

Styles FMX : Introduction à la personnalisation

Comment appliquer un style différent de celui de l'application à un ou plusieurs contrôles particuliers.

Par Serge Girard

Date de publication : 31 janvier 2025

DÉBUTANT

Durée : 20 mn

Dans le précédent tutoriel d **'introduction aux styles Firemonkey** (chapitre III.D), j'avais introduit la notion de personnalisation de style . Ce tutoriel a pour objectif de proposer diverses techniques de base de la personnalisation.



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



EN COURS DE RELECTURE TECHNIQUE
CORRECTION GRAMMATICALE
INCOMPLÈTE

C'est donc pour l'instant à titre d'auteur que je partage ce document.

Commentez

I - Changer la présentation d'un contrôle.....	3
I-A - Étape 1 : ajout d'un élément de style.....	4
I-B - Étape 2 : Ajouter des composants dans le Layout.....	5
I-B-1 - Améliorations possibles.....	5
I-B-2 - Pourquoi un TLabel et pas un TText ?.....	9
I-C - Ajouter un second élément.....	10
I-C-1 - La propriété StyleName.....	11
I-D - Tester.....	11
I-D-1 - Modifier le style d'un composant à l'exécution.....	12
I-D-2 - Utilisation sur un bouton.....	13
II - Plus loin que les formes basiques.....	13
II-A - Création du nouvel élément.....	14
II-A-1 - Accéder aux propriétés de ce style.....	15
II-A-2 - Osez plus : le composant TSKSVG.....	16
III - L'utilisation des images.....	18
III-A - État des lieux.....	18
III-B - Ajouter d'autres images.....	19
III-B-1 - Utiliser l'image.....	20
III-B-2 - Résolutions des écrans.....	23
III-B-2-a - Ajout d'images pour d'autres résolutions.....	24
III-B-3 - Conclusion sur les résolutions.....	27
IV - Références.....	27
IV-A - Le concepteur de styles.....	27
IV-B - Les graphiques vectoriels évolutifs (SVG).....	27
V - Débriefing.....	28
VI - Remerciements.....	28

I - Changer la présentation d'un contrôle

Pour commencer doucement, nous allons modifier la forme d'un panneau (**TPanel**).

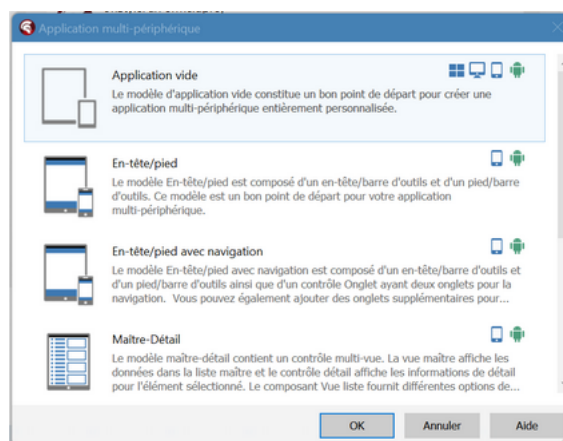
Ne vous attendez pas, ici, à quelque chose de vraiment spectaculaire.



Ce chapitre est plus une initiation qu'un style complet.

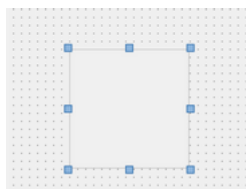
Plutôt que de poser un TPanel sur une fiche, songez qu'il existe d'autres formes (**TRectangle**, **TRoundRect**...) qui feraient tout aussi bien l'affaire.

Créez une application FMX (Firemonkey) vide. (Fichier/Nouveau/Application multi-périphérique – Delphi)



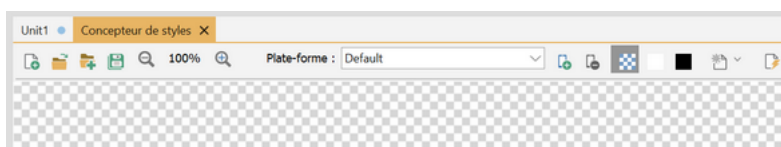
Déposez sur la fiche générée par ce dialogue, deux **TPanel**, un bouton et un **TStyleBook**.

Le bouton nous servira pour changer la présentation des composants.



Double-cliquez sur le **TStyleBook** nommé **StyleBook1**.

Un nouvel onglet s'ouvre dans l'IDE (Environnement de Développement Intégré), intitulé « concepteur de style ».

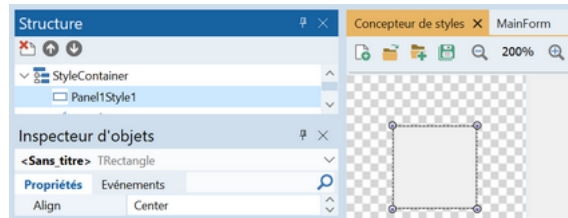


A contrario du précédent tutoriel, nous n'allons pas utiliser la barre d'outil du concepteur de styles pour charger un fichier de style, mais s'appuyer sur le style par « défaut » (dépendant de votre version de Windows).

La partie qui va nous intéresser, c'est plus la structure du style.

I-A - Étape 1 : ajout d'un élément de style

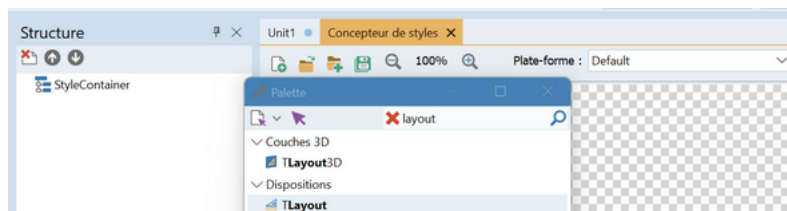
Si vous avez lu [ce tutoriel](#) et que vous êtes vraiment curieux, cliquez avec le bouton droit sur un des deux **TPanels** et choisissez l'option « modifier un style », vous avez alors accès à la structure définie dans le style par « défaut ».



panel1style1

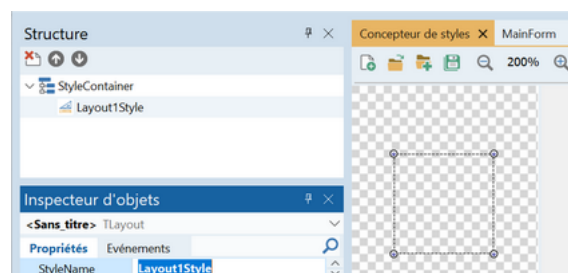
Vous constaterez qu'un panneau est on ne peut plus basique, il s'agit en fait d'un **TRectangle** coloré.

Nous pourrions donc nous contenter de déposer un **TRectangle** pour rester conforme au modèle mais je conseille plutôt de sélectionner un **TLayout** dans la palette de composant et de glisser celui-ci à la racine de l'arbre de la structure : **StyleContainer**.



Pourquoi commencer par ce conteneur ? C'est une bonne habitude à prendre si l'on veut utiliser ensuite, non pas un mais, des composants utilisant des alignements, je l'ai appris plusieurs fois à mes dépens me retrouvant alors avec une conception délicate, le conteneur couvrant la totalité de la zone de conception.

Une fois le **TLayout** ajouté, la propriété importante à renseigner est **StyleName** (1), c'est cet identifiant (propriété StyleName) que vous indiquerez ici qui sera utilisée ensuite quand vous indiquerez l'apparence à utiliser pour le panneau (propriété **StyleBook**)



style_layout

Indiquez **prectangle** comme identifiant (StyleName).



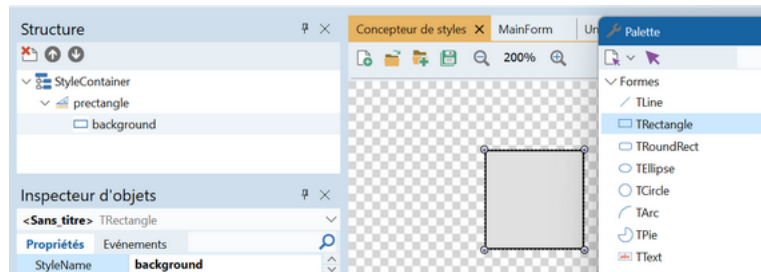
C'est à dessein que j'indique une valeur inexistante voir chapitre [I.C.1](#)

Contentez-vous, pour l'instant, d'appliquer la règle suivante :
Un élément à la racine de StyleContainer doit avoir un identifiant unique.

