FMX – Boîte de choix

Auto complétion et alternatives

Par Serge Girard

Date de publication : 21 février 2019

TOUT PUBLIC

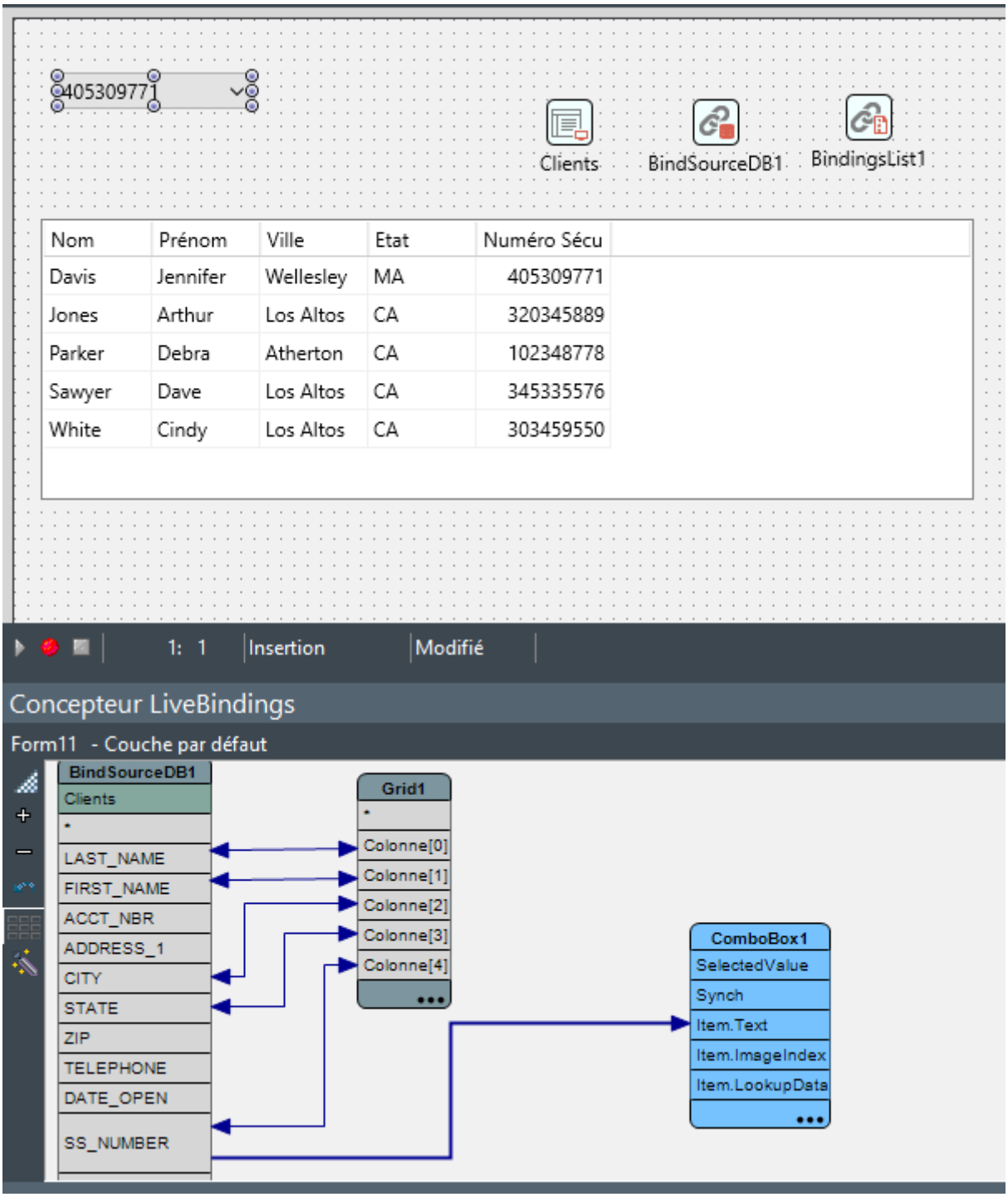
Au cours de mes développements d'application, j'ai souvent eu à proposer des boites de choix à l'utilisateur, choix généralement fournis à partir d'une autre table de données. Remplir une boîte de choix (**TComboBox**) par l'intermédiaire des LiveBindings est chose aisée, plus ardu est d'aider l'utilisateur à son choix lorsque les propositions sont nombreuses, en exemple : sélectionner un client pour obtenir son code. Le challenge de cet article est de proposer et étayer plusieurs solutions en fonction des besoins.

