.

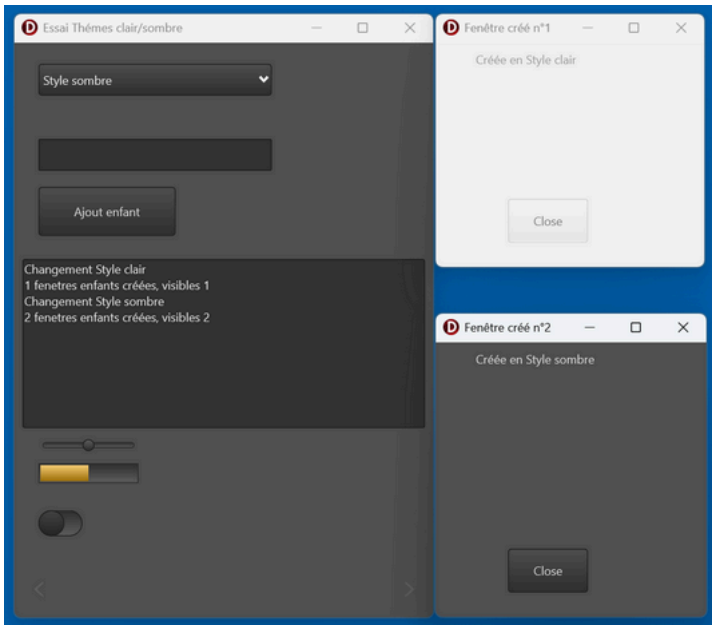
Pour en faire la démonstration, j'ai repris le programme de cette démonstration et ajouter un fiche qui sera crée à l'exécution.
Programme téléchargeable ici

```
procedure TMainform.btnAddChildClick(Sender: TObject);
var f : TChildForm;
begin
 // Utilisation de compteurs pour le debogage
 inc(nbchild);
 inc(nbCreation);
 // création de la fenêtre enfant
 f:=TChildForm.Create(Self);
 f.Caption:=format('Fenêtre créé n°%d',[nbcreation]);
 f.Label1.text:='Créée en '+Combobox1.Items[combobox1.ItemIndex];
 f.parent:=Self;
 f.StyleBook:=StyleBook;
 f.Show;  // affichage de la fenêtre
 memo1.lines.add(format('%d fenêtres enfants créées, visibles %d',[nbCreation,nbChild]));
end;
```

Un problème se pose alors, si l'utilisateur décide de changer de style alors que des formes enfants existent, celles-ci ne seront pas modifiées.

Ci-dessous, le scénario est le suivant :

- Ouverture du programme, la détection du mode fait que la fenêtre utilise le style clair ;
- Création d'une fenêtre enfant ;
- Sélection du style sombre, la fenêtre enfant ne change pas de thème ;
- Création d'une seconde fenêtre enfant, cette nouvelle fenêtre utilise bien le style sombre.



C'est d'ailleurs pour cela que le code f.parent:=Self; a été écrit

afin de changer de réinitialiser la propriété ***StyleBook*** de celles-ci.

```
// Balaye les composants de la fiche principale
for var i := 0 to ComponentCount-1 do
 begin
   if Components[i] is TChildForm then
     TChildForm(Components[i]).StyleBook:=StyleBook;
 end;
```

I-B - Utiliser UseStyleManager

Autre solution envisageable, activer **UseStyleManager** en fonction du thème, mais, comme il ne peut y avoir qu'un seul ***TStyleBook*** de marqué ainsi. En théorie, cela nécessiterait de désactiver le ***TStyleBook*** préalablement activé et d'activer le second.

```
If DataStyle.modesombre then
 begin
  DataStyle.clair.UseStyleManager:=False ;
  DataStyle.sombre.UseStyleManager:=True ;

end
else begin
  DataStyle.sombre.UseStyleManager:=False ;
  DataStyle.clair.UseStyleManager:=True ;
end ;
```

Malheureusement, comme je vous l'ai indiqué en introduction de ce chapitre, cette propriété n'a d'impact qu'en mode design, donc, non valide.

S'il n'y avait qu'un seul TStyleBook, évidemment la question ne se poserait pas en ces termes, la question deviendrait :
« Comment charger un nouveau style à l'exécution ? », ce sera l'objet du chapitre **II.D** et **II.E**

Bref, ce n'est pas très pratique. À la place, je vais vous parler de ***TStyleManager***.

I-C - Utilisation de StyleManager

À ne pas confondre avec la propriété ***UseStyleManager*** du composant ***TStyleBook***. Pour pouvoir l'utiliser nous devrons déclarer l'utilisation de l'unité **FMX.Styles** dans la liste des unités utilisées mais, avant toutes choses, il faudrait aussi savoir détecter le thème actuel du système d'exploitation.

I-C-1 - Détecter le thème à afficher

Pour cela, je vais faire appel au service **IFMXSystemAppearanceService**, ce qui va nécessiter l'ajout de l'unité **FMX.Platform**. Si le service est présent, il récupère le type de thème (**TSystemThemeKind**) actuel.

