

Delphi FMX : L'utilisation de TStyleBook

**Diverses manières d'utiliser le composant
TStyleBook pour enjoliver votre interface utilisateur**

Par [Serge Girard](#)  

Date de publication : 14 février 2025

DÉBUTANT

Le développement à destination de plateformes multiples (Windows, Android, iOS...) impose de créer des applications FMX (FireMonkey). Une interface « moderne » nécessitera d'utiliser un ou des styles via l'intermédiaire du composant ***TStyleBook***. Où placer ce composant, quelles conséquences seront liées à ce placement, que changent certaines propriétés de ce dernier ? C'est l'objectif de ce tutoriel.



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

Commentez

I - Introduction.....	3
I-A - Créer une application FMX.....	3
I-B - Plus de styles.....	6
I-C - Comment choisir un style ?.....	6
I-D - Mais où se trouve ce style dans mon programme ?.....	7
II - La bonne utilisation des styles.....	7
II-A - Codes et propriétés des diverses unités.....	9
II-A-1 - Le projet.....	9
II-A-2 - Le module de données.....	10
II-A-3 - La fenêtre secondaire.....	12
II-A-4 - Unité principale.....	14
II-A-4-a - Charger un fichier de style (extensions .style ou .fsf).....	14
II-A-4-b - Le commutateur pour utiliser un style par fenêtre.....	16
II-A-4-c - L'ouverture d'une fenêtre modale.....	16
II-A-4-d - La technique d'ancrage d'une fenêtre.....	18
III - Le coin du mécano.....	19
III-A - « Je n'utilise pas les styles ».....	19
III-B - Mises en garde.....	20
III-B-1 - Au design.....	20
III-B-2 - ShowMessage et autres dialogues.....	22
III-C - Showmodal ou Show.....	23
III-D - Personnalisation.....	24
IV - Bibliographie, visionnages intéressants.....	26
IV-A - Livres.....	26
IV-B - Liens.....	26
V - Débriefing.....	27
VI - Remerciements.....	27

I - Introduction

Il n'est pas dans mon intention dans cet article d'aborder toutes les versions de Delphi permettant le développement multiplateforme, mais j'aimerais quand même indiquer ici que si ce tutoriel s'écrit autour de la version 12 ([téléchargeable ici](#)), il est en grande partie applicable à partir des versions supérieures ou égales à 10.

L'objectif n'est pas non plus de faire des comparatifs avec les apparences Windows proposées par la VCL, ce que je réserve pour un futur ouvrage dont, certainement, cet article fera partie.

Si je dois définir ce qu'est un style à un néophyte Delphi, je dirais qu'il s'agit d'une sorte de papier peint ou modèle qui va dessiner les divers composants sur l'écran.

À un amateur de Delphi, en particulier de la version référence : Delphi 7, je lui proposerai le terme de « Skin », bien qu'il y en ait plusieurs types allant de celles où il faut utiliser des composants spécifiques en lieu et place des composants standards jusqu'à celles beaucoup plus proches des styles FMX (FireMonkey).

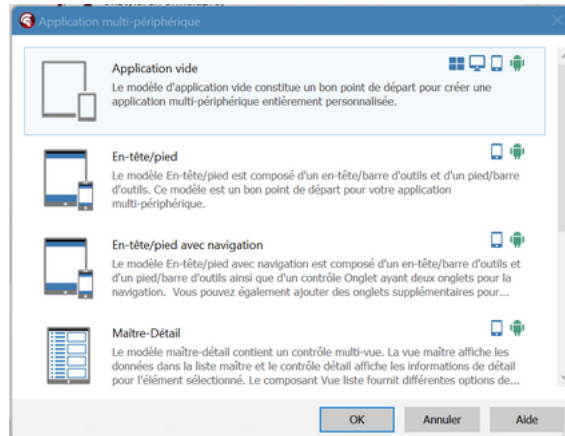


Cette « session » n'abordera que l'OS cible Windows. J'envisage d'écrire un autre tutoriel expliquant comment faire pour les autres cibles.

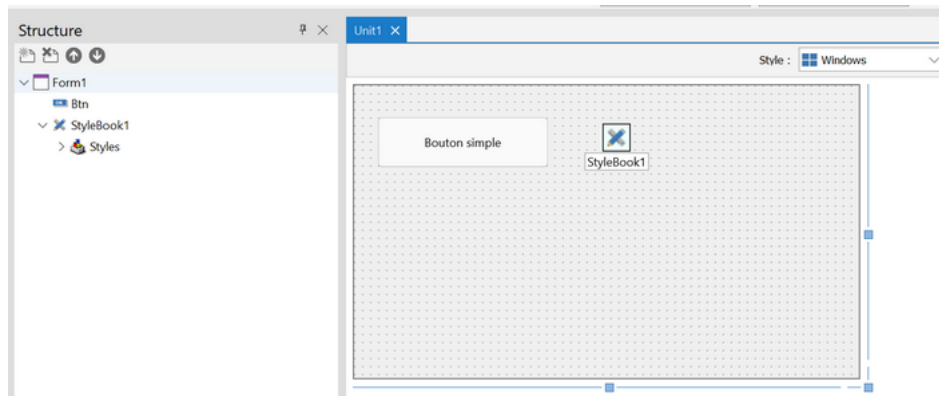
I-A - Créer une application FMX

Démarrez votre Delphi et sélectionnez l'option *Fichier/Nouveau/Application multi-périphérique*.

Plusieurs modèles vous seront alors proposés, mais pour les besoins de la démonstration une application vide servira amplement.



Une première fenêtre vide est alors créée.



Ne vous y trompez pas, même si vous n'avez pas encore défini de style, un style par défaut est déjà à l'œuvre !
Pour les curieux, je vous présenterai une manipulation dans le chapitre **III.A** qui le prouvera.

Déposez-y un simple bouton (**TButton**) et un composant **TStyleBook**.

Pas flagrant ! Oui, c'est logique, car aucune indication n'a encore été fournie pour le style.

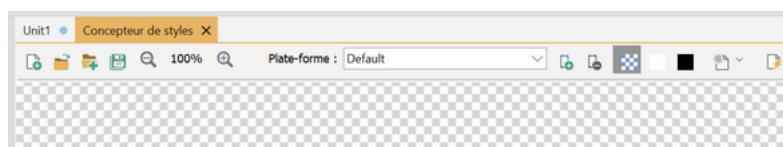
De surcroît, il y a plusieurs manières de procéder.

Je commencerai par la plus simple consistant à double-cliquer sur le composant **StyleBook1**.



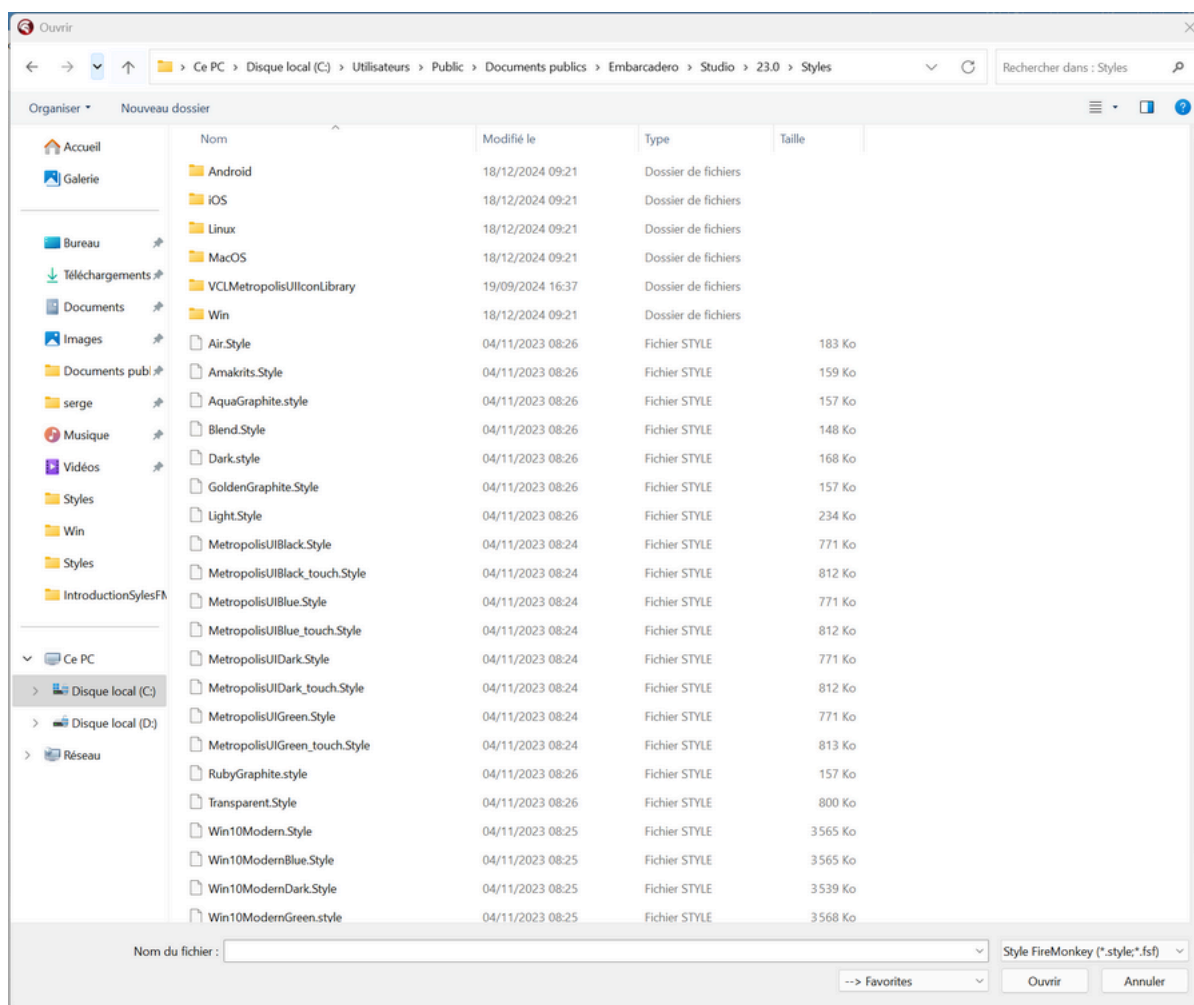
Un clic droit affichera un menu contextuel avec en première option :
« Modifier... ».