I-B - Étape 2 : Ajouter des composants dans le Layout

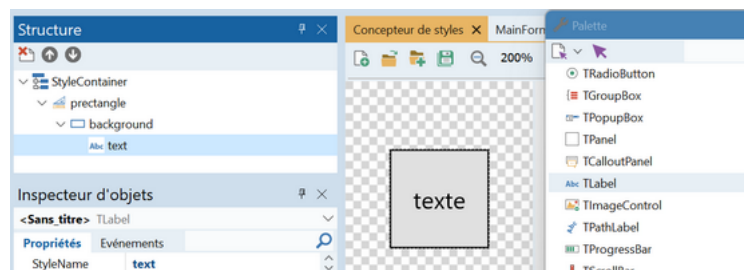
Il s'agit là, de mon point de vue, d'un des très grands avantages de Firemonkey, la possibilité qu'a chaque composant d'être le parent d'autres.

A ce **TLayout**, nous allons ajouter un **TRectangle**,



ajout TRectangle

dans lequel nous mettrons un **TLabel**.



ajout TLabel

Quelques retouches à faire, comme pour le contenant, nous allons modifier la propriété **StyleName** de ces ajouts. Sans être des conventions obligatoires, en pratique je vous suggère de donner au rectangle un identifiant comme **background** et **text** à celle du **TLabel**.

Si l'identifiant **background** n'est pas vraiment une règle, pour accéder aux propriétés du rectangle cela sera plus aisé (vous auriez tout aussi bien pu utilisé **rectangle**).

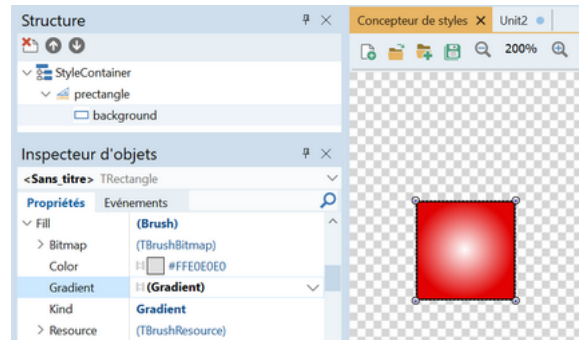
Nous verrons dans le chapitre **I.D.2** en quoi il est intéressant d'avoir indiqué **text** comme **StyleName** au **TLabel**

Changez aussi les alignements : propriété **Align** à **Content** ou **Client**.

I-B-1 - Améliorations possibles

Un élément de base (rectangle, cercle...) a bon nombre de propriétés facilement personnalisables en particulier le remplissage et la bordure.

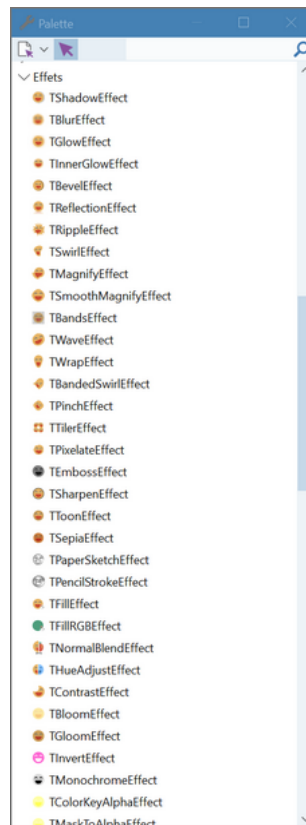
En exemple, le changement du remplissage en utilisant un gradient.



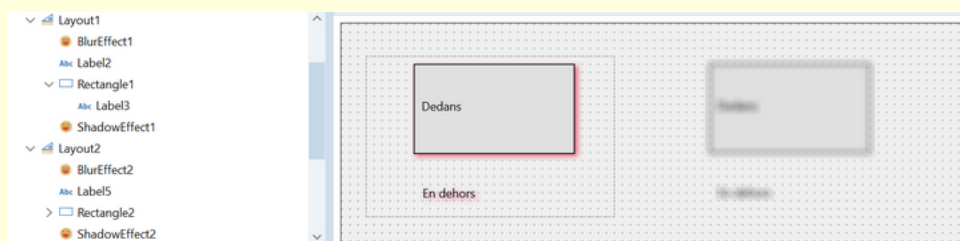
Mais l'on peut aussi ajouter **des effets** aux éléments.

Deux types d'effets :

. Ceux qui se retrouvent dans la palette de composants ;



Dire que tout les effets sont applicables à tout les composants visuels c'est peut-être s'avancer un peu trop. De plus, certains des effets ne fonctionnent pas ensemble. En exemple :




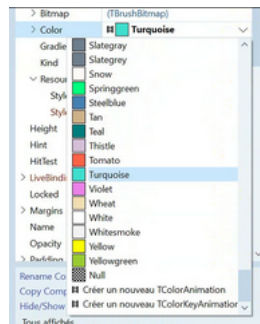
Dans la partie gauche de l'image, les deux effets, appliqués sur le conteneur (un **TLayout**) sont actif, l'effet de floutage (**BlurEffect**) ne fonctionne pas. A contrario, dans la partie droite, l'effet d'ombre (**ShadowEffect**) est désactivé d'où le rendu de floutage.

Donc, testez, re-testez et n'en abusez pas trop.



Attention à certains faux-amis comme **TFillEffect** que vous seriez tenté d'utiliser pour changer la couleur

. Plus difficiles à trouver, ceux « intrinsèques », créés via certaines propriétés du composant lui-même. Ces propriétés sont signalées par ce symbole  devant la valeur de la propriété. Un sous-menu permet alors de créer l'effet en utilisant l'option de menu « Créer un nouveau TColorAnimation... ».



Ces effets peuvent être actifs ou non mais aussi déclenchés selon certaines conditions via les **triggers**, une sorte de gestion d'évènement au niveau du style.

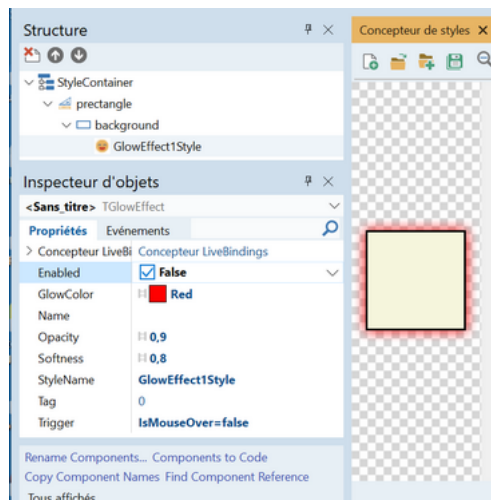


À ce propos, dans un style, les évènements d'un composant ne sont pas programmables contrairement à ce que l'inspecteur d'objets laisse supposer. Par contre, un expert pourra, à l'exécution, assigner des méthodes aux dits évènements.

Dans ce tableau, les triggers disponibles :

Nom du trigger	Applicable sur les composants et leurs descendants
IsDragOver	
IsFocused	
IsMouseOver	
IsVisible	
IsActive	TCustomForm
IsChecked	TMenuItem
IsOpen	TEffect
IsPressed	TCustomButton
IsSelected	TMenuItem, TTabItem, TListBoxItem, TTreeViewItem

Pour illustrer ce propos j'ajoute au **TRectangle** (identifiant background) un second effet de lumière, (**GlowEffect**), trouvé dans la palette des composants « Effets ». Cet effet sera désactivé au survol de la souris.



En général, c'est plutôt utilisé avec le trigger inverse (IsMouseOver=True).

Je l'ai écrit ainsi uniquement pour une capture d'écran facile.

Cerise sur le gâteau, et c'est aussi ce que je voulais signaler, si, durant ce design, vous positionnez la souris dans le cadre, l'effet n'est plus actif.