Cette détection va se faire dès la création du module de données, l'évènement **OnCreate**.

```
    { Déclarations publiques }
     modesombre: boolean;

procedure TDataStyle.DataModuleCreate(Sender: TObject);
var
  svc: IFMXSystemAppearanceService;
begin
  // détection du thème "système"
  if TPlatFormServices.Current.SupportsPlatformService
    (IFMXSystemAppearanceService, svc) then
    modesombre := svc.ThemeKind = TSystemThemeKind.dark
  else
    modesombre := false;
end;
{Voici ce que fait ce code :

1. **Détection du thème "système"** : Il vérifie d'abord si le service
 `IFMXSystemAppearanceService` est présent.
    Ce service est utilisé pour déterminer le thème actuel du système d'exploitation.
    Si le service est présent, il récupère le type de thème (`TSystemThemeKind`) actuel.
2. **Définition de la variable `modesombre`** : En fonction du type de thème, la variable
 `modesombre` est définie :
   - Si le thème est sombre (`TSystemThemeKind.dark`), `modesombre` est défini sur `true`.
   - Sinon, `modesombre` est défini sur `false`.
```

Une fois le thème détecté, une procédure, à créer, va appliquer le style voulu (voir chapitre **I.C.2**).

```
    { Déclarations publiques }
    procedure changestyle(const dark: boolean ;
                          const owner : TComponent =nil);
```

Un programmeur Delphi confirmé préférera, peut-être, définir et utiliser une propriété. Pour tout public, je préfère en rester à cette version.

Pourquoi une variable **modesombre** ? Pour obtenir ce renseignement facilement avec les autres unités du programme.

Quant à la procédure, le fait quelle soit publique, va me permettre de l'utiliser dans les autres unités du programme afin de changer de thème selon la décision de l'utilisateur.

I-C-2 - La procédure SetStyle

Pour appliquer le style, je vais utiliser la procédure **SetStyle** du **TStyleManager**.

Tout d'abord, une petite précision sur l'argument de cette procédure.

La syntaxe de celle-ci **TStyleManager.SetStyle(const Style: TFmxObject)** peut porter à confusion et vous aurez tendance à indiquer comme valeur, le nom du TStyleBook.

ERREUR
```
TStyleManager.SetStyle(clair) ;
```

L'analyseur syntaxique ni verra aucune erreur mais, à l'exécution vous aurez la désagréable surprise d'obtenir une fenêtre sans représentation des composants !

Désagréable, n'est-ce pas ? J'avais posé plusieurs contrôles dans cette forme, seul truc visible, le curseur d'un **TEdit**.

Mais, j'anticipe, je ne vous ai pas encore décrit la fenêtre principale.

La bonne formulation serait :

OK
```
TStyleManager.SetStyle(clair.Style) ;
```

Mais, il y a un hic, tentez de basculer plus d'une fois sur le même style et c'est à nouveau un écran vide qui s'affiche. Le coupable ? TStyleManager semble libérer les données qu'on lui a fournies si on change le style. La solution : cloner le style. En clonant le style, le contenu n'est plus effacé.

```
procedure TDataStyle.changestyle(const dark: boolean);
begin
  // application du thème "système"
  if dark then
     TStyleManager.SetStyle(sombre.Style.Clone(self))
   else
    TstyleManager.SetStyle(clair.Style.Clone(self));
  // mémoriser le thème appliqué
  modesombre := dark;
end;
```

Utiliser un clonage n'est qu'une question d'habitude de codage, c'est un peu perturbant et semble insensé mais il faut faire avec.

I-C-3 - Le test, design de la fenêtre principale

Comme vous le constaterez dans l'image, j'ai posé plusieurs composants :

- Les classiques d'une application de démonstration : un **TEdit**, un **TButton** et un **TMemo** ;
- Quelques composants, qui ne sont là que pour étoffer la présentation. Un **TTrackbar**, un **TProgressBar** et un **TSwitch**. (aucune action particulière sur ceux-ci) ;
- Un **TComboBox**, boite de choix qui va permettre à l'utilisateur de sélectionner le thème. Trois choix seront proposés :

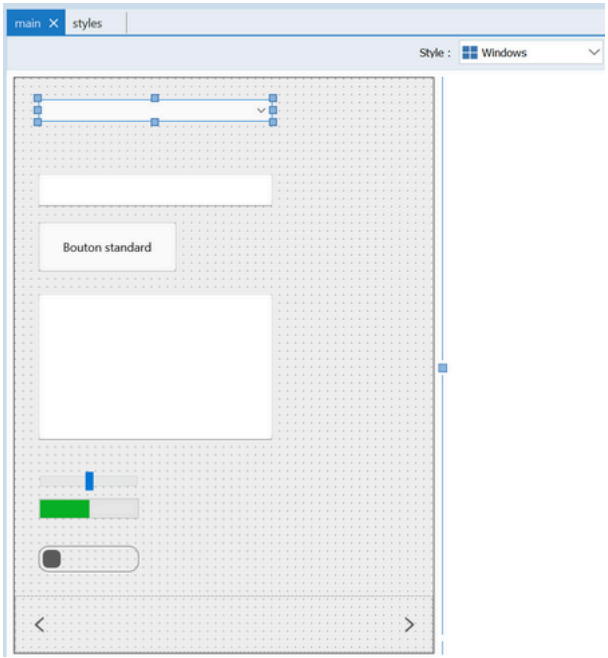
- Le style par défaut (pour démontrer qu'il existe),

- Le style clair,

- Le style sombre.