Quelles que soient ces deux premières actions, l'interface de développement (IDE) va ouvrir un nouvel onglet contenant un concepteur de styles.



concepteur de styles

Cliquez sur le bouton de chargement (le second en partant de la gauche) et sélectionnez un fichier.



Embarcadero fournit plusieurs fichiers de style, généralement d'extension « .style » ou « .fsf ». En cas d'installation standard, vous les retrouverez dans le répertoire : `C:\Users\Public\Documents\Embarcadero\Studio\23.0\Styles`

Mais aussi, pour les styles redistribuables (1), dans le répertoire : `C:\Program Files (x86)\Embarcadero\Studio\23.0\RIDEst\styles\Fmx`

Une fois le choix effectué, et le fichier chargé, fermez l'onglet du concepteur de styles.

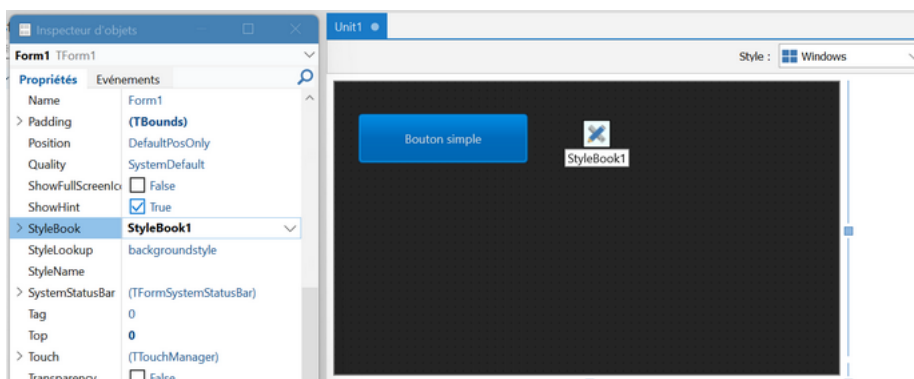
Un message de confirmation vous demandera si vous désirez bien appliquer les modifications apportées. Bien entendu, répondez « Oui ».

Surprise ! L'écran est toujours identique, une manipulation oubliée ?



En quelque sorte, il faudra indiquer dans la propriété **stylebook** de la fenêtre que le composant **StyleBook1** est utilisé.

Vous obtiendrez alors un affichage avec application du style.

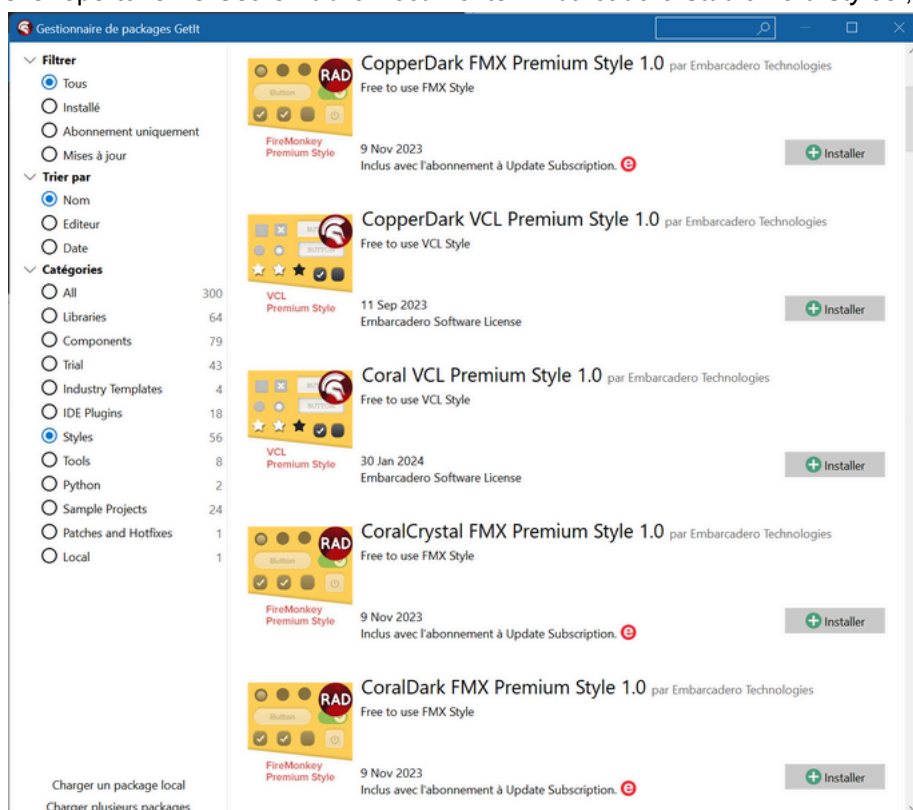


Attention, ne choisissez que des styles communs ou spécifiques à Windows.
De même, évitez de piocher vos fichiers dans une version précédente de Delphi, il y a parfois certaines incompatibilités.

I-B - Plus de styles

Si l'offre ne vous semble pas suffisante, faites un tour dans Getit :

- soit en utilisant le menu de Delphi : *Outils/Gestionnaire de packages Getit...*, ces styles téléchargés, vous les retrouverez dans le répertoire : `C:\Users\Public\Documents\Embarcadero\Studio\23.0\Styles` ;





- soit en allant sur le site [Embarcadero Getit](https://delphistyles.com/fmx/index.html) ;
- mais vous pouvez aussi en obtenir à cette adresse <https://delphistyles.com/fmx/index.html>.

I-C - Comment choisir un style ?

Chacun ses goûts et ses couleurs, toutefois, visualiser les styles disponibles sur votre ordinateur via le concepteur de styles de l'IDE serait très fastidieux.

Un outil, la visionneuse de style FMX (FireMonkey), est fourni. Situé dans le répertoire **bin** de votre installation de Delphi, il s'agit du programme « **FMXStyleViewer.exe** ».

 Pour une installation standard, il s'agit du répertoire :
C:\Program Files (x86)\Embarcadero\Studio\23.0\bin

 Utilisez le menu *Outils/Configurer les outils...* pour pouvoir démarrer ce programme rapidement (sans avoir à le chercher) directement à partir de Delphi.


I-D - Mais où se trouve ce style dans mon programme ?

Bonne question ! Il faut avoir la curiosité d'ouvrir la fenêtre en mode texte (clic droit, option « *voir comme texte* ») pour obtenir une réponse.

TForm en mode texte

```
object Form1: TForm1
  Left = 0
  Top = 0
  Caption = 'Form1'
  ClientHeight = 292
  ClientWidth = 504
  StyleBook = StyleBook1
  FormFactor.Width = 320
  FormFactor.Height = 480
  FormFactor.Devices = [Desktop]
  DesignerMasterStyle = 0
object Btn: TButton
  Position.X = 24.0000000000000000
  Position.Y = 32.0000000000000000
  Size.Width = 169.0000000000000000
  Size.Height = 49.0000000000000000
  Size.PlatformDefault = False
  TabOrder = 0
  Text = 'Bouton simple'
  TextSettings.Trimming = None
end
object StyleBook1: TStyleBook
  Styles = <
    item
      ResourcesBin = {
        464D585F5354594C4520322E3501060B7377697463687374796C65039101060D
        7468756D626261727374796C6503B70306147363726F6C6C6261727674726163
        6B7374796C6503800106147363726F6C6C62617268747261636B7374796C6503
        800106137363726F6C6C6261726C656674627574746F6E03190406147363726F
        6C6C6261727269676874627574746F6E030E0406127363726F6C6C626172746F
        70627574746F6E03180406157363726F6C6C626172626F74746F6D627574746F
        6E030F04060B687468756D627374796C6503B503060B767468756D627374796C
        6503B503060A747261636B7374796C65037B01060D747261636B626172737479
```

Évidemment, c'est loin d'être flagrant, car **ResourcesBin** est loin d'être explicite puisque le fichier semble avoir subi des transformations lors de l'application du style choisi.