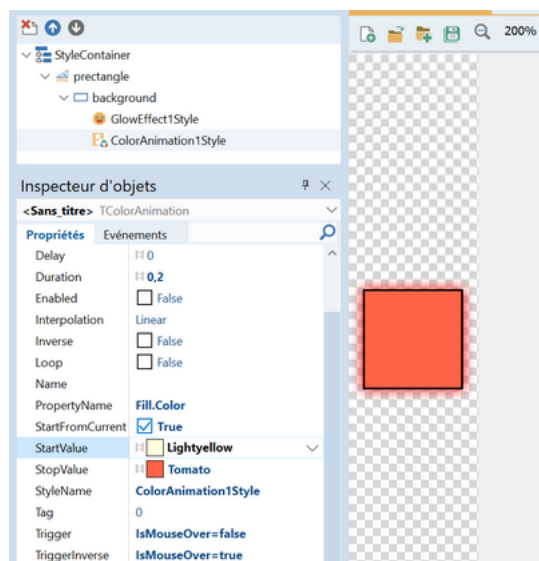
Comme je l'ai indiqué au début du chapitre **I.B**, l'avantage de FMX est que chaque composant peut être le parent d'autres composants. Pour corser et démontrer **TColorAnimation**, ajoutons en un.

Sélectionnez la couleur de remplissage du **TRectangle**. Au lieu de sélectionner une couleur, utilisez l'option (tout en bas du menu des couleurs) « *Créer un nouveau TColorAnimation...* ». Modifiez les couleurs de début et de fin (**StartValue**, **StopValue**) et indiquez les triggers de déclenchement.

Pour ce genre d'effet, il y a deux triggers à déclarer, **Trigger** et **TriggerInverse**. Bien sûr, vous alternerez sinon ce serait plutôt étrange.

Trigger
TriggerInverse

isMouseOver=false
IsMouseOver=true



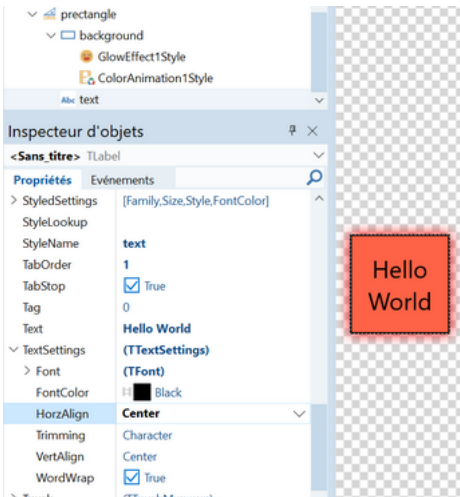
Déplacez ensuite la souris sur le composant pour voir ce qui se passe.

En dernier lieu je vais ajouter un **TLabel** (2) pour que cela ressemble à un bouton.

Deux choses sur ce dernier ajout :



- J'ai indiqué la propriété **StyleName=text**, j'y reviendrai ;
- J'ai renseigné la propriété **Text** ('Hello World') et modifié son alignement (**TextSettings.HorzAlign**).



Il faut savoir que l'on peut également combiner ces déclencheurs, exemple :

`IsMouseOver=true;IsPressed=false`

Vous avez donc, un nombre incroyable de possibilités, combinaisons d'effets et de composants laissant libre court à seule votre imagination!

I-B-2 - Pourquoi un TLabel et pas un TText ?

TLabel est un composant stylé contrairement à **TText**.

Qu'est-ce que cela implique ? Les caractéristiques de la police de caractères seront celles du style appliqué à la fiche (dans cette démo le style par « défaut »), sauf si vous stipulez les propriétés **StyledSettings** indiquant quelles parties du style vous voulez garder. En exemple si vous voulez uniquement changer la couleur, décochez **FontColor**.



Pour un **TText**, vous devrez indiquer toutes les caractéristiques de la police de caractères que vous désirez.

Il y a du pour et du contre

TLabel	Pour	Par défaut ce sont les caractéristiques définies du style de la fiche qui seront utilisées.
---------------	------	---

		Accéder aux propriétés du composant à l'exécution est assez simple.
	Contre	Ces mêmes caractéristiques peuvent s'avérer gênantes, le texte pouvant disparaître selon le fond défini (exemple : fond noir)
TText	Pour	Tout est à définir pour le composant, le style appliqué à la fiche n'a aucune répercussion.
	Contre	Toute modification de police de caractères devra s'opérer, si besoin, à l'exécution. N.B. Il est assez « galère » d'obtenir ces informations dans le but de les modifier.

I-C - Ajouter un second élément

Au choix :

- Vous pouvez répéter les étapes des chapitres **I.A** et **I.B** ;
- Copier l'élément ***prectangle*** déjà créé, et faire un coller dans le **StyleContainer**;

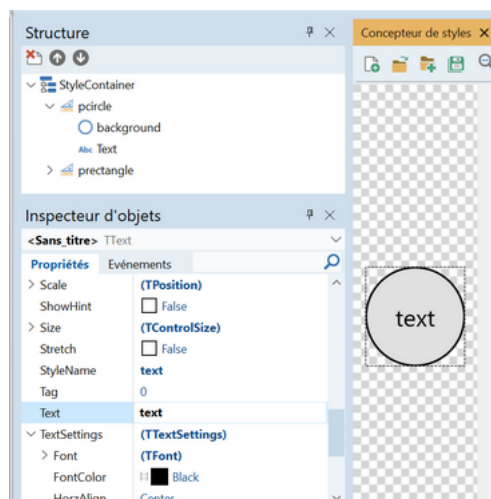
Assurez vous de bien changer l'identifiant du nouvel élément de style, c'est-à-dire la propriété **StyleName**, et que ce dernier soit unique (pour la démo : ***pcircle***).

Remplacez la forme de base (remplacer le ***TRectangle*** par un ***TCircle***)

Si vous optez pour l'option coller-copier, il faudra :

- Poser un ***TCircle***.
- Déplacer le ***TLabel*** de façon à ce que son parent soit le cercle.
- Supprimer le ***TRectangle***.

L'objectif est d'obtenir ceci



I-C-1 - La propriété StyleName

Quelques précisions à ce sujet. La règle énoncée à la fin du chapitre **I.A** n'est pas complète ou plutôt doit être nuancée.

« Un élément à la racine de **StyleContainer** doit avoir un identifiant unique. »

C'est applicable au sein d'un **TStyleBook** et encore, plus précisément, au sein de chaque parent (**StyleContainer** inclus).

Par contre, entre deux styles c'est loin d'être une obligation, a contrario cela peut même s'avérer payant.

Par exemple, vous désirez que vos **TPanel**s soient tous circulaires. La solution, renommer l'identifiant **pcirculaire** en **panelstyle**. L'identifiant **panelstyle** étant déjà utilisé par le style de la fiche (Rappel : comme nous n'en avons indiqué aucun, il s'agit du style par défaut), le **TStyleBook** personnalisé ayant priorité sur le style de la fiche, tout **TPanel** posé aura donc une forme circulaire.

D'où sort cet identifiant **panelstyle** ? La réponse n'est pas simple, même s'il s'agit pourtant d'un identifiant « standard ». De plus, comme il s'agit de personnaliser à partir des styles par défaut (contenu dans les ressources du programme) on ne peut même pas en étudier le texte a contrario des fichiers d'extension « style ».



Une règle comme : **type de composant(sans le T)||style** est peut-être applicable mais certainement pas universelle.

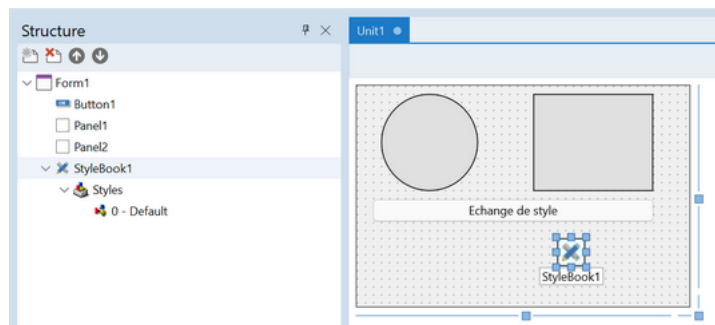
Étudier un fichier de style déborde largement de ce tutoriel, dans ma liste des choses à faire c'est en bonne place comme futur tutoriel.

I-D - Tester

Pour tester, déposez sur la fiche deux composants stylés, au plus simple : deux **TPanel**.

Pour l'un des deux indiquez la propriété **LookupStyle** avec la forme rectangle, pour le second, évidemment, la forme de cercle.