Et enfin, en bas de la fenêtre, un TLayout contenant deux boutons, respectivement l'un aligné à gauche de **stylelookup=arrowlefttoolbutton**

et l'autre à droite, **stylelookup=arrowrighttoolbutton** . Ceci juste pour vous faire une petite surprise.



Côté code, tout d'abord, if faut bien indiquer dans la clause uses que nous utiliserons l'unité du module de données **DataStyle** et l'unité **FMX.Styles**.

Côté code, l'évènement **OnCreate** de la fenêtre nous permettra de définir l'index de la boite de choix au démarrage de l'application.

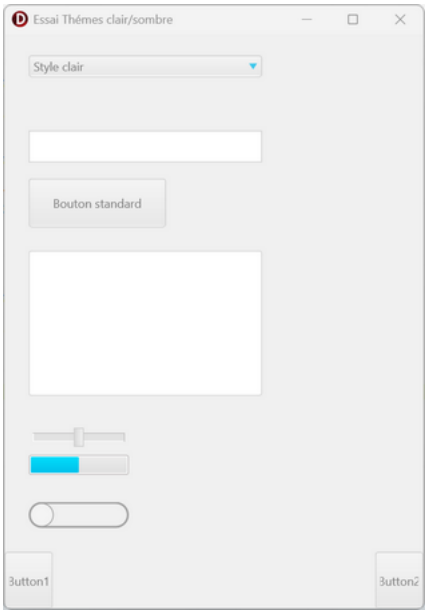
```
procedure TMainform.FormCreate(Sender: TObject);
begin
if DataStyle.modesombre then combobox1.ItemIndex:=2
                        else combobox1.ItemIndex:=1;
end;
```

Le plus intéressant, reste bien évidemment, l'application des choix de thème par l'utilisateur via la boite de choix.

```
procedure TMainform.ComboBox1Change(Sender: TObject);
begin
case combobox1.ItemIndex of
 0 : TstyleManager.SetStyle(nil); // style par « défaut »
 1 : DataStyle.changestyle(false); // style clair
 2 : DataStyle.changestyle(true); // style sombre
end;
end;
```

Une fois le code complété, exécutez le programme (F9, mode débogage).

Si, comme moi, le thème du système d'exploitation est clair, vous obtiendrez ceci :

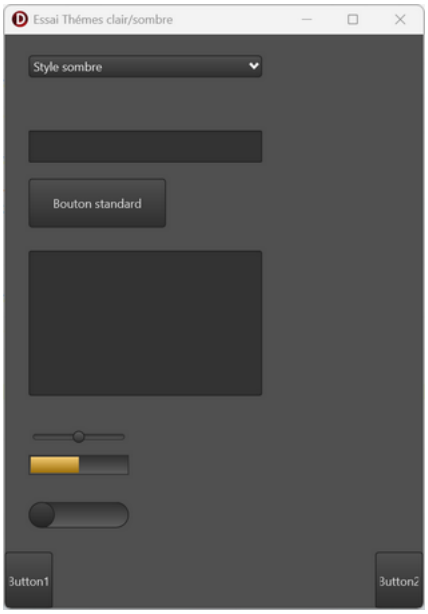


S'il y avait une conclusion à en tirer, ce serait :

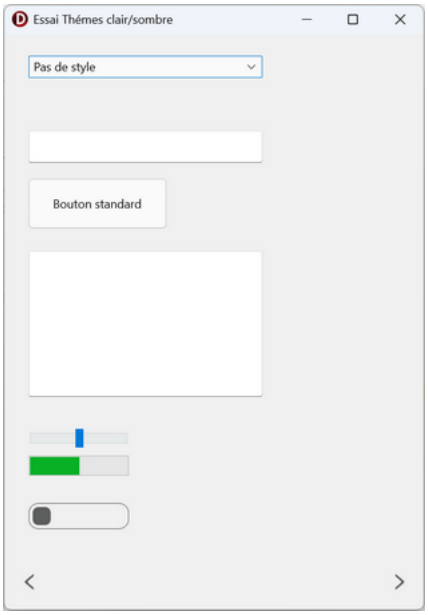
- de bien choisir les fichiers de style à charger,
- de tester au design les divers **TStyleBook** pour éviter de désagrément.

Plus vous avancerez dans le domaine des styles, plus vous penserez à la nécessité d'une charte graphique.

Faites ensuite, le choix du style sombre.



Puis du style par défaut.



Chouette, nous retrouvons le design.

Cela démontre aussi, s'il le fallait encore, qu'il existe quelque part, une représentation de ce style. En fait, dans le cas de Windows, dans les ressources du programme.

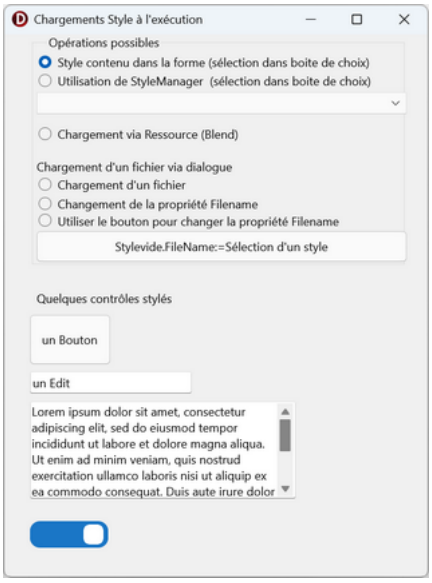
En attendant, découvrons les autres possibilités offertes.