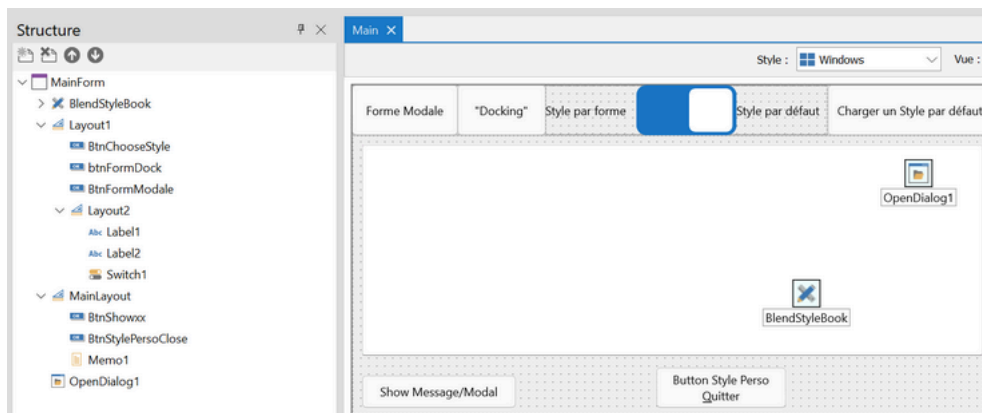
 Si vous êtes curieux, utiliser un éditeur de texte pour ouvrir un fichier « .style ». Vous y retrouverez, une structure assez proche d'un **TForm**, et surtout facile à lire.

II - La bonne utilisation des styles

Lors de l'introduction, je me suis contenté d'une application ne contenant qu'une seule fiche et d'un style chargé au moment du design. Mon objectif est quand même de fournir quelques pistes et bonnes pratiques dans le cas d'applications plus concrètes.

Pour ce faire, je vous propose de créer une application constituée d'un module de données et deux fiches. Chacune de ces unités va contenir un **TStyleBook** (oui, même le module de données !).

La fiche principale dont voici le design...



... nous permettra de tester différentes options :

- l'utilisation d'un style par fenêtre, un différent pour chaque ;
- l'utilisation d'un style « centralisé », celui du module de données ;
- le chargement d'un fichier de style à l'exécution.

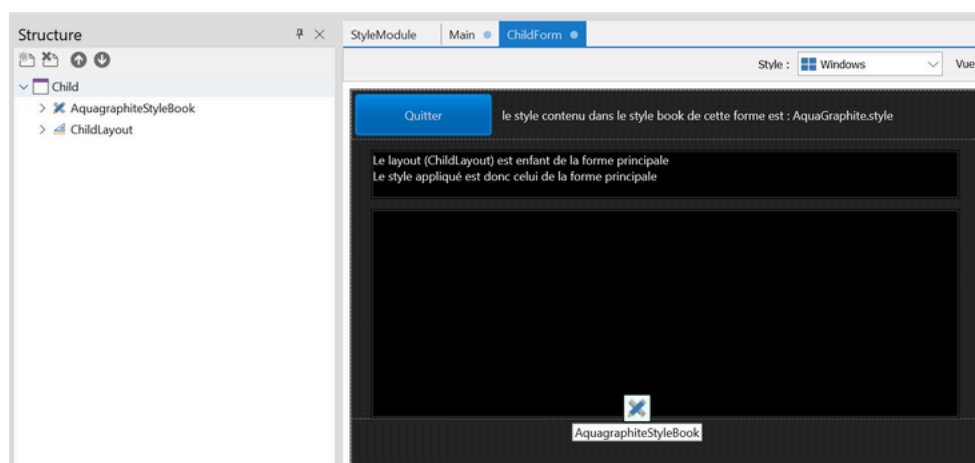
Pour la fenêtre secondaire, deux techniques différentes seront appliquées pour l'utiliser :

- un appel via un **ShowModal** ;

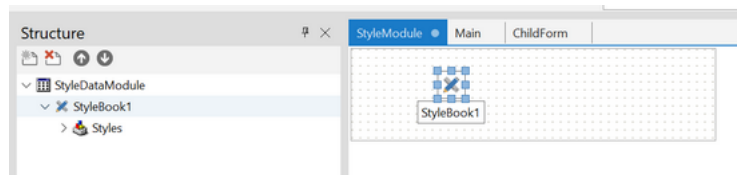


Pour rappel, une application Android ne permet pas des fenêtres modales. L'objectif n'est, pour l'instant, qu'une application « desktop » et plus particulièrement Windows.

- une technique que je nomme « docking », que vous pourrez retrouver in extenso dans mon [tutoriel sur les techniques d'enchaînement de formes](#).



Enfin, le module de données est extrêmement simple puisque ne contenant qu'un seul composant : un **TStyleBook**.



Tout néophyte à la programmation FMX (FireMonkey) et l'utilisation de style(s) se retrouvera face à de petites frustrations lors de cette réalisation, voir le chapitre [III](#).



Mes habitudes (10 ans d'applications Firemonkey à ce jour) me font éviter des « pièges » sans en avoir conscience.

II-A - Codes et propriétés des diverses unités

Vous retrouverez les sources du programme dans cette archive [Télécharger](#).

Si vous préférez tenter de reproduire pas à pas ma démarche, il m'est vite apparu que la lecture du tutoriel en serait pénalisée, car ajouter le texte de chacune des fiches pour obtenir le détail des contrôles et leurs propriétés augmenterait drastiquement le nombre de pages.

J'ai donc choisi d'indiquer dans un cadre les unités de mon programme dans lesquelles vous puiserez les informations nécessaires. Le cadre se présente comme ci-dessous :



Si vous optez pour une expérimentation pas à pas, inspirez-vous de l'unité **xxxxxxx.pas** contenant la fenêtre **Txxxxxx**.

II-A-1 - Le projet

Le projet contient trois unités, deux sont des fenêtres standards et la dernière un module de données.

En mode pas-à-pas, vous ajouterez donc, via le menu de l'EDI, une nouvelle fiche multipériphérique et un module de données.

Sauvegardez ensuite le tout :



- le projet, que j'ai renommé **UnStyleParForme** ;
- la fiche principale, j'ai remplacé le nom par défaut Form1 par **Main** et sauvegardé sous **Main.pas** ;
- la fiche secondaire, pour celle-ci j'ai donné le nom de **ChildForm** dans l'unité **Childform.pas**.

Deux points remarquables :

L'unité « **ChildForm** » est déclarée, mais je ne veux pas que celle-ci soit créée au démarrage de l'application, préférant le faire au besoin lors de l'exécution du programme.

Le module de données (unité « **StyleModule** ») sera créé avant la fiche principale (unité « **Main** »).

Pour cela deux solutions :

Ouvrez le source du projet (menu **Projet/Voir le source**) et modifiez ainsi :

Projet

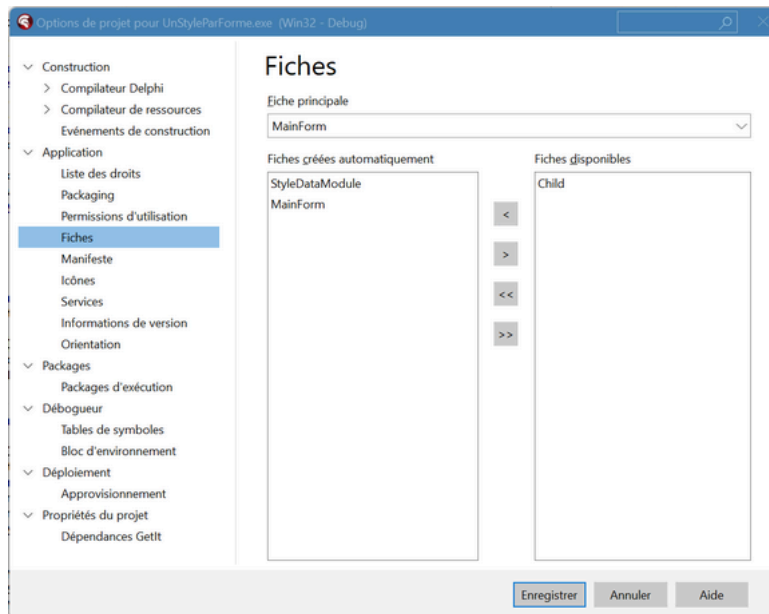
```
program UnStyleParForme;

uses
  System.StartupCopy,
  FMX.Forms,
  Main in 'Main.pas' {MainForm},
  ChildForm in 'ChildForm.pas' {Child},
  StyleModule in 'StyleModule.pas' {StyleDataModule: TDataModule};

{$R *.res}

begin
  ReportMemoryLeaksOnShutdown:=true;
  Application.Initialize;
  Application.CreateForm(TStyleDataModule, StyleDataModule);
  Application.CreateForm(TMainForm, MainForm);
  Application.Run;
end.
```

Ou, utilisez l'option de menu **Projet/Options** (Ctrl+Maj+F11), et modifiez l'ordre des fiches ainsi :



II-A-2 - Le module de données



Si vous optez pour une expérimentation pas à pas, inspirez-vous de l'unité **StyleModule.pas** contenant le module **TStyleDataModule**.

La seule propriété qu'il va falloir changer pour le composant **TStyleBook** est la propriété **UsedStyleManager** qui sera mise à **True**.

Un seul **TStyleBook** pourra avoir cette propriété à **True**.




Toute autre tentative que ce soit au design ou par code ne lèvera aucune erreur, mais la clôture du programme sera « catastrophique » : fuites de mémoire.