L'IDE (Environnement de Développement Intégré) réagit aussitôt et vous obtenez



I-D-1 - Modifier le style d'un composant à l'exécution

Il faut passer par la modification de la propriété **StyleLookup** mais il ne faut pas oublier de l'appliquer ce que fait l'instruction **ApplyStyleLookup** d'un composant stylé.

exemple

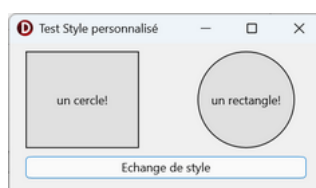
```
Panel1.StyleLookup := 'prectangle';
Panel1.ApplyStyleLookup;
```

L'accès au texte n'est pas si simple que ça non plus (dans le cas d'un TPanel, contrairement à celui de la VCL, pas de propriété caption). Il faut utiliser la méthode **FindStyleResource** pour retrouver un élément du style.

```
// TText plutôt que TLabel permet d'obtenir plus de possibilités de
// mise en forme
TText (Panel1.FindStyleResource('text', False)).text := 'un cercle!';
```

Pour la démonstration il suffit d'ajouter un bouton (**TButton** ou **TCornerButton**) et de coder son évènement **onClick**.

```
procedure TMain.Button1Click(Sender: TObject);
begin
  if Button1.Tag = 0 then
  begin
    Panel1.StyleLookup := 'prectangle';
    TText (Panel1.FindStyleResource('text', False)).text := 'un cercle!';
    Panel1.ApplyStyleLookup;
    Panel2.StyleLookup := 'pcercle';
    TText (Panel2.FindStyleResource('text', False)).text := 'un rectangle!';
    Panel2.ApplyStyleLookup;
    Button1.Tag := 1;
  end
  else
  begin
    Panel1.StyleLookup := 'pcercle';
    TText (Panel1.FindStyleResource('text', False)).text := 'un cercle';
    Panel1.ApplyStyleLookup;
    Panel2.StyleLookup := 'prectangle';
    TText (Panel2.FindStyleResource('text', False)).text := 'un rectangle';
    Panel2.ApplyStyleLookup;
    Button1.Tag := 0;
  end;
end;
```

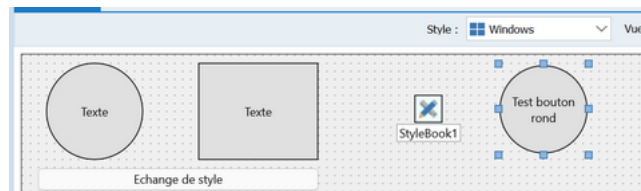


Résultat au premier clic

I-D-2 - Utilisation sur un bouton

Pas très flagrant l'utilisation avec un **TPanel** ?

Déposez un nouveau bouton sur la fiche en indiquant le **StyleLookup** contenant le cercle. Saisissez ensuite un libellé (propriété **Text**) pour ce bouton.



Vous remarquerez que l'identifiant de style pour le libellé est **text**, tout comme le nom de la propriété.

Codifiez l'évènement **onClick** de ce bouton pour faire une opération quelconque par exemple afficher un simple message.

```
procedure TMain.BoutonRondClick(Sender: TObject);
begin
  Showmessage('Clic sur le bouton');
end;
```

Cela ne fonctionne pas !



Vous venez de tomber dans un piège fréquent : La propriété **HitTest** des composants enfant.

Faites bien en sorte que les composants enfants (le cercle et le texte) aient cette propriété décochée. Fermer le concepteur et le problème est corrigé.

II - Plus loin que les formes basiques

Dans le premier chapitre nous avons utilisé deux formes de bases, le rectangle et le cercle.

Il en existe quelques autres dans la palette des composants rubrique **Formes**.

Ce chapitre va introduire les chemins vectoriels du composant **TPath** pour diversifier les formes que l'on peut obtenir.

Le composant **TPath** est un composant de FireMonkey (FMX) utilisé pour définir des formes de type chemin 2D, représentant des groupes de courbes et de lignes connectées. Ces spécifications sont inscrites dans la propriété **Data** de type **TPathData**. Les instructions de dessin suivent la **norme SVG (Scalable Vector Graphics) 1.0**.

En résumé, **TPath** permet de créer des dessins 2D complexes en utilisant des instructions de chemin similaires à celles utilisées dans les fichiers SVG (Scalable Vector Graphics).



À ne pas confondre ce **TPath** inclus dans l'unité **FMX.Objects** et le **TPath** de l'unité **System.IOUtils** permettant la gestion de fichier.

Si je pouvais faire une critique, ce **TPath** aurait dû être nommé autrement pour éviter les confusions.

II-A - Création du nouvel élément

Ouvrez à nouveau le concepteur de styles.

Déposez un nouveau **TLayout**, donnez lui comme identifiant (propriété **StyleName**) « **untriangle** ».

Déposez dans ce composant un **TPath** de la palette, utilisez l'alignement **Client**.

La propriété clé de ce composant, c'est la propriété **Data.Data** qui va contenir le dessin.

La personnalisation des couleurs de remplissage et contours est identique à celle des autres composants déjà vu, chapitre **I.B.1**.

Pour un triangle j'indique : M0,50 L100,100 L100,0 Z

Oui, ce n'est pas forcément évident. Il n'est pas non plus dans mon intention d'expliquer le dessin par vectorisation il y a de bons tutoriels et les références que vous retrouverez dans le chapitre **IV.B**.



Pour tout vous avouer, à mes débuts d'utilisation de ce composant, j'allais piocher **dans ce site** et extrayais du SVG, la substantifique moelle, c'est à dire la partie notée entre les balises **<path>**.

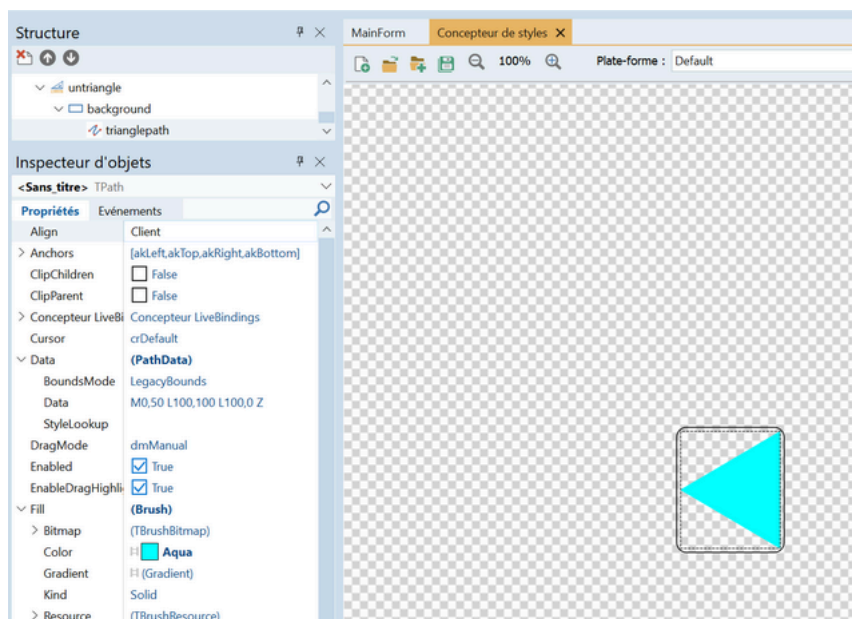
Pour plus de détail, je vous invite à aller dans **mon dépôt GitHub**, vous y trouverez le PDF d'un tutoriel non publié, ni corrigé et désormais en parti dépassé.

J'ai voulu montrer quelques modifications à l'exécution aussi ne suis-je pas resté sur un simple **TPath**, préférant l'inclure dans un rectangle au coins arrondis.

Pour y arriver, j'ai posé au sein du **TLayout**, un **TRectangle**. Les coins arrondis sont des propriétés de ce dernier : **CornerType** à **Round** et **Xradius** et **Yradius** mise à la valeur 10.

De même, pour éviter un chevauchement, j'ai défini toutes les valeurs de **Padding** du **TRectangle** à 5.

Enfin j'ai déplacé le composant TPath pour qu'il devienne un enfant du TRectangle.



Comme d'habitude, clôturez le concepteur pour sauvegarder les modifications effectuées. Déposez ensuite un **TPanel** et indiquez la propriété **StyleLookup** (**untriangle**).

II-A-1 - Accéder aux propriétés de ce style

Comme l'indiquent les textes de légende, je vais coder l'évènement **OnClick** du panneau **TPanel**, que j'ai renommé **PTriangle**, pour faire varier les couleurs de façon aléatoires.