II - D'autres manières de charger les fichiers de style

Je vais encore rester dans l'environnement Windows pour faire le tour des possibilités.

Un nouveau programme, à télécharger

Le concept, pour la démonstration, est différent du premier chapitre. L'objectif étant de tester les diverses méthodes utilisables, recensées dans la boite de groupe sur cette image.



II-A - Style contenu dans la fiche de la fenêtre

Comme relaté chapitre **I.A** et détaillé en **I.A.1**: il est tout à fait possible d'utiliser la propriété **StyleBook** d'une fenêtre.

Dans cette nouvelle application, au niveau de la fenêtre principale, j'ai ajouté plusieurs **TStyleBook**s.

Une sélection dans la boîte de choix, qu'un peu de code va remplir.

```
procedure TMain.FormCreate(Sender: TObject);
var
  Index: Integer;
begin
  // initialisations de la boite de choix
  ComboBox1.Items.add('défaut');
  // énumère les TStyleBook disponibles
  EnumObjects(
    function(AObject: TFmxObject): TEnumProcResult
    begin
      if AObject is TStyleBook And not Sametext(TStyleBook(AObject).Name,'stylevide') then
      begin
        Index := ComboBox1.Items.add(TStyleBook(AObject).Name);
        ComboBox1.ListItems[Index].Data := AObject;
      end;
      Result := TEnumProcResult.Continue;
    end);
end;
```

Ceci va nous permettre de tester une première possibilité :

II-B - Utiliser la propriété StyleBook de la fiche

Déjà largement abordé chapitre **I.A** et surtout **I.A.1**, il faut donc juste ajuster son utilisation dans ce nouveau contexte.

II-C - L'utilisation de la propriété FileName d'un TStyleBook

Bon, elle existe donc il fallait essayer, même à mon corps défendant.
C'est pour ce faire que j'ai ajouté un ***TStyleBook*** nommé ***StyleVide***.

Au design, c'est simple, changez la propriété.

Problème, aucune boite de sélection de fichier ne s'ouvre. Il faut donc obtenir le chemin complet du fichier d'une autre manière.

⚠ Il faudra, bien s'assurer que le fichier existe sous peine d'une erreur, surtout en mode design.

Cependant, au design, effacer cette valeur de propriété est un vrai casse tête, générateur d'erreur(s) !

💡 La meilleure solution dans ce cas est , à mon avis, de supprimer le composant et de le recréer.

À l'exécution par contre, utilisé en relation avec un dialogue d'ouverture de fichier, c'est assez intéressant.

```
procedure TMain.btnLoadClick(Sender: TObject);
begin
    if opendialog1.Execute then
     begin
      // s'assure que le composant Stylevide est bien vide
      StyleVide.Styles.Clear;
      // Force une mise à jour de la forme
      UpdateStyleBook;
      // Indique un nom de fichier contenant un style valide
      StyleVide.FileName:=Opendialog1.FileName;
      // Indique que la forme utilise le composant Stylevide
      SetStyleBook(StyleVide);
      // Notifier au StyleManager de redessiner la/les fenêtre(s)
      TStyleManager.UpdateScenes;
     end;
end;
```

⚠ Attention, le composant **TOpenDialog** n'est pas supporté par toutes les plateformes.

II-D - Le chargement d'un fichier de style à l'exécution

C'est à peu près le même comportement que changer la propriété **FileName** du **TStyleBook**.

ℹ **Avantage :** contrairement à l'utilisation de **FileName**, pas d'erreur en mode design.

Inconvénient : pas de **WISIWIG** au design (sauf à faire des tests « manuels » via le concepteur de styles).

Cela correspond, un peu, à ce que vous faites lorsque vous utilisez le concepteur de styles et que vous chargez un fichier de style.

```
procedure TMain.rbLoadFileChange(Sender: TObject);
begin
    Stylebook := nil; // s'assure que la propriété est effacée
    if OpenDialog1.Execute then
    begin
      TStyleManager.SetStyleFromFile(OpenDialog1.FileName);
    end;
```

```
end;
```

La clé est donc d'utiliser une des fonction proposée par **TStyleManager** : **SetStyleFromFile**.

Pour un public confirmé, je signalerai qu'il est possible d'utiliser les **streams,** l'idée de stocker des fichiers de style dans une base de données, une colonne de type blob (texte ou binaire selon le format du fichier), est donc envisageable.

La fonction retourne un booléen en cas de succès et surtout, ne modifie pas l'affichage en cas de format incompatible, il est donc possible de faire un traitement particulier (envoyer un message étant le plus évident) dans ce cas.

```
if Not TstyleManager.SetStyleFromFile(Opendialog1.Filename) then  ShowMessage('Style
 incompatible') ;
```

II-E - L'utilisation des ressources

Proche du chargement d'un fichier, **TStyleManager** propose une autre fonction **TrySetStyleFromResource** dont le principe est similaire.