Pourquoi alors, après cet avertissement, indiquer cette propriété ?

De mon point de vue, c'est le plus pratique dès qu'un programme contient plusieurs fiches. Pas besoin d'indiquer la propriété **StyleName** de chaque fiche, avec le risque d'oublis possibles, toutes les fiches utiliseront le style stipulé ainsi. C'est d'ailleurs ce que le programme tente de démontrer.

Secundo, cela permet d'indiquer de ne pas utiliser les styles par défaut ce qui sera le cas pour toute fiche, dont la propriété **StyleName** ne serait pas renseignée.


Au design, je charge, via le concepteur de styles (cf. [Introduction I.A](#)), le fichier **GoldenGraphite.Style**.

 N'oubliez pas de fermer le concepteur, pour sauvegarder cette modification.

Dans ce module de données, je vais définir une fonction qui permettra, quand elles existent, d'obtenir des informations telles que :

- l'auteur ;
- son adresse mail ;
- titre ;
- version ;
- plateformes cibles ;
- indication pour savoir s'il s'agit d'un style pour plateforme mobile.

Cette fonction retournera les informations dans un tableau de chaînes caractères qui sera utilisé pour remplir les mémos des fenêtres.

 Il est regrettable que les styles redistribuables ne contiennent plus ces informations.


```
uses
  System.SysUtils,
  System.Classes,
  System.Generics.Collections,
  FMX.Types,
  FMX.Controls,
  FMX.Styles,
  FMX.Objects;

...

// ajout des commentaires de documentation XML voir code suivant
{ Déclarations publiques }
function GetStyleDescription(const StyleUsed: TStyleBook = nil)
  : TArray<String>;
```

Commentaires de documentation

```
/// <summary> Récupère des informations sur le style indiqué
/// </summary>
/// <param name="StyleUsed">le TStyleBookitem à étudier
/// </param>
/// <remarks>
/// Si le paramètre "StyleUsed" est null, alors c'est le style défini
/// comme par défaut (UseStyleManager=True) qui est étudié
/// </remarks>
/// <returns>Retourne un ensemble de chaînes
/// Si aucune description ne se trouve dans le style chargé, la chaîne "'Style sans
description'" est envoyée.
/// </returns>
```

 Une fois la déclaration de la fonction écrite, utiliser Ctrl+C pour créer la fonction dans la section implémentation.

Code de la fonction

```
{ TStyleDataModule }

function TStyleDataModule.GetStyleDescription(const styleUsed
: TStyleBook = nil): TArray<String>;
var
  AStyle: TFMXObject;
  ASd: TStyleDescription;
  ResultList: TList<String>;
begin
  ResultList := TList<String>.Create;
  try
    // Obtenir le style en cours
    if styleUsed = nil then
      AStyle := StyleBook1
    else
      AStyle := styleUsed;

    // Obtenir des informations de description
    ASd := TStyleManager.FindStyleDescriptor(AStyle);
    if Assigned(ASd) then
    begin
      ResultList.Add('Auteur : ' + ASd.Author);
      ResultList.Add('Titre : ' + ASd.Title);
      ResultList.Add('Version : ' + ASd.Version);
      // également disponible AuthorEmail, PlatformTarget, MobilePlatform
    end
    else
    begin
      if styleUsed = nil then
        ResultList.Add('Stylemanager utilisé');
      ResultList.Add('Style sans description');
    end;
  ASd.Free;
  // Lister les ressources utiles
  ResultList.Add('Applicable aux OS suivants');
  TStyleManager.EnumStyleResources(
    procedure(const AResourceName: string; const APlatform: TOSPlatform)
    begin
      ResultList.Add(AResourceName);
    end);
  Result := ResultList.ToArray;
finally
  ResultList.Free
end;
end;
```



Pourquoi dans le module de données ? Pour éviter de dupliquer le code dans les deux autres unités.

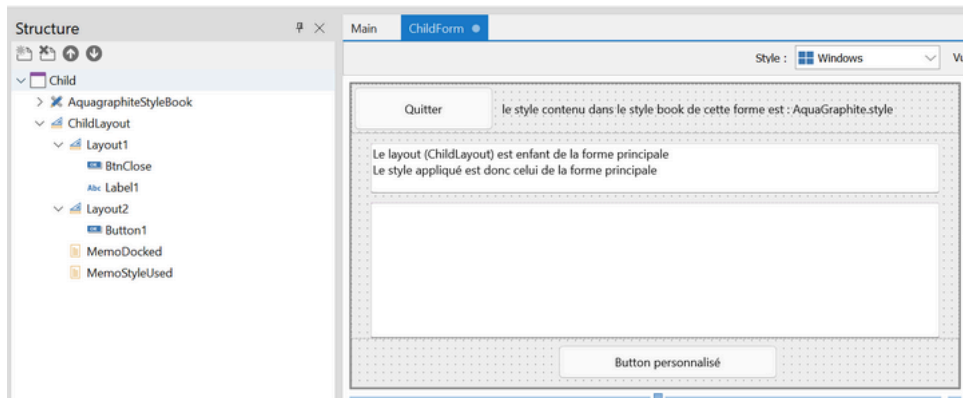
II-A-3 - La fenêtre secondaire



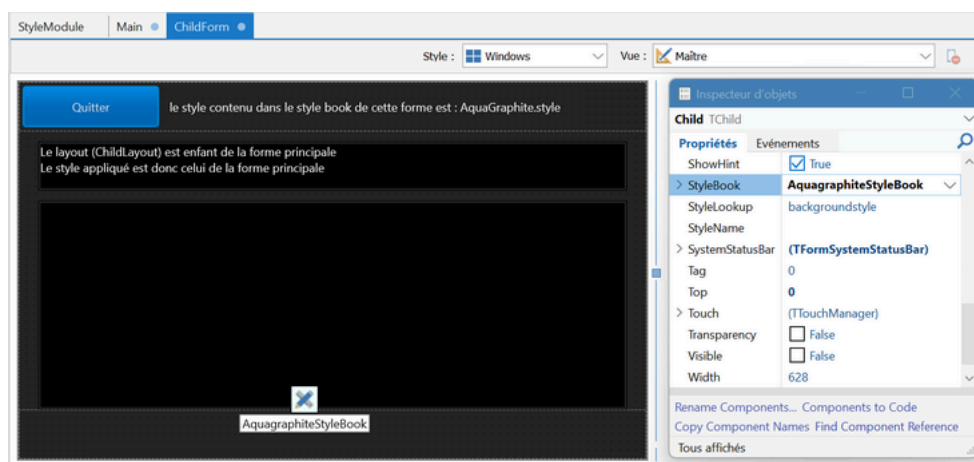
Si vous optez pour une expérimentation pas à pas, inspirez-vous de l'unité **Childform.pas** contenant la fiche **TChild**.

Avant de passer à l'unité principale, je dois d'abord montrer ce que propose la fenêtre secondaire, car celle-ci va être utilisée.

Quoique d'un design un peu plus complexe, car elle nécessite un **TLayout** (que je nomme **ChildLayout**, avec la propriété **Align=Client**) pour la démonstration de la technique d'ancrage, cela reste une forme très classique, un bouton, quelques mémos (tous enfants de **ChildLayout**) et d'un **TStyleBook**.



Une fois le style chargé (fichier **aquagraphite.style**) et appliqué (en indiquant la propriété **StyleBook** de la fenêtre), on obtiendra ceci :



Pour le code quelques recommandations :

- la codification de l'évènement de la fiche **OnClose** qui permettra de libérer la fenêtre ;
- la nécessité de coder l'action **OnClick** du bouton.

```
procedure TChild.BtnCloseClick(Sender: TObject);
begin
  Close;
end;

procedure TChild.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  if docked then
    ChildLayout.parent := self;
  // libère la forme à la clôture
  Action := TCloseAction.caFree;
end;
```

Vous remarquerez l'utilisation d'une variable **docked**, booléenne, introduite dans la section **public**.

```
public
  public
  { Déclarations publiques }
  docked: Boolean;
```

Pas vraiment nécessaire, j'aurais pu utiliser la visibilité du premier mémo (nommé **memodocked**), mais j'y vois une facilité de lecture de code.



Un puriste aurait même déclaré, non pas une variable, mais une propriété.
Pour cet exemple, que je vise simple (le moins de code possible), j'en resterai à la variable.

II-A-4 - Unité principale



Si vous optez pour une expérimentation pas à pas, inspirez-vous de l'unité **Main.pas** contenant la fiche **TMainForm**.

Pour le **TStyleBook**, nommé « **BlendStyleBook** », j'ai chargé le fichier de style **blend.style**.

Le design précédent montre que la propriété **Stylebook** de la fenêtre est égale à « **BlendStyleBook** ». Par la suite, à l'exécution, j'effacerai celle-ci pour obtenir, en premier lieu, le style défini par le module de données (voir chapitre **II.A.4.b**).



Quelques explications supplémentaires dans le « coin du mécano », chapitre **III.B.1**

J'ai, plus ou moins sciemment, utilisé une pseudobarre de boutons pour tester les différents cas à tester.

II-A-4-a - Charger un fichier de style (extensions .style ou .fsf)

Le bouton le plus à droite va permettre de charger un fichier de style à l'exécution.