Ce code me permet d'introduire deux manières différentes d'utiliser la fonction **FindStyleResource**.

Faire varier les couleurs

```
procedure TMain.PTriangleClick(Sender: TObject);
var Triangle : TFMXObject;
    rectangle : Trectangle;
begin
    // deux manières d'accéder aux éléments de style
    // FindStyleResource
    triangle:=PTriangle.FindStyleResource('trianglepath');
    if Assigned(Triangle) AND (Triangle is FMX.Objects.TPath) then
        FMX.Objects.TPath(Triangle).fill.Color:=TAlphaColorF.Create( random(255) / 255,
        random(255) / 255, random(255) / 255, 1).ToAlphaColor;
    // FindStyleResource<Type>
    PTriangle.FindStyleResource<Trectangle>('background',rectangle);
    if assigned(rectangle) then
        begin
            rectangle.Fill.kind:=TBrushKind(ifthen(random(2)<1,0,1));
            rectangle.fill.color:=TAlphaColorF.Create( random(255) / 255, random(255) / 255,
            random(255) / 255, 1).ToAlphaColor;
        end;
end;
```

À l'exécution vous pourriez obtenir quelque chose comme ceci :



i C'est du pur hasard donc ne vous attendez pas forcément à obtenir ceci.

J'ai dû cliquer plusieurs fois sur ce pseudo bouton pour avoir quelque chose à mon goût.

II-A-2 - Osez plus : le composant TSKSVG

L'inconvénient du composant **TPath**, c'est qu'il est « mono-couleur » si l'on excepte l'utilisation en remplissage de gradient ou d'image.

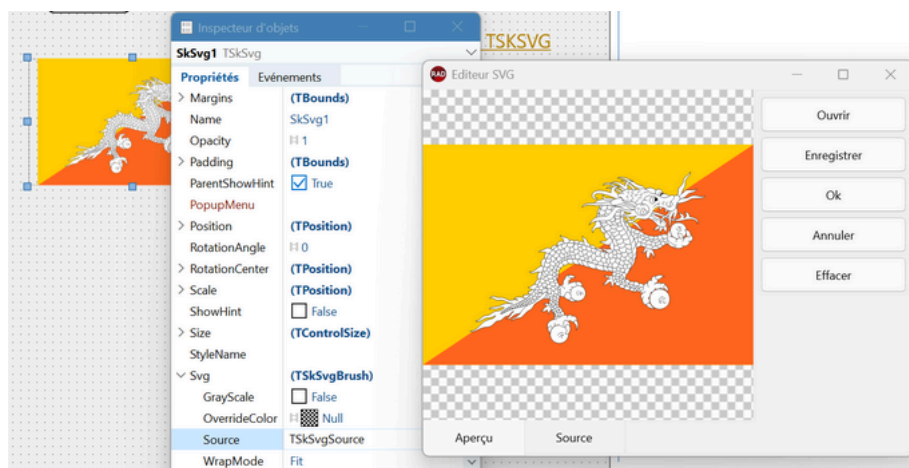
L'utilisation de la bibliothèque Skia est depuis peu intégrée à Delphi, ce qui nous permet d'obtenir des dessins proches des images au format **.png** sans les inconvénients inhérents aux diverses résolutions des écrans.

Déposez un composant **TSKSVG** sur votre forme, choisissez votre remplissage à partir d'un fichier.



Personnellement j'aime bien utiliser des dessins complexes pour mes essais. Ma source préférée : Wikipédia et en particulier les drapeaux de pays.

J'ai choisi de montrer le drapeau du Bhoutan qui est quand même bien complexe niveau dessin !



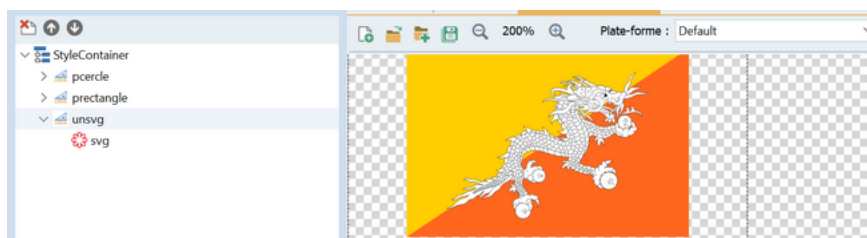
La question qui se pose est alors « peut-on utiliser ce composant au sein d'un style et comment le modifier ? »

Pour ce qui est de l'utilisation, pas de problème, les mêmes techniques que précédemment fonctionnent parfaitement à savoir :

- Déposer un **TLayout**, lui donner un identifiant de style.
- Mettre un composant **TSKSvg** « à l'intérieur », et jouer sur les alignements.



Là encore, l'utilisation du **TLayout** n'est pas une obligation mais s'avère très pratique dès que l'on veut centrer l'image plutôt que d'utiliser un alignement qui pourrait déformer le dessin.



Le programme de démonstration va vous proposer trois méthodes de modification et cela via l'utilisation d'une boîte de choix déposée en dessous d'un nouveau **TPanel** utilisant le nouveau style défini (**stylename = unsvg**).

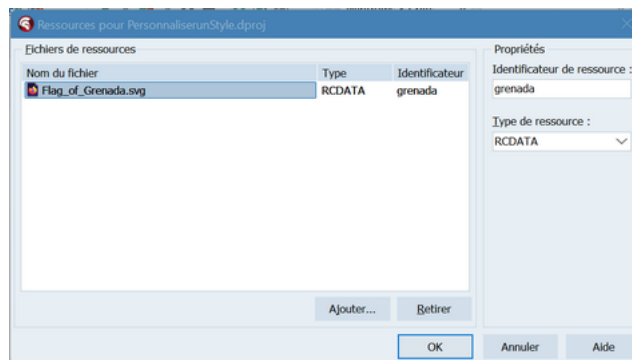
La boîte de choix va contenir quatre éléments :

- Bhoutan (fichier)
- France
- Ukraine
- Grenades (ressource)



Quant au dernier, je l'ai inséré directement en tant que ressource.

Pour cela j'ai fait appel au menu de l'IDE (**Projet/Ressources et images...**) ce qui m'a ouvert ce dialogue :



```
procedure TMain.ComboBox1Change(Sender: TObject);
const
france = '<svg xmlns="http://www.w3.org/2000/svg" width="900" height="600">'+
'<path fill="#CE1126" d="M0 0h900v600H0"/>'+
'<path fill="#fff" d="M0 0h600v600H0"/>'+
'<path fill="#002654" d="M0 0h300v600H0"/>'+
'</svg>';

const
ukraine = '<svg xmlns="http://www.w3.org/2000/svg" width="1200" height="800">'+
'<rect width="1200" height="800" fill="#0057B7"/>'+
'<rect width="1200" height="400" y="400" fill="#FFD700"/>'+
'</svg>';

var
rs : TResourceStream;
st : TStringStream;
begin
case ComboBox1.ItemIndex of
0:
begin
TSKSVG (Panel4.FindStyleResource('svg')).svg.Source :=
TFile.ReadAllText('..\..\Flag_of_Bhutan.svg');
end;
1:
TSKSVG (Panel4.FindStyleResource('svg')).svg.Source := france;
2:
TSKSVG (Panel4.FindStyleResource('svg')).svg.Source := ukraine;
3:
begin
Rs := TResourceStream.Create(MainInstance, 'grenada', RT_RCDATA);
st := TStringStream.Create;
try
st.LoadFromStream(Rs);
```

```
TSKSVG(Panel4.FindStyleResource('svg')).svg.Source :=
    st.ReadString(st.size);
finally
    Rs.free;
    st.free;
end;
end;
end;
end;
```



Avec un peu de connaissance en SVG (Scalable Vector Graphics), il est tout à fait possible de modifier les sources de ces dessins à l'exécution.

Un exemple dans cette discussion

III - L'utilisation des images

A ce stade si, par curiosité, vous avez ouvert des fichiers de style de type texte (**.style**) via un éditeur de texte, vous vous posez certainement la question sur le fait que je ne l'ai pas mentionné avant.


La première raison est toute personnelle : je préfère économiser les octets, un SVG (Scalable Vector Graphics) étant souvent beaucoup moins « lourd » qu'une image, image qui je le rappelle doit être déclinée en plusieurs tailles (x 1, x 1.5, x 2, et même désormais x 3) à cause des résolutions d'écran.