Pour utiliser ceci, il s'agit plus de savoir comment et où ajouter les fichiers de styles aux ressources du programme que de coder.

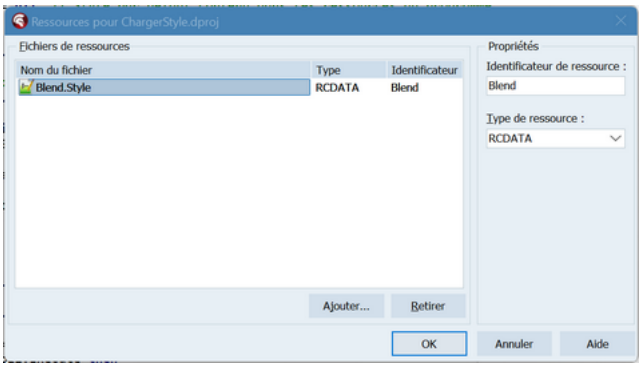
- Le code est simple, dans les ressources du programme, j'ai ajouté une ressource identifiée '**Blend**', je l'utilise ainsi :

```
procedure TMain.rbLoadResourceChange(Sender: TObject);
begin
    TStyleManager.TrySetStyleFromResource('Blend');
end;
```

II-E-1 - Ajouter une ressource

Utilisez le menu de L'IDE : *Projet/Ressources et images…*

Vous obtiendrez ce dialogue.



Sélectionnez alors le fichier de style que vous voulez mettre.

Pour pouvoir trouver les fichiers de style FireMonkey (pour rappel *.style ou *.fsf) vous devrez utiliser le filtre « Tous les fichiers (*.*) ».

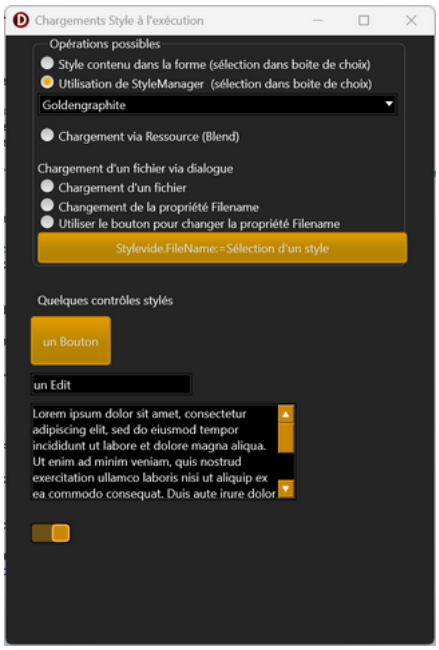
Une fois le fichier chargé, indiquez un identificateur unique.

Vous pouvez ensuite répéter l'opération autant de fois que de fichier que vous voulez distribuer.

Surtout, n'oubliez pas de valider en utilisant le bouton [OK].

II-F - Exécution du programme

Impossible de fournir les images de toutes les possibilités proposées par celui-ci.
Pour vous appâter :



III - Et pour une autre plateforme ?

À ce point, étant donné que j'ai veillé à n'utiliser que des styles "neutres", pouvant être utilisés sur toutes les plateformes, il serait possible d'envisager une compilation pour d'autres plateformes.

La cible la plus probable qui vous viendra à l'esprit est Android. Est-ce que ces deux applications vont être déployable telles quelles ?

Malheureusement non, le clonage des styles tel que je l'ai montré sous Windows (chapitre **I.C.2**), ne fonctionne pas sous les autres plateformes.

III-A - L'utilisation de TStyleManager

Cloner le style est donc à bannir. Mais, si vous ne proposez pas à l'utilisateur de bascule successives vous pourrez quand même utiliser ***SetStyle*** ainsi :

```
{$IFDEF ANDROID}
 TStyleManager.SetStyle(aStyleBook.Style);
 TStyleManager.UpdateScenes ;
{$ENDIF}
```

Vous utiliserez aussi ***TStyleManager*** pour charger un fichier ou une ressource.

Dans ces deux cas , le problème sera plus un problème de déploiement.

Dans le cas des fichiers,

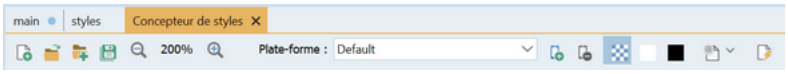
III-A-1 - Le coin du mécano

J'ai quand même découvert quelque chose d'assez intéressant lors des essais. Si vous revenez au chapitre , j'y indiquais que les styles était contenus dans le source de la fenêtre. Cette petite instruction

```
{$R *.fmx}
```

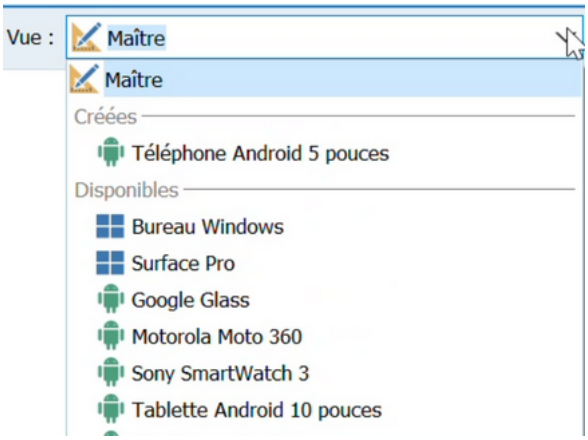
indique au compilateur d'inclure tous les fichiers `.fmx` dans le répertoire courant et ses sous-répertoires dans la compilation. Ces fichiers contiennent généralement l'interface utilisateur des formulaires créés.