Une fois le style chargé, la présentation des fenêtres change, et les informations sur le style dans le **TMemo** sont mises à jour. Ces informations seront obtenues grâce à la fonction décrite au chapitre **II.A.2**

Cette fonctionnalité ne sera possible que si le commutateur (à sa gauche) est positionné sur l'utilisation du style défini par défaut, c'est-à-dire celui pris en charge par le **TStyleManager**.



TStyleManager n'est pas un composant visuel, ne le cherchez pas.
Pour y accéder, vous devrez indiquer l'utilisation de l'unité **FMX.Styles**.

RAPPEL : Un seul TStyleBook dans toutes les unités de l'application doit avoir cette valeur à True.

Chargement d'un fichier

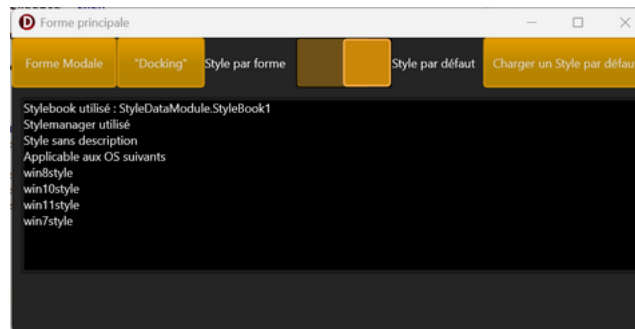
```
procedure TMainForm.BtnChooseStyleClick(Sender: TObject);
var r : boolean;
begin
  if Switch1.IsChecked then // Utilisation du TStyleManager
  begin
    if opendialog1.Execute then
    begin
      Memo1.Lines.Clear; // efface les informations
      try
        // La fonction SetStyleFromFile doit retourner True
        // sauf si le fichier n'est pas interprétable.
        r := TStyleManager.SetStyleFromFile(OpenDialog1.FileName);
      except
        // un fichier style ne correspondant pas à la plateforme
        // lèvera une exception.
        r:=false;
      end;
      if not R then
        // Style incompatible, pas de changement sur les fenêtres
        Memo1.Lines.Add(OpenDialog1.filename+' incompatible')
      else begin
        // informations
```

Chargement d'un fichier

```
Mem1.Lines.Add('Style chargé : '+Opendialog1.FileName);
Mem1.Lines.Add('Stylebook utilisé : StyleDataModule.'+StyleDataModule.StyleBook1.Name);
// Fait appel à la fonction définie dans le module de données
// voir chapitre II.a.2
mem1.Lines.AddStrings(StyleDataModule.GetStyleDescription(StyleBook));
end;
end;
end;
end;
```

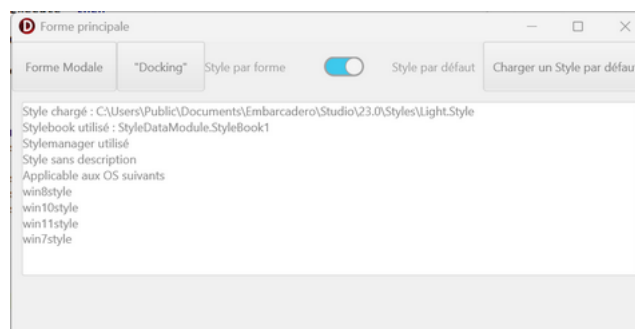
À l'exécution vous obtiendrez cet enchaînement :

Ouverture du programme :

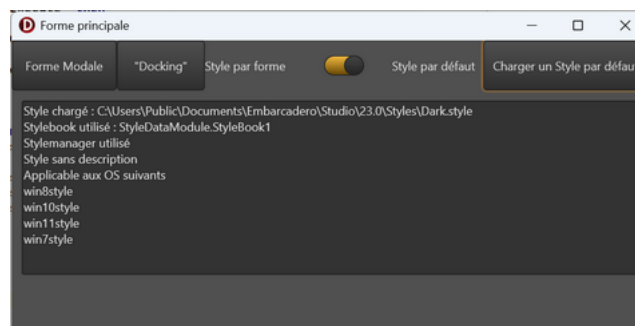


Démarrage de l'application

Choix d'un style clair dans le dialogue de choix de fichier (**light.style**) :



Choix d'un style foncé, dans le dialogue de choix de fichier (**dark.style**) :




Super ! Avec ce code, vous avez déjà une première approche des interfaces utilisateurs dites « modernes », à savoir permettant de basculer d'un thème sombre à un thème clair.

Bien évidemment, il vous faudra, si vous déployez votre application, fournir les fichiers de style. Or, je vous ai déjà averti des problèmes de licences.

Déployer des fichiers sous forme binaire (.fsf) est, peut-être, autorisé.
Pour Windows, vous n'en avez pas de fourni !



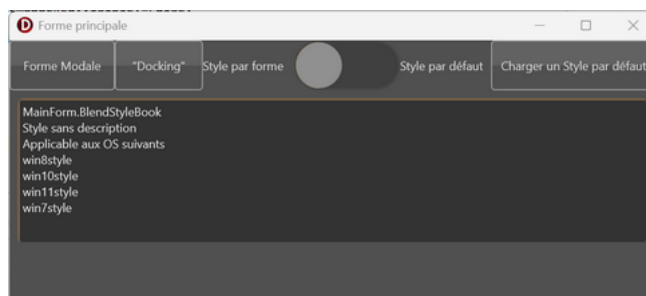
L'astuce, sauvegarder le style en utilisant le bouton  dans l'onglet « concepteurs de styles » lors du design de l'application.

II-A-4-b - Le commutateur pour utiliser un style par fenêtre

L'astuce consiste à jouer avec la propriété **StyleBook** de la fiche, une valeur nulle forcera le programme à utiliser le style défini via le **TStyleManager**.

```
procedure TMainForm.Switch1Switch(Sender: TObject);
begin
    mem01.lines.Clear;
    if Switch1.IsChecked then
    begin
        Mem01.lines.Add('Stylebook utilisé : StyleDataModule.'+StyleDataModule.StyleBook1.Name);
        StyleBook:=nil;
    end
    else begin
        Mem01.lines.Add('MainForm.'+BlendStyleBook.Name);
        StyleBook:=BlendStyleBook;
    end;
    // obtenir des informations sur le style
    Mem01.Lines.AddStrings(StyleDataModule.GetStyleDescription(StyleBook));
end;
```

Si le commutateur est en position éteinte, on obtient :



II-A-4-c - L'ouverture d'une fenêtre modale

Une application contient souvent plus d'une fenêtre. Un des premiers « réflexes » alors est de créer la fiche et de l'afficher via l'instruction **Show** ou **ShowModal**. Parti pris de ma part, je préfère la fonction **ShowModal**, pour de plus amples explications, je vous renvoie une fois de plus dans le « coin du mécano », chapitre III.

Pour créer la fiche enfant, rien de plus simple, et peu de code.

fenêtre Modale

```
procedure TMainForm.BtnFormModaleClick(Sender: TObject);
var
    M: TChild;
begin
    M := TChild.Create(Self);
    try
        M.ShowModal;
    finally
    end;
```


fenêtre Modale

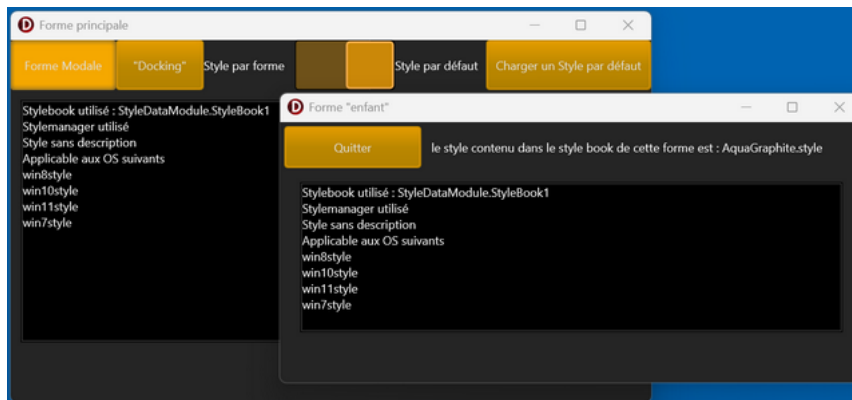
```
// M est libéré via son évènement OnClose
end;
end;
```

Évidemment, cette procédure va être modifiée pour prendre en compte le choix indiqué par le commutateur d'utilisation du style, qui sera soit celui contenu dans le **TStyleManager** soit celui de la fiche créée. Et enfin, pour afficher dans le mémo les informations du style sélectionné :

Modifications (bloc try finally)