En complément sur Developpez.com

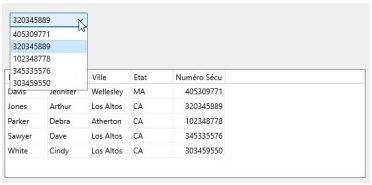
- LiveBindings et POO

I - Illustration

L'application que je présente dans ce chapitre n'est là que comme base de départ de la réflexion. J'utilise un fichier des clients disponible dans le répertoire exemple de Delphi (***<répertoire d'installation>\Samples\Data\clients.cds***). En voici le design :



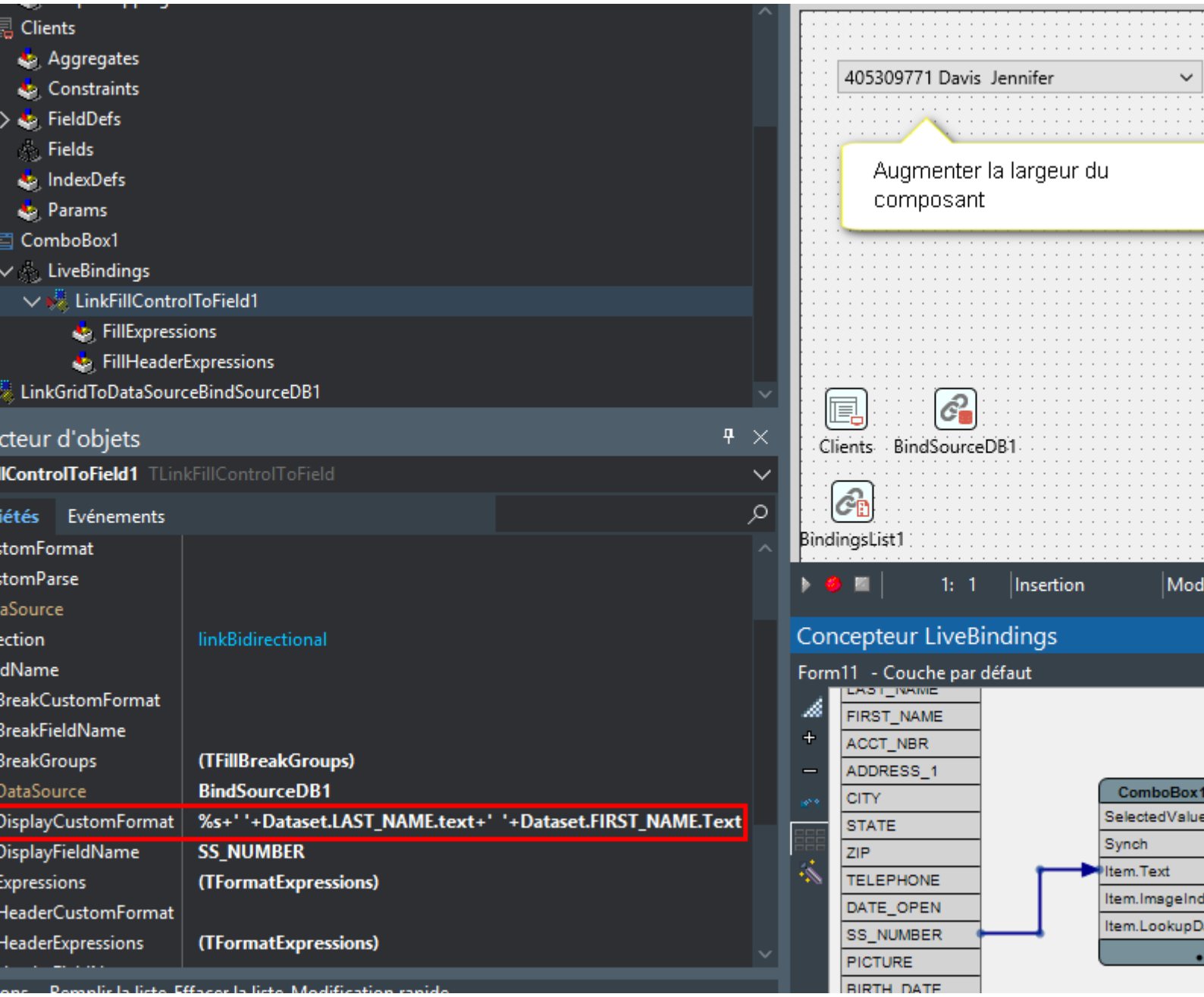
À l'exécution j'obtiens ceci :



La grille n'est là que pour montrer les données chargées. Dans la réalité, seule la boite de choix sera utilisée.