Firemonkey permet de créer des vues, c'est à ça que cette boite de choix lors du design

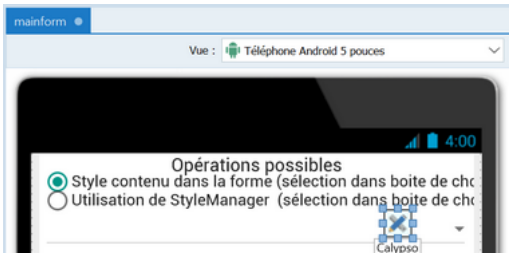


sert et elle vous intriquait peut-être ?

Sélectionnez « Téléphone Android 5 pouces »



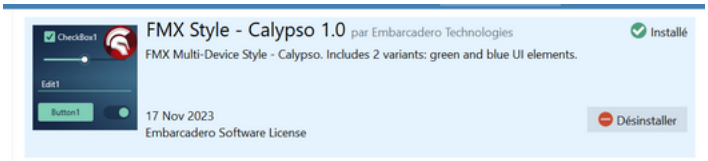
et, non seulement créer la vue va vous permettre de visualiser le cadre de la fiche mais vous pourrez réaligner les composants, ce qui est fortement nécessaire pour ce programme.



Conséquence, dans le source de l'unité, une nouvelle ligne a été ajoutée.

```
{$R *.fmx}
{$R *.LgXhdpiPh.fmx ANDROID}
```

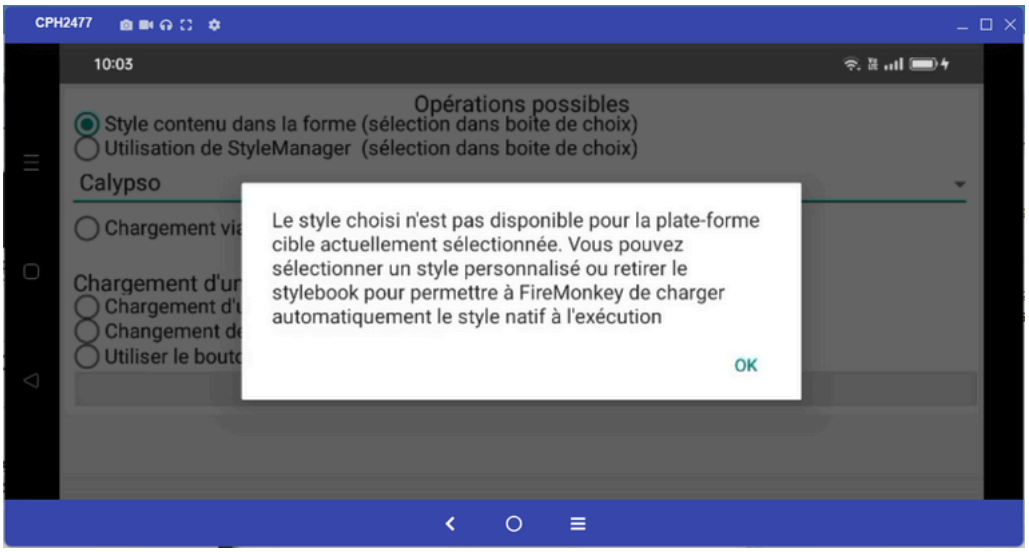
Est-ce là le seul bénéfice ? Je vous avais indiqué que je n'avais utilisé que des styles, a priori, multiplateforme. Pour l'expérience j'ai ajouté un TStyleBook, **calypso** (téléchargeable via Getit)



contenant, bien évidement selon la vue, le bon fichier : **Calypso_Win.style** ou **Calypso_Android.style**.

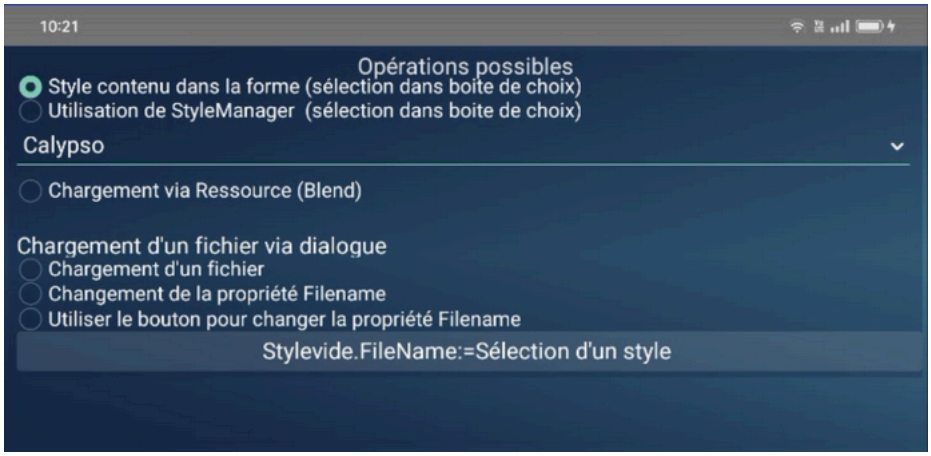
En cas de non compatibilité que pourrait-il se passer ? L'image suivant vaut plus que tout discours.