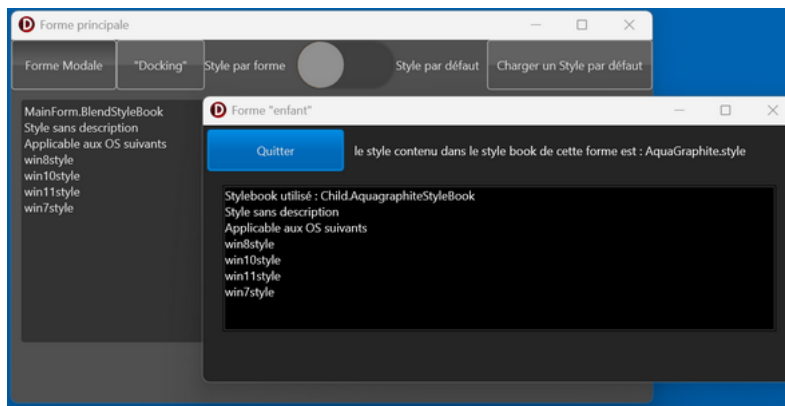
```
if Switch1.IsChecked then
begin
// force l'utilisation du style contenu dans le module de données
M.StyleBook := nil;
M.MemoStyleUsed.Lines.Add('Stylebook utilisé : StyleDataModule.' +
StyleDataModule.StyleBook1.Name);
end
else
begin
// utilisation du style de la fenêtre
M.StyleBook := M.AquagraphiteStyleBook;
M.MemoStyleUsed.Lines.Add('Stylebook utilisé : Child.' +
M.StyleBook.Name);
end;
M.MemoDocked.Visible := false;
M.Docked:=false;
// Récupération d'informations sur le style utilisé
M.MemoStyleUsed.Lines.AddStrings (StyleDataModule.GetStyleDescription
(M.StyleBook));
M.ShowModal;
```

À l'exécution, selon la position du commutateur, nous obtiendrons deux résultats possibles :



Style par défaut

Ou



Style de la forme

II-A-4-d - La technique d'ancrage d'une fenêtre

La technique d'ancrage (**Rappel du tutoriel**) donnera un résultat différent.

Pour la réalisation de cette technique, l'ensemble des éléments de la fiche secondaire est contenu dans un **TLayout**.

C'est un des grands avantages de FMX (FireMonkey) par rapport à VCL, tout composant peut être enfant d'un autre composant.



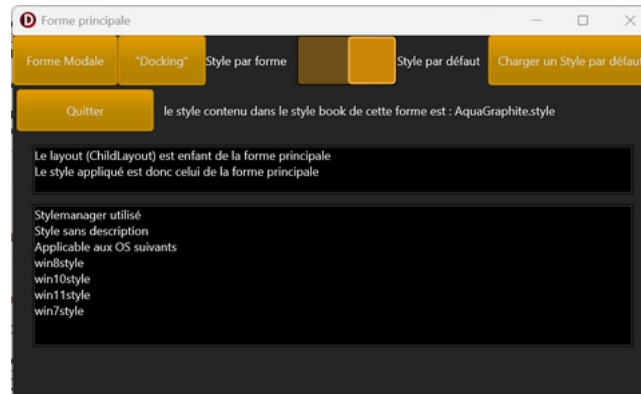
Pour ceux qui en sont restés à la VCL, un **TLayout** est l'équivalent d'un **TPanel**, transparent et sans bordure.

Il vous faudra un certain temps d'adaptation pour éviter le « réflexe » **TPanel** et le remplacer par un **TLayout** (ou un **TRectangle** si vous avez besoin de bordures).

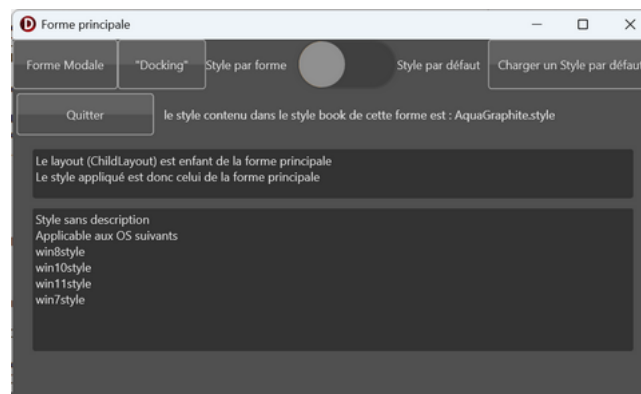
Ancrage de fenêtre

```
procedure TMainForm.btnFormDockClick(Sender: TObject);
var
  M: TChild;
begin
  Memo1.Visible := false;
  M := TChild.Create(Self);
  try
    M.MemoDocked.Visible := True;
    M.ChildLayout.Parent := MainLayout;
    M.Docked := True;
    if Switch1.IsChecked then
      M.StyleBook := nil
    else
      M.StyleBook := M.AquaGraphiteStyleBook;
  // informations M.MemoStyleUsed.Lines.AddStrings (StyleDataModule.GetStyleDescription
    (M.StyleBook));
  finally
    ///
  end;
end;
```

À l'exécution, le résultat est différent, l'affichage de la fiche utilisera le style de la fiche principale, quelle que soit l'option du commutateur.



Ou, si le choix est d'un style par fenêtre :



Il existe d'autres techniques :



- définir un cadre plutôt qu'une fenêtre ;
- utiliser des composants tiers comme **TFrameStand** (téléchargeable via **GetIt** ou directement [ici](#)).

III - Le coin du mécano

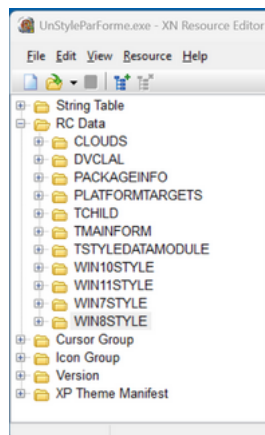
Dans ce paragraphe, ce qui peut coincer [III.B.1](#), ce qui peut être frustrant [III.B.2](#), mais aussi une explication de mon parti pris concernant la méthode d'affichage de la fiche secondaire [III.C](#) et, en guise de mise en bouche, les possibilités de personnalisation.

III-A - « Je n'utilise pas les styles »

C'est une remarque qui revient souvent dans le forum Delphi. De mon point de vue cela revient à dire : « L'air est invisible, donc n'existe pas ».

Si vous êtes vraiment poussé par la curiosité, utilisez un outil d'éditeur de ressource (par exemple : [XN Resource Editor](#)).

Après installation de cet utilitaire, ouvrez l'exécutable que vous avez déjà dû recréer (*File/Open*), Ouvrez la branche *RC Data* et vous y découvrirez ceci :

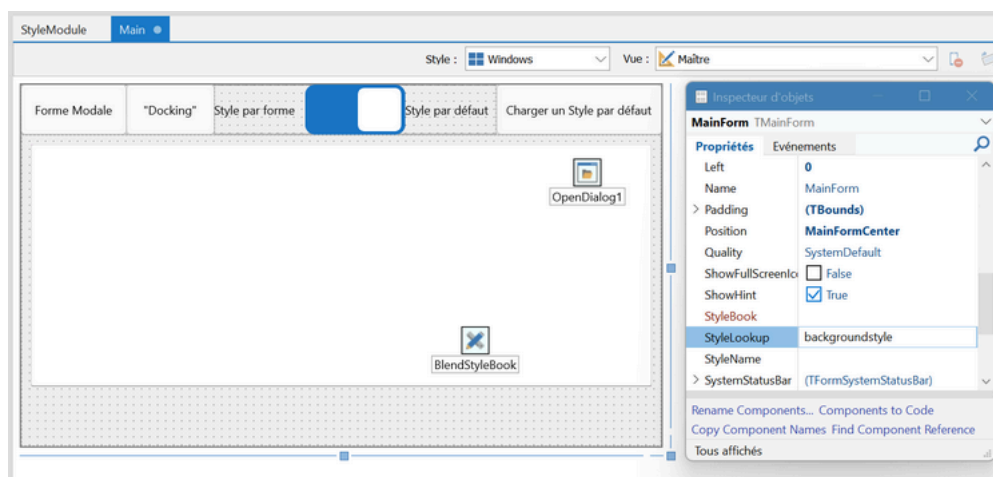


III-B - Mises en garde

III-B-1 - Au design

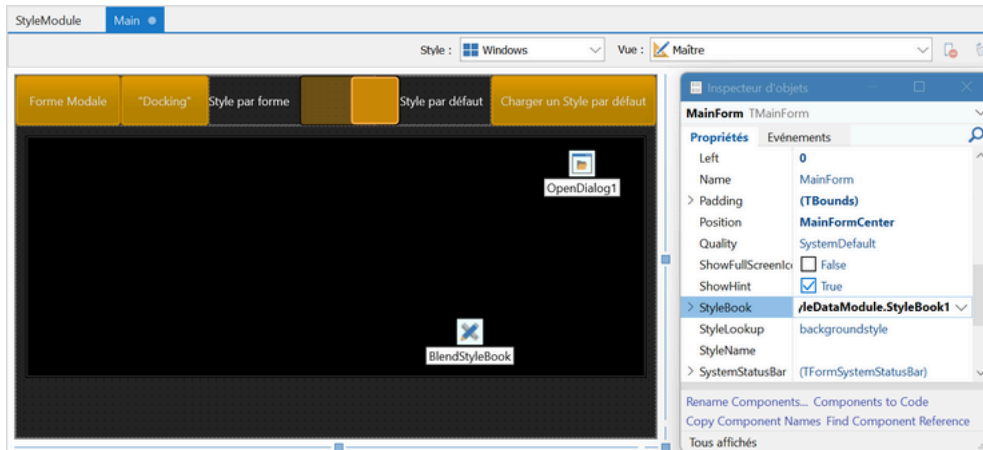
L'affichage dans l'IDE n'est pas toujours aussi WYSIWYG (ce que je vois est ce que je vais obtenir) d'où la manipulation que je vous ai proposée : renseigner la propriété **StyleBook** de la fiche. Cette impression de « *ça fonctionne pas* » se ressent surtout quand on utilise le **TStyleManager** (ce que pourtant je recommande vivement dans le cas d'applications contenant plusieurs fiches ou à cibles multiples).