L'objectif, sélectionner le numéro du client est atteint mais, à moins que l'utilisateur ne connaisse le petit nom de chacun en fonction de son numéro, cette boîte de choix n'est pas très significative. Est-ce améliorable ? Heureusement oui, en utilisant la propriété **FillDisplayCustomFormat** de la liaison.

```
%s+' '+Dataset.LAST_NAME.Text+' '+Dataset.FIRST_NAME.Text
```



Cette manipulation rend certes la boîte de choix plus explicite mais me force à élargir le composant afin de voir le texte entier. Je signale que jouer sur la propriété ***ItemWidth*** permet d'obtenir une liste plus large que la boîte de choix.

Bien sûr, la « formule » de l'expression peut être changée. Notez surtout le ***Dataset.<nom_de_colonne>.text*** *qui permet d'accéder aux valeurs.*

II - Auto complétion

La première chose pratique à proposer à l'utilisateur c'est bien cette fonctionnalité, surtout si la liste est importante. Le principe en est relativement simple, si l'utilisateur frappe des touches en un laps de temps donné, la boîte se positionne sur l'élément correspondant au plus près au groupe de touches frappées, cela évite à l'utilisateur de faire une recherche dans toute la liste en utilisant le défilement.

II-A - Première solution : dériver le composant

Simple dans le principe et impliquant peu de code. Deux variables privées sont à ajouter au **TCombobox** de base, une pour tester les délais (***chronometre***) et une autre pour mémoriser les touches(***keys***). En dernier lieu, la procédure **OnKeyDown** sera surclassée pour implémenter la fonctionnalité souhaitée.

```
type
  // inconvénient tous les comboboxs auront la fonctionnalité
  TCombobox = class(FMX.ListBox.TComboBox)
  private
    Chronometre:TDatetime;
    Keys:string;
  protected
    procedure KeyDown(var Key: Word; var KeyChar: System.WideChar; Shift: TShiftState);override;
  end;
```


Vous noterez la formulation de la dérivation TCombobox = class(FMX.ListBox.TComboBox)

et non TCombobox = class(TComboBox).

*Pourquoi ce choix ? Parce que cela me permettra d'accéder à des propriétés non publiées, comme **Count** ou **ItemIndex**, ou inaccessible comme **Items**.*

```
{ TCombobox }

procedure TCombobox.KeyDown(var Key: Word; var KeyChar: System.WideChar;
  Shift: TShiftState);
var I: Integer;
begin
  if key=vkReturn then exit;
  if CharInSet(keychar,[chr(48)..chr(57),chr(65)..chr(90),chr(97)..chr(122)])
   then begin
    // préférences personnelles de délai 500 ms
    if MilliSecondsBetween(Chronometre,Now)<500
      then keys:=keys+keychar
      else keys:=keychar; // sinon nouvelle séquence

    //reset du chronomètre
    Chronometre:=Now;
    //Recherche de l'élément
    for I := 0 to Self.count-1 do
 //    if StartsText(Keys,Items[i]) then begin
      if ContainsText(Items[i],Keys) then begin
       Self.itemindex:=i;
       break;  //premier élément trouvé
      end;
  end;
  inherited;
end;
```

*L'utilisation des fonctions **MilliSecondsBetween, StartsTextWith** ou **ContainsText** implique l'ajout des unités **System.DateUtils** pour la première et **System.StrUtils** pour les autres.*

Avantage de cette démarche sa simplicité, mais elle n'est pas sans inconvénients :

- Du fait de la dérivation du composant de base tous les composants **TCombobox** sur la forme auront le même comportement ce qui n'est peut-être pas souhaité.
- La modification se fait à l'unité or une application n'en contient rarement qu'une seule.
- Le délai de frappe n'est pas modifiable.

- Les touches sont filtrées par la fonction **CharInSet** (chiffres, lettres majuscules et minuscules), ce qui vous a peut-être un peu choqué et qui ne fait pas la part belle à l'internationalisation du procédé. Une solution alternative serait de faire l'inverse et de filtrer les touches de clavier « spéciales », cela suppose, quand même, d'une bonne connaissance des divers claviers.