J'ai obtenu ce résultat avant d'utiliser **UpdateScenes**. Dans le programme si vous voulez tester, il vous suffira de commenter la ligne.



Une fois **UpdateScenes** codé, le résultat est conforme à mon attente pour les deux premiers choix.

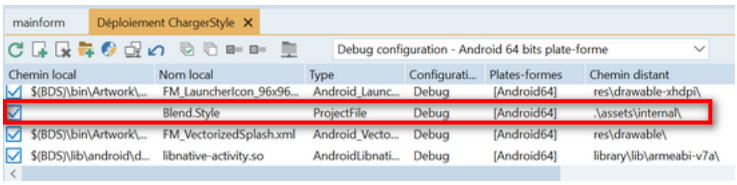
UpdateScenes ! Là où les habitués de VCL auraient le réflexe d'utiliser des instructions comme **Repaint** ou **InValidate**, en ce qui concerne les styles, c'est bien cette instruction qu'il faudra utiliser.



III-B - L'utilisation d'une ressource programme

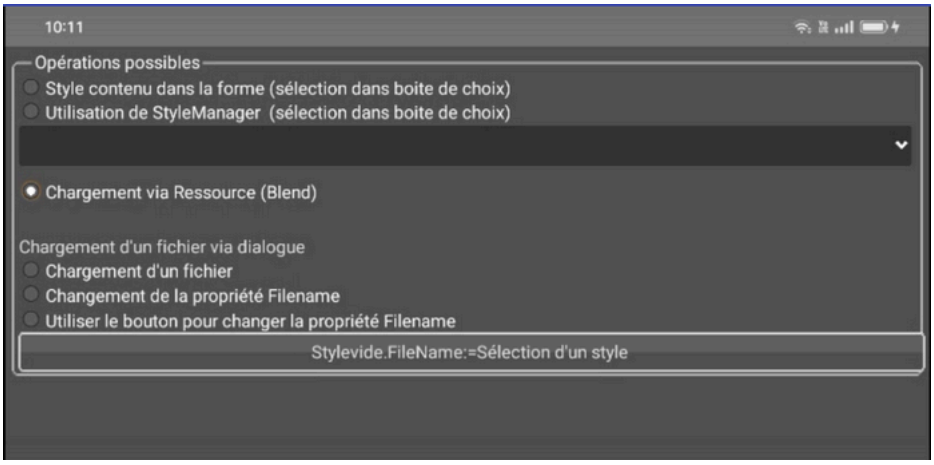
Un programme Android ne contient pas de ressource mais Delphi va faire en sorte lors de la construction du package que le fichier soit copié dans un endroit accessible ***./assets/internal***.

Utilisez l'option du menu de l'IDE : *Projet/Déploiement*



Rien à changer dans le code, l'IDE se charge de déployer le fichier au « bon endroit ».

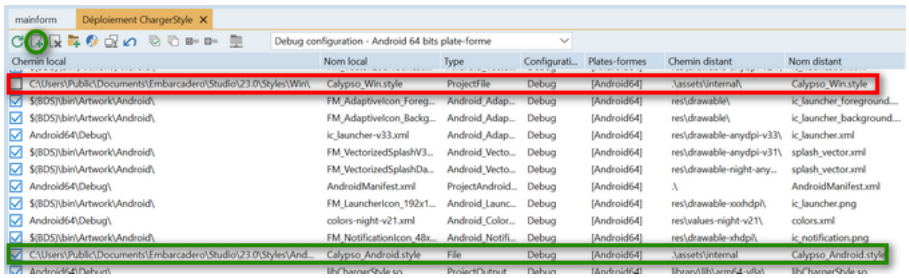
Un petit test le confirme



III-B-1 - Le coin du mécano

Attention quand même, si vous avez déclaré en ressource, un fichier non compatible, par exemple **Calypso_Win.Style** avec comme identifiant **calypso**, bien évidemment il ne faudra pas le déployer. Donc décocher la case correspondante.

À la place, on ajoutera le fichier **Calypso_Android.Style** en prenant soin de bien indiquer de l'installer dans **./assets/internal**.



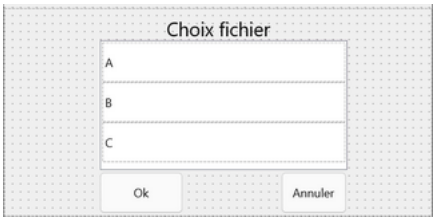
⚠ Bonne idée de procéder ainsi ? Eh bien, non, deux choses coincent :

- Le programme Windows va contenir une ressource, d'ailleurs assez importante en taille d'octets, qui ne sert à rien,
- Il sera nécessaire de coder quelque chose pour obtenir le bon nom de ressource selon la cible. Je vous rappelle qu'un identifieur doit être unique.