Prenons, même le projet déjà décrit au chapitre II, si vous n'avez rien indiqué pour la propriété **StyleBook** de la fiche, vous obtiendrez un affichage non stylé, ou plus exactement avec le style contenu dans les sources des composants, et ce même si l'unité du module de données est ouverte dans l'IDE.



Cela ne pénalise en rien l'exécution, mais c'est un peu énervant.

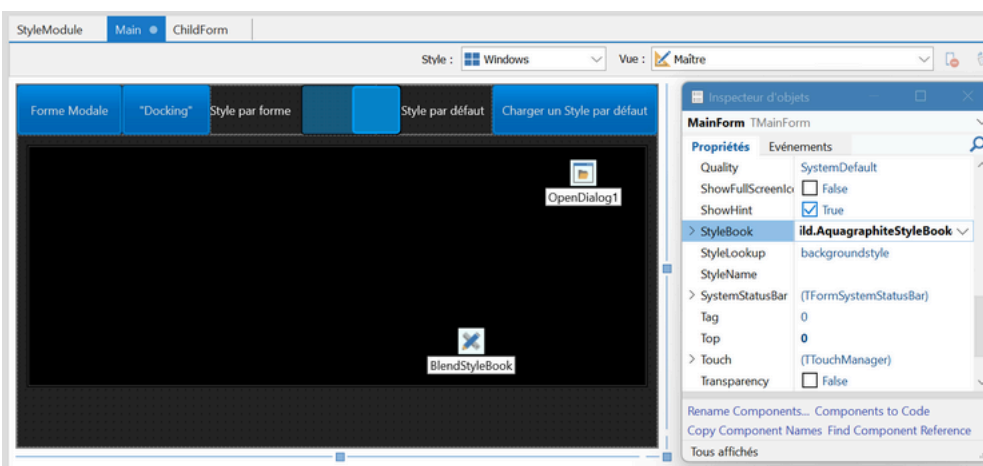
Ma solution est donc d'indiquer la propriété **StyleBook** de la fiche.



Mais, c'est là où cela devient « scabreux », nous pouvons également renseigner tout autre style, que ce soit, dans le cadre de mon tutoriel, celui de l'unité :



ou celui des autres unités pour peu que l'unité soit ouverte dans l'IDE !



Oui, il y a de quoi se perdre, surtout lorsque l'on va rouvrir le projet, si vous avez indiqué la propriété **StyleBook** et, même si l'unité contenant le style indiqué n'est pas ouverte, vous vous retrouverez avec l'affichage défini à la clôture du projet (étrange !).

Pour pallier ce dernier point, vous pouvez toujours modifier les options de l'IDE (menu : « **Outils/Options** », branche « **IDE→Enregistrement et récupération** ») et cocher la case « **Enregistrer le bureau du projet à la fermeture** ».

III-B-2 - ShowMessage et autres dialogues

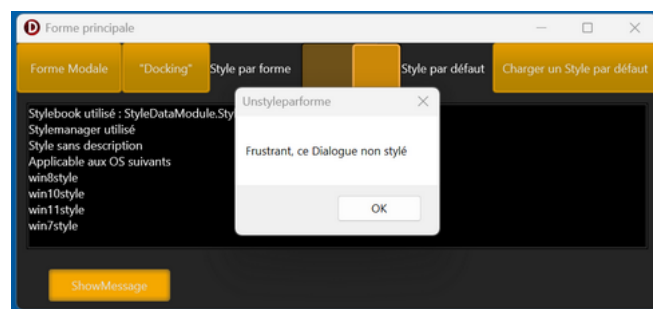
C'est, certainement, la chose la plus frustrante qui soit. Vous avez déjà peut-être pu le remarquer lors de l'utilisation du bouton permettant le choix d'un style le dialogue de sélection n'est pas « stylé ».

Il en va de même pour tout affichage de message que ce soit la simple instruction **ShowMessage**.

En démonstration, j'ai simplement ajouté un bouton (qui va d'ailleurs également nous servir pour le chapitre suivant) et codé l'évènement **OnClick (2)** de ce dernier ainsi :

ShowMessage

```
procedure TMainForm.BtnShowxxClick(Sender: TObject);
begin
  ShowMessage('Frustrant, ce Dialogue non stylé');
end;
```



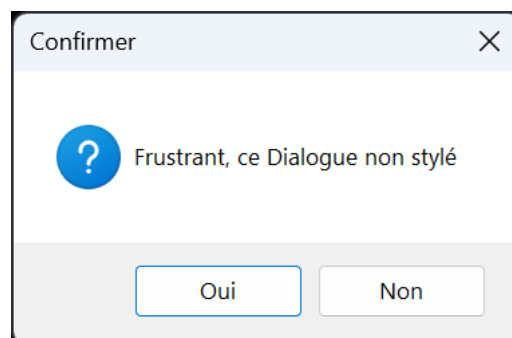
ShowMessage

L'instruction :

```
//[Avertissement] Le symbole 'MessageDlg' est déprécié
// il vaut mieux utiliser les méthodes de FMX.DialogService

MessageDlg(
  'Frustrant, ce Dialogue non stylé',
  TMsgDlgType.mtConfirmation, [TMsgDlgBtn.mbYes, TMsgDlgBtn.mbNo], 0);
```

ne fait pas mieux.



L'utilisation des services de message (via **TDialogService** ou **IFMXDialogService**), pourtant recommandé, ne fera pas mieux.
Développer ceci n'apportera rien, donc pas de démonstration inutile.

Je signalerai juste que **ShowMessage** utilise **TDialogServiceSync.ShowMessage** et affiche une fenêtre modale, ce qui me fait une introduction au chapitre suivant.

III-C - Showmodal ou Show

Fiche modale ou pas, j'ai pris le parti d'utiliser la première option, non utilisable pour les mobiles, pour n'avoir qu'une seule fiche « enfant ». Cette utilisation ressemble beaucoup à un affichage synchrone de message.

L'utilisation de fiches non modales ne change en rien mon discours sur l'utilisation des styles.

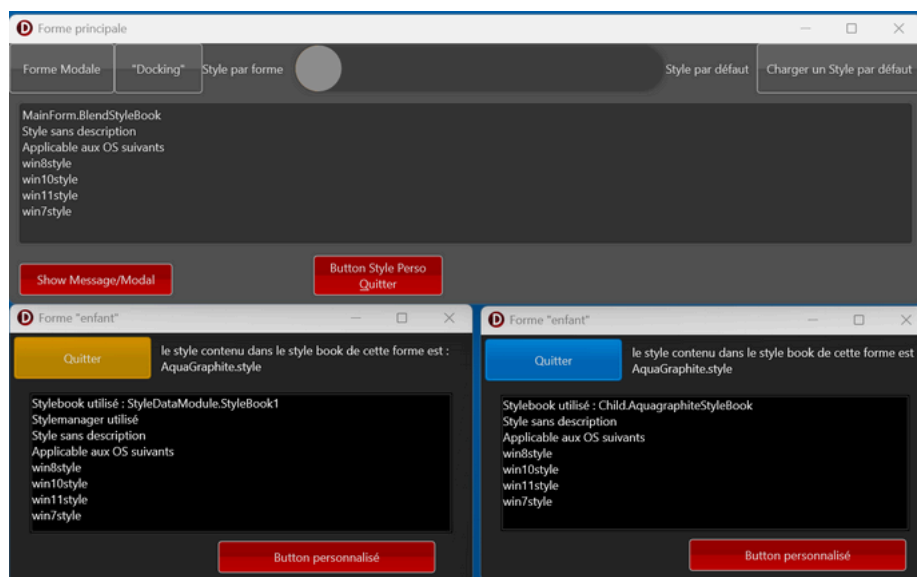
Ce code,

Fenêtres modales

```
procedure TMainForm.BtnShowxxClick(Sender: TObject);
var
  M: TChild;
begin
  M := TChild.Create(Self);
  try
    if Switch1.IsChecked then
    begin
      M.StyleBook := nil;
      M.MemoStyleUsed.Lines.Add('Stylebook utilisé : StyleDataModule.' +
        StyleDataModule.StyleBook1.Name);
    end
    else
    begin
      M.StyleBook := M.AquagraphiteStyleBook;
      M.MemoStyleUsed.Lines.Add('Stylebook utilisé : Child.' +
        M.StyleBook.Name);
    end;
    M.MemoDocked.Visible := false;

    M.MemoStyleUsed.Lines.AddStrings(StyleDataModule.GetStyleDescription
      (M.StyleBook));
    M.Show;
  finally
    //
  end;
end;
```

...permettra d'obtenir ceci :



Vous remarquerez, dans cette dernière image, des boutons de couleur rouge.

Aucun des styles du programme décrit n'a ce style, le chapitre suivant va vous permettre d'aborder une des possibilités des styles : la personnalisation.

III-D - Personnalisation

Est-ce que l'on peut mixer deux styles ? La réponse courte est non.