La seconde est, je l'avoue, que je maîtrise moins bien cette technique. Pour moi c'est assez difficile de bien sélectionner les zones, j'y reviendrai chapitre **III.B.1**.

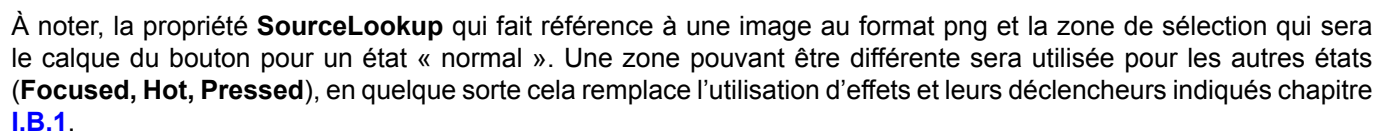
III-A - État des lieux

Pas tous, mais beaucoup de fichiers de style contiennent des images. Même le style par défaut en contient.

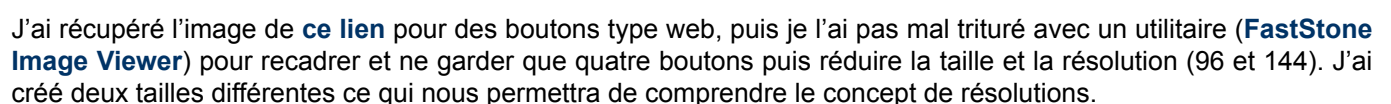
Pour les besoins de la démonstration :

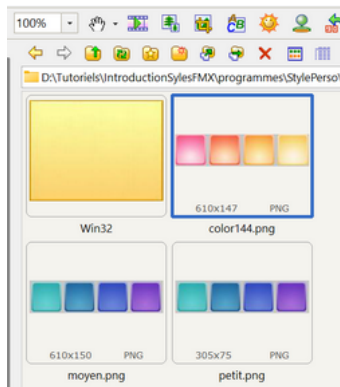
- Créez une nouvelle fiche multi-périphérique.
- Déposez un bouton.
- Faites un clic droit sur celui-ci.
- Utilisez l'option **Modifier un style...** (personnalisé ou non, à ce stade aucune importance)
- Sélectionnez le background, vous remarquerez qu'il ne s'agit pas d'un **TRectangle** mais d'un **TButtonStyleObject**.
- Sélectionnez la propriété **NormalLink**.
- Cliquez sur l'éditeur (le bouton )

Vous allez obtenir ceci :



Quand l'utiliserai-je ? Quand un dessin vectoriel ne peut pas faire le job, mais cette réponse est toute personnelle. Disons plutôt que l'on l'utilisera pour des besoins d'images, je pense à de jolis boutons de couleurs, des caractères avec des polices particulières etc.





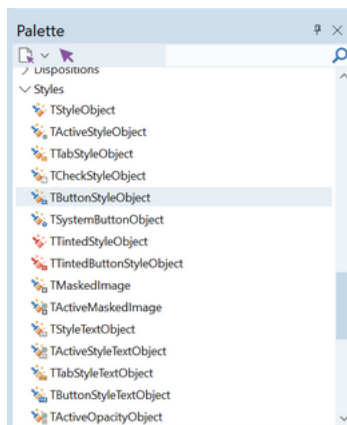
Ayez la curiosité de regarder dans l'archive du programme de démonstration, la taille des fichiers obtenus. N'oubliez pas que ce ou ces fichiers utilisés se retrouveront au sein de votre programme et donc, augmenteront sa taille d'autant. Effrayant !

III-B-1 - Utiliser l'image

Ouvrez le **TStyleBook**, ajoutez un **TLayout**, donnez lui un identifiant de style (j'ai choisi de le nommer **webbutton**).

Ces opérations, je pense que désormais, vous devez commencer à les maîtriser.

Cette fois, nous n'allons pas utiliser un des composants de la palette **Formes** mais un de ceux de la palette **Style**.



Il en existe quand même pas mal, comme nous allons définir un bouton, de surcroît pour une application Windows, mon choix s'arrête sur le composant **TButtonStyleObject**.

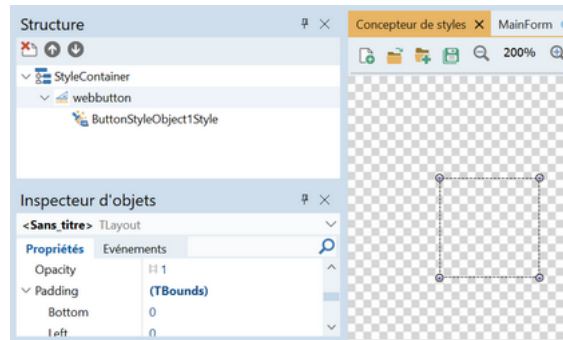
Pour ma part, ayant très peu utilisé ces objets, je les maîtrise mal.



Je vous invite à lire [la documentation officielle](#) pour de plus amples informations.

Malheureusement, vous constaterez qu'elle n'est que peu fournie, mes recherches plus précises sur un de ces composants me renvoient toujours au lien déjà cité.

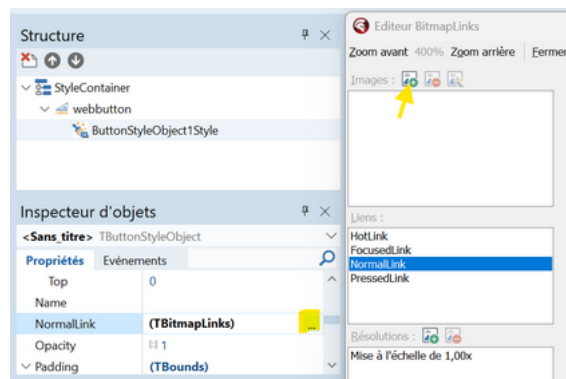
Déposez donc ce composant dans le **TLayout**, n'oubliez pas l'alignement (**client**) et de donner un identifiant (propriété **StyleName**) si d'aventure vous vouliez, à l'exécution, le triturer un peu (l'identifiant par défaut : **ButtonStyleObject1Style**, même s'il est explicite, n'est quand même pas très maniable) .




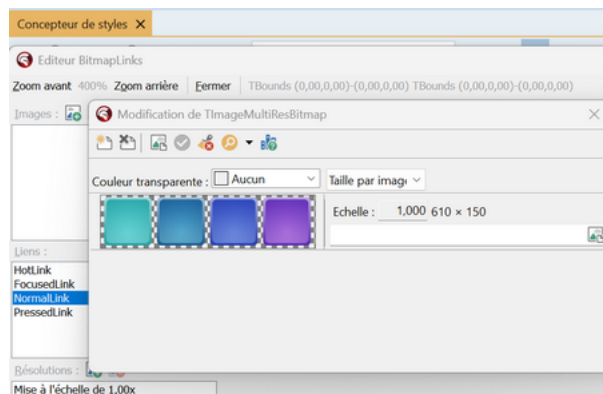
Sélectionnez le composant de style et, plus particulièrement, une des propriétés dont le nom se termine par **link**.



Mon conseil, commencez par **NormalLink**.



Utilisez le bouton  qui, aucune image n'étant définie, va vous ouvrir un dialogue de sélection .



Une fois l'image sélectionnée (j'ai délibérément choisie une petite taille), nous devons en sélectionner une zone.



C'est, sans aucun doute, la partie la plus délicate de l'opération!

Vous apercevez un point vert au centre, beaucoup moins visibles vous découvrirez des points jaunes au quatre coins ainsi que deux points bleus (coins supérieur gauche et inférieur droit).

Comme on le voit sur l'image, les points jaunes sont presque invisibles, déplacez légèrement les deux points bleus. Le coin supérieur gauche vers le bas et la droite de façon à obtenir une petite marge et répétez l'opération pour le coin inférieur droit.



Tentez d'obtenir une marge plus ou moins de la taille de la marge d'un des boutons. Vous pourrez ajuster ensuite.