Donc, ce n'est pas encore la panacée, toutefois cela m'a permis d'introduire le déploiement d'un projet Android et, si vous avez remarqué le petit cercle vert dans l'image, la possibilité d'ajout de fichiers.
Cela va introduire la dernière méthode proposée dans le chapitre **II.D**

III-C - Chargement d'un fichier de style à l'exécution.

Malheureusement, Android n'accepte pas **TOpenDialog**. Avant tout il nous faut donc créer un pseudo dialogue de même genre. J'ai opté pour le plus simple en concoctant un TLayout.




```
{$IFDEF ANDROID}
 // Charger les fichiers disponibles
 // ceux qui seront mis au cours du déploiement dans ./assets/internal
 ListBox1.Items.Clear;
 var sl:=TDirectory.GetFiles(TPath.GetDocumentsPath,'*.style');
 for var s in  SL do ListBox1.Items.Add(extractfilename(s));
 sl:=TDirectory.GetFiles(TPath.GetDocumentsPath,'*.fsf');
 for var s in  SL do ListBox1.Items.Add(extractfilename(s));
{$ENDIF}
```

Quelques ajustements seront alors nécessaires pour récupérer le chemin entier du fichier avant chargement via l'instruction **SetStyleFromFile** lors de l'utilisation du bouton de validation.

```
  var stylefilename :=Listbox1.Items[Listbox1.ItemIndex];
  stylefilename:=TPath.Combine(Tpath.GetDocumentsPath,stylefilename);
  TStyleManager.SetStyleFromFile(stylefilename) ;
```

Le programme propose deux choix :

- Utiliser **TStyleManager** pour charger la fichier ou,
- Utiliser la propriété **FileName** du **TStyleBook**.

Le code fourni est donc plus complexe

```
procedure TMain.btnOkClick(Sender: TObject);
begin
{$IFDEF ANDROID}
if ListBox1.ItemIndex>-1  then
begin
  Stylebook:=nil;
  var stylefilename :=Listbox1.Items[Listbox1.ItemIndex];
  stylefilename:=TPath.Combine(Tpath.GetDocumentsPath,stylefilename);
 // Utilisation de TStyleManager
 if rbLoadFile.ischecked then TStyleManager.SetStyleFromFile(stylefilename)
 // Utilisation de Self.StyleBook
  else begin
   // Force une mise à jour de la forme
```

```
    Self.UpdateStyleBook;
   // Indique un nom de fichier contenant un style valide
    StyleVide.FileName:=stylefilename;
  // Indique que la forme utilise le composant Stylevide
    StyleBook:=StyleVide;
  // Notifier au StyleManager de redessiner la/les fenêtre(s)
      TStyleManager.UpdateScenes;
   end;
end;
  LytOpenDialog.Visible:=false;
{$ENDIF}
end;
```

Notez que, dans le choix d'utilisation de la propriété **StyleBook** de la fenêtre, l'appel à **UpdateScenes** reste un impératif.

IV - Débriefing

Je vous ai présenté la plupart des méthodes (oui, il en reste quelques autres) pour charger un style à l'exécution. Dans les tableaux ci-dessous le récapitulatif par OS cible.

Windows		
Utilisation d'une ressource	Chapitre **II.E**	Uniquement en utilisant **TStyleManager**
Utilisation des TStyleBook de la fenêtre	Chapitre **II.A**	
Propriété **StyleBook** de la fenêtre ou **SetStylebook**		Problème des fenêtres enfants Chapitre **I.A.1**
Utilisation de **TStyleManager**		En utilisant le clonage
Chargement d'un fichier	Chapitre **II.D**	Uniquement en utilisant **TStyleManager**
Propriété **FileName** d'un **TStyleBook** & Propriété **StyleBook** de la fenêtre		L'un ne peut se faire sans l'autre, la technique consistant à créer un TStyleBook puis modifier la propriété de la fenêtre
Utilisation de **TStyleManager**		TStyleManager.

Autres OS		
Utilisation d'une ressource	Chapitre **III.B**	**TStyleManager.GetStyleResource**
Utilisation des TStyleBook de la fenêtre	Chapitre **II.A**	
Propriété **StyleBook** de la fenêtre		
Utilisation de **TStyleManager**		Clonage ne fonctionne pas
Chargement d'un fichier		
Propriété **StyleBook** de la fenêtre		
Utilisation de **TStyleManager**		
Propriété **FileName** d'un **TStyleBook**		

	W I	A N	M A	I O	L I	

	N D O W S	D R O I D	C O S	S	N U X	
Self.StyleB ou SetStyleBo						Inconvénient ne s'applique pas aux fenêtres enfants
StyleMana						On ne peut utiliser un StyleBook.Style qu'une seule fois Sous Windows, il est possible de cloner le style
StyleMana						Le plus compliqué est d'indiquer le bon emplacement. Attention à la casse pour tout autre OS que Windows.
TStyleBook						Non modifiable à l'exécution
TStyleBook						

V - À lire ou regarder

Ajouter une selecteur de style à votre application

Astuces Firemonkey de Sarina Dupont

using-custom-styles-in-rad-studio-10-3