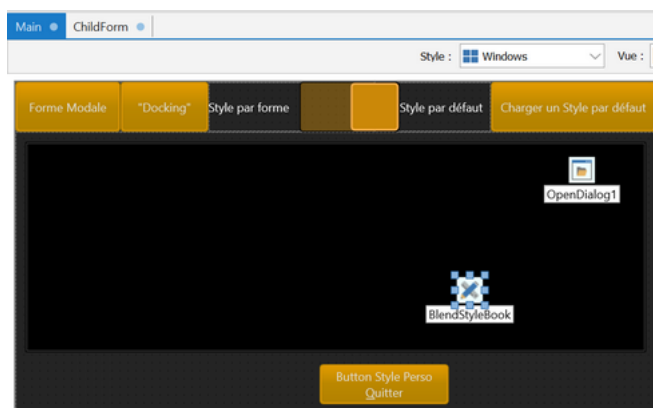
Réponse courte, veut quand même dire que la réponse est plus mitigée.



Via certaines manipulations, débordant largement du cadre du tutoriel, la réponse est plus mitigée : « Oui, si... ».

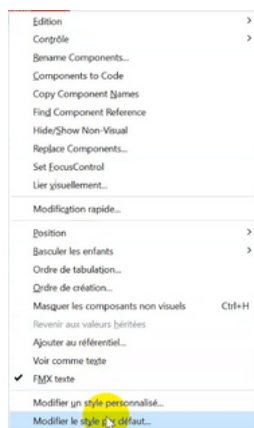
Néanmoins, pour ouvrir l'horizon, je vous propose de créer un bouton de couleur rouge, quel que soit la fenêtre, ou le style appliqué à celle-ci.

Déposez un nouveau bouton sur la fenêtre principale.



Cliquez avec le bouton droit dessus, un menu contextuel apparaît et, tout en bas deux options vous sont proposées :

- « Modifier un style personnalisé... » ;
- « Modifier le style par défaut... ».



menu contextuel

Utilisez la première option, ce qui va vous ouvrir l'onglet concepteur de styles.

Vous allez vous retrouver positionné sur un nouveau **TLayout** nommé **btnStylePersoStyle1**.



Si un fichier de style est déjà chargé, cet élément sera « noyé » au sein de l'ensemble, il faudra se repositionner dessus. On constatera alors le même comportement que l'option « Modifier le style par défaut... ».

Si aucun fichier de style n'a encore été défini, c'est-à-dire que le style défini par défaut est celui de Windows, seul l'élément à modifier apparaîtra.



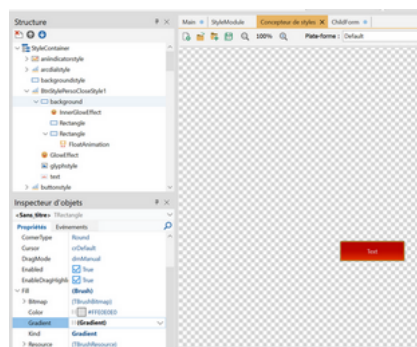
Attention, il ne s'agit pas du nom du composant (propriété **Name**, non indiquée), mais de la propriété **StyleName** qui est affichée.

Profitez-en pour la renommer, par exemple : **boutonrouge**

Modifiez ensuite le remplissage, c'est-à-dire la propriété **Fill** du **TRectangle** (**StyleName=background**).



Attention, tout dépend du style modifié. Il y a beaucoup de variations possibles, débordant largement du cadre du tutoriel.



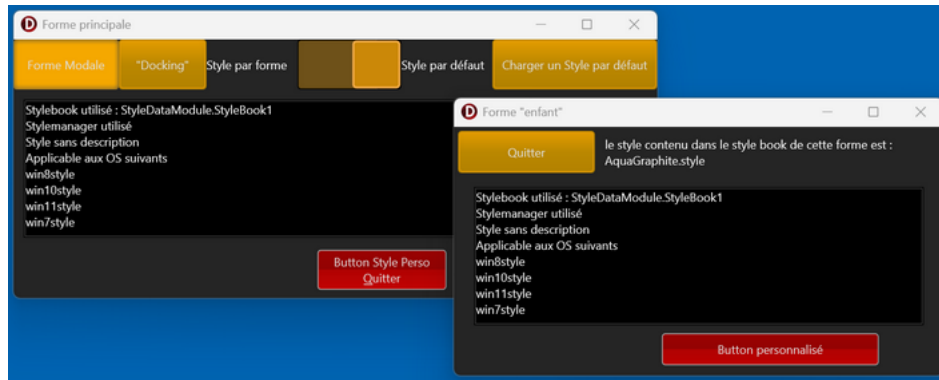
Le hic de cette technique, c'est qu'elle ne s'applique qu'à un seul **TStyleBook**, vous vous voyez modifier tous les styles que vous proposeriez (comme l'alcool, avec modération) à l'utilisateur final ? Heureusement, il y a une astuce qui va éviter ça.

- Sélectionnez l'élément de style (le **TLayout** avec le nom de style choisi) créé.
- Coupez-le. (**Ctrl+X**).
- Fermez le concepteur de styles. Pour rappel, cela sauvegarde les modifications dans le fichier **dfm**.
- Déposez un nouveau **TStyleBook** sur la fiche.
- Double-cliquez sur ce nouveau composant pour ouvrir le concepteur de styles.
- Vous avez alors uniquement un **TStyleContainer**.
- Collez l'élément de style. (**Ctrl+C**).
- Fermez le concepteur de styles.

Si vous appliquez cette astuce, vous pourrez déplacer ce nouveau **TStyleBook**, par exemple dans le module de données et utiliser ce style de bouton dans toutes les fiches.

Pour tout bouton que vous déposerez sur une fiche et que voudriez de ce style, il vous suffira de changer sa propriété **StyleLookUp** et d'y renseigner le **StyleName** défini.

Le résultat :



Ce billet de mon blog approfondit, un peu, les personnalisations.

IV - Bibliographie, visionnages intéressants

En dehors de la documentation officielle d'Embarcadero, vous pouvez trouver des ressources supplémentaires pour la personnalisation des applications FireMonkey :

- livres spécialisés ;
- forums et communautés en ligne ;
- blogs et tutoriels ;
- vidéos YouTube.

Ces ressources devraient vous fournir des informations complémentaires et des exemples pratiques pour personnaliser vos applications FireMonkey.

IV-A - Livres

Delphi 10.3 Programmation orientée objet en environnement Windows, consultez l'**extrait**.

Delphi GUI Programming with FireMonkey, excellent livre au demeurant, ouvre des horizons dans le chapitre 7 *Understanding FMX Style Concept*.

IV-B - Liens

Bien évidemment, consultez **la documentation en ligne**

et, plus particulièrement, **La personnalisation des applications FireMonkey avec les styles**.

Un peu d'historique qui permettra de voir les avancées.

Beaucoup de ressources ont disparu, mais j'ai pu retrouver quelques liens




Styles Firemonkey XE2

Le blog de Sarina Dupont contient beaucoup de ressources même si, hélas, beaucoup d'articles sont perdus.




Il est possible de retrouver (sans les images) les articles perdus dans **son flux RSS**.

Dans l'ordre chronologique :

- **How to load custom styles at runtime**
- **FireMonkey Styles, Part 1: Customizing the Style Template**

-  **FireMonkey Styles, Part 2: Creating a custom style for TSwitch**
-  **Using Custom Styles in RAD Studio 10.3**
-  **Creating a custom button style with RAD Studio 10 Seattle**
-  **Adding a style selector to your application** (une autre manière de charger des styles).

Quelques vidéos :

-  **Une playlist** des vidéos de la chaîne Youtube Embarcadero (recherchez avec les mots clés « style firemonkey ») pour affiner la recherche.
-  **Styler une application iOS.**
-  **Styler une application Android**, édifiante démonstration d'une personnalisation de style utilisant l'outil : Concepteur de styles.

V - Débriefing

Au fil de ce tutoriel, vous avez pu découvrir diverses manières d'utiliser le composant **TStyleBook** et quelques petites astuces.

Pour une application plus professionnelle, un **TStyleBook** au sein d'un **TDataModule** utilisant le **TStyleManager** est le plus adapté. Dans le cadre d'un développement multi-OS, on pourrait même créer un module par cible que des instructions de compilation conditionnelle gèreraient.

Prenez bien en compte que ce ne sont pas les seules possibilités d'utilisation, juste celles qui me paraissent les plus simples.

Vous retrouverez les sources du programme  **Télécharger**, mais osez le pas-à-pas pour découvrir ces techniques.

Notes à propos des sources de ce programme, si vous tentez de rétrograder vers une version plus ancienne de Delphi :

- modifiez les divers **TStyleBook** afin de charger des fichiers spécifiques à la version ;
- modifiez le chemin par défaut du **TOpenDialog**.

Sans ces modifications, il est possible que des erreurs soient levées.

VI - Remerciements

Je remercie les lecteurs (hors forum) de mes préversions, ainsi que **Rémi Gouyon** pour sa relecture technique et le relecteur orthographique **f-leb**, membres de **developpez.com** des soins qu'ils ont apportés pour l'obtention de la version finale.

1 : Attention, certains styles sont sous licence.

2 : Convention d'écriture : Les termes en gras et italique indiquent des composants ou des propriétés des dits composants. En gras également, les noms de fichiers (exemple : **monstyle.style**).