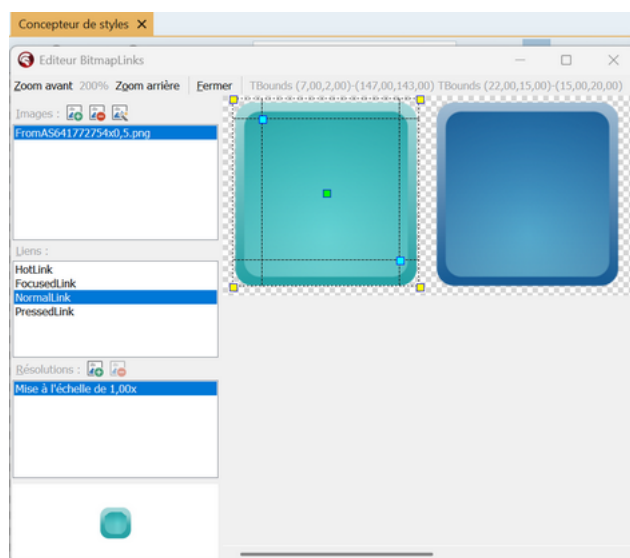
pour plus de détails, je vous renvoi vers [le tutoriel de Rémi Gouyon](#) ou sur [ce billet de Ray Konopka](#), ce dernier bien que parlant plus des styles VCL, est édifiant sur beaucoup de points.

Une fois ces marges obtenues, vous devriez voir plus facilement ces points jaunes.

Utilisez désormais les points jaunes pour retailer l'image.

N'hésitez pas à utiliser le zoom !

L'objectif, définir le cadre et le fond du bouton, comme ci-dessous.



À ce stade, répétez l'opération pour les trois autres types de liens.

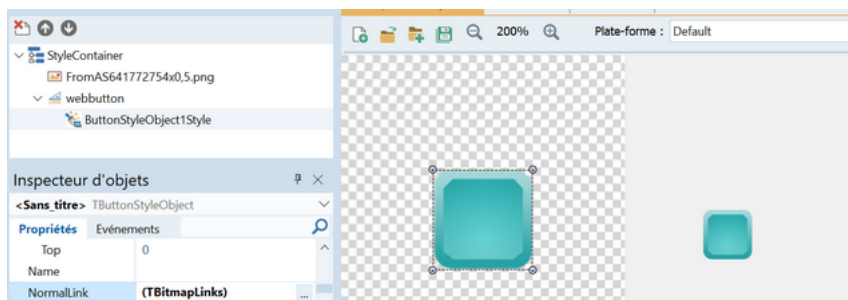
NormalLink	Image du bouton.
PressedLink	Image du bouton lors de l'appui sur celui-ci.
FocusedLink	Image du bouton si celui-ci est le contrôle actif.
HotLink	Image du bouton lors de son survol (avec la souris).

Vous remarquerez en bas à gauche une première représentation de votre futur bouton.

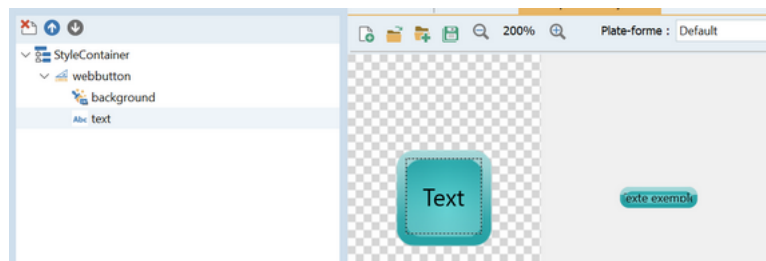
Une fois les sélections faites, fermez l'« **Éditeur de BitmapLinks** ».



Déplacez la souris sur le bouton représenté (celui de droite). Si vous avez déclaré une zone pour **FocusedLink**, il changera de couleur.



N'oubliez pas d'ajouter à ce style, un **TLabel** ou un **TText** (rappel de la différence, le tableau **TLabel vs TText I.B.2**). Dans ce cas précis, j'ai plutôt choisi un **TText** ne voulant pas que la couleur puisse changer en fonction de ce que le **StyleManager** pourrait appliquer plus tard, ce qui pourrait rendre le texte peu visible, imaginez, par exemple que le style propose une couleur bleu clair ! Peu probable mais...

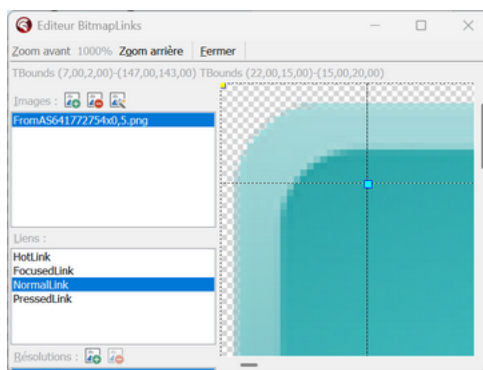


III-B-2 - Résolutions des écrans

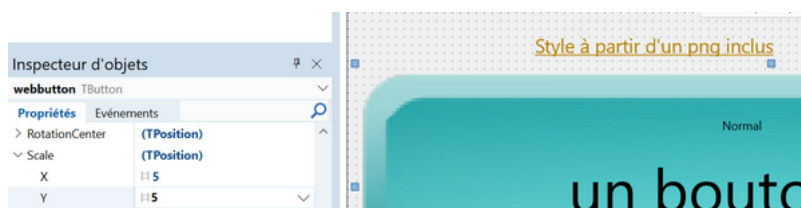
Pourquoi ajouter des images pour prendre en compte diverses résolutions ? Pour éviter les crénelages (phénomène aussi nommé pixellisation)

Deux exemples en « poussant le bouchon », toujours avec le style ne contenant que la plus petite des images.

- En utilisant le zoom de l'**Éditeur de BitmapLinks**, indiquez une valeur importante ;



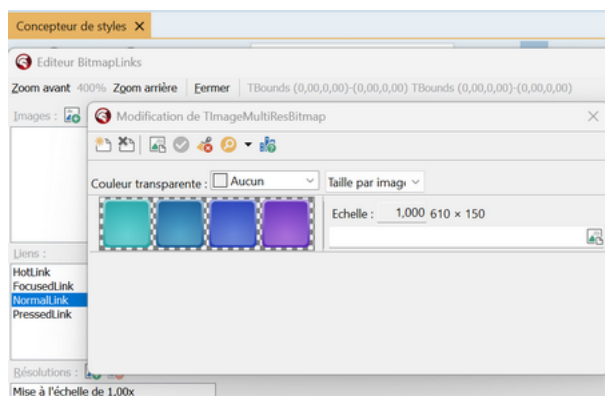
- En modifiant le facteur d'échelle au design.




III-B-2-a - Ajout d'images pour d'autres résolutions

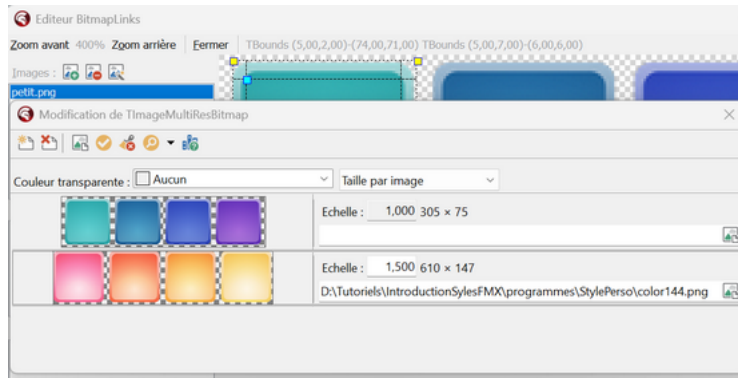
Pour la modification de l'image, commencez par ouvrir l'une des propriétés de lien et sélectionnez l'image.

Cliquez sur le bouton de modification  pour ajouter une nouvelle image.



Ajoutez la nouvelle image avec le bouton  en n'oubliant pas de spécifier l'échelle de celle-ci.

 Pour les besoins de la démonstration, je me suis amusé un peu en changeant carrément de code couleurs. Ce n'est pas forcément ce que l'on ferait dans une application.




Validez les modifications d'image  et fermez ce dialogue

Nous devons alors ajouter des échelles en utilisant cette barre d'outil,

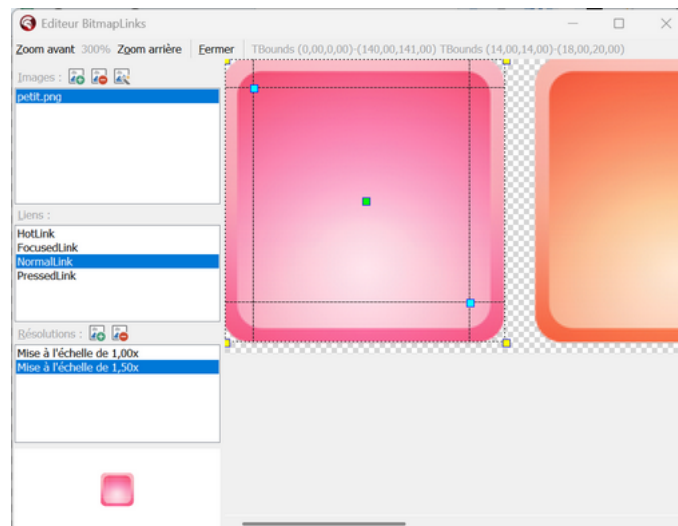


qui ouvre ce dialogue

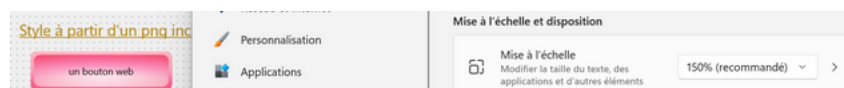



 Sans être une recommandation, il faudrait ajouter les échelles 1.5, 2 et même 3 pour couvrir les résolutions possibles des écrans actuels.

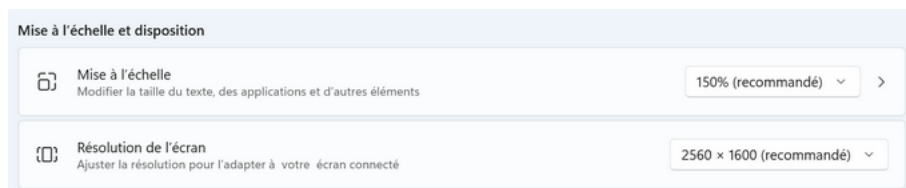
Ensuite, vous devrez répéter les sélections de zones pour le bouton avec à l'échelle 1.5x



Si, comme moi, vous avez un écran de résolution plus élevé (dans mon cas 150%), à la sauvegarde du style, le bouton est devenu rose.



Tester les différentes résolutions est assez fastidieux puisque vous devrez modifier les paramètres d'affichage de votre écran ( **Paramètres/paramètres d'écran**) et jouer avec l'échelle.



debutzoom[D:\Tutoriels\IntroductionStyleFMX\personnalisation\images\echelle100.png]finzoom



Avertissement

J'ai eu quelques problèmes pour redéfinir certains des liens pour la résolution 1,5x.
Les **PressedLink** et **HotLink** refusaient systématiquement de passer sur la seconde image.

Symptôme : Mise à l'échelle de 1,00x apparaissait deux fois !

Je vous livre mon astuce :

- Sauvegarder, via le le style dans un fichier ;
- Éditer le fichier via l'IDE ou un éditeur de texte ;

Style

```
PressedLink = <
  item
    CapInsets.Left = 9.0000000000000000
    CapInsets.Top = 9.0000000000000000
    CapInsets.Right = 8.0000000000000000
    CapInsets.Bottom = 7.0000000000000000
    SourceRect.Left = 155.0000000000000000
    SourceRect.Top = 3.0000000000000000
    SourceRect.Right = 224.0000000000000000
    SourceRect.Bottom = 71.0000000000000000
  end
  item
    CapInsets.Left = 10.0000000000000000
    CapInsets.Top = 12.0000000000000000
    CapInsets.Right = 13.0000000000000000
    CapInsets.Bottom = 15.0000000000000000
    Scale = 1.5000000000000000
    SourceRect.Left = 310.0000000000000000
    SourceRect.Right = 452.0000000000000000
    SourceRect.Bottom = 142.0000000000000000
  end>
```

- 3 Ajouter l'élément **Scale** manquant (en gras) ;
- 4 Recharger le fichier modifié toujours via le concepteur de styles ;
- 5 Et définir à nouveau les liens erronés ;
- 6 N'oubliez pas ensuite de sauvegarder à nouveau (pour la postérité) mais, surtout de fermer le concepteur afin que le style ainsi modifié soit appliqué dans le source de la fiche (fichier **.fmx**).



Ouvrir ce fichier de style, comme il s'agit d'un fichier texte, augmentera peut-être votre curiosité.

Il n'est pas dans mon intention dans cette introduction d'en faire une plus ample étude.

III-B-3 - Conclusion sur les résolutions

Tout d'abord, cela va beaucoup dépendre des images utilisées. En second lieu, plus il y aura d'images plus la taille du style va augmenter. Et enfin, il faudrait dans le cas d'un **TButtonStyleObject** répéter à chaque fois les opérations décrites au début du chapitre **III.B.1**.

Dans le cas de trois résolutions, cela revient à répéter les opérations de sélections douze fois !

Loin d'être agréable, surtout que, sans rentrer dans les détails et à peaufiner, il est possible d'avoir un rendu assez approchant uniquement en utilisant ce que l'on a vu chapitre **I**.



IV - Références

Tout d'abord, je tiens à indiquer que ce tutoriel fait logiquement suite à mon **Introduction aux styles FMX**. Un rappel de quelques liens déjà proposés :

- [🇬🇧 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.](#)
- [🇬🇧 Things I Learned Building a Custom VCL Style](#)

Du même genre que mon tutoriel, vous trouverez **Personnalisation des styles FMX de Rémi Gouyon**.

Les articles ou vidéos que j'ai pu consulter sont nombreux, je les classe en deux thèmes différents.

IV-A - Le concepteur de styles

En ce qui concerne la conception je retiendrai surtout ceux-ci :

La documentation officielle **Personnalisation de applications Firemonkey** est à lire.

L'incontournable site [🇬🇧 delphistyles](#) vous montrera beaucoup de possibilités.

Avant d'être des styles Firemonkey, ce sont souvent des styles VCL qu'un outil fourni, le concepteur de styles de bitmaps, permettra de sauvegarder en style Firemonkey. Donc, toute littérature ou vidéo sur cet outil est bonne à consulter.

[🇬🇧 Videos CodeRage XI](#)

[🇬🇧 Creating custom VCL styles](#) Code Rage 2022

mais lisez aussi [🇬🇧 l'expérience de Ray Konopka](#) qui douche un peu ce bel optimisme.

IV-B - Les graphiques vectoriels évolutifs (SVG)

Je mettrai en avant, ce tutoriel [🇫🇷 SVG, la syntaxe Path](#).

Pour plus, la [Documentation Mozilla](#) et, plus spécifiquement pour *Path* [ce tutoriel](#).

N'oubliez pas que SVG du W3C (World Wide Web Consortium) répond à une norme la **version 1.1**, publiée en 2011.

Où trouver des icônes en SVG :

[SVG Silh](#), libre de droits et gratuites.

Sur le site de [pictogrammers](#) ou [icones8](#), attention aux licences possibles.

Et certainement bien d'autres.

V - Débriefing

Dans cet article, je vous ai présenté mes trucs et astuces pour ajouter des représentations de composants existants (style) et même changer leur dessin à l'exécution. Deux techniques, bien que complémentaires, s'opposent : l'utilisation de formes de base ou l'utilisation de zones d'une image.

Ai-je été objectif ? Je crains que non, car je préfère l'utilisation d'éléments simples (formes) à l'utilisation d'images (objet styles).

Ma seconde technique, consistant à isoler ces ajouts, des fichiers de style, permet de ne pas avoir à dupliquer ces personnalisations si l'on pense à utiliser plusieurs thèmes (comme par exemple un clair et un sombre).

Croyez-moi, je n'ai fait qu'entrouvrir une boîte, tant les possibilités sont nombreuses.

Faites vos propres expériences en fonction de vos besoins, n'hésitez pas à ouvrir les fichiers **.Style** pour découvrir ce qui s'y cache.

Vous me trouverez souvent disponible sur le [forum](#) prêt à répondre à vos questions. Tout challenge sur le sujet m'intéresse. Il y a des pièges dont je n'ai pas parlé dans cette introduction et je compte bien rédiger plusieurs autres tutoriels, de plus en plus pointus.

VI - Remerciements

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

1 : Voir chapitre **I.C.1**

2 : Voir chapitre suivant **I.B.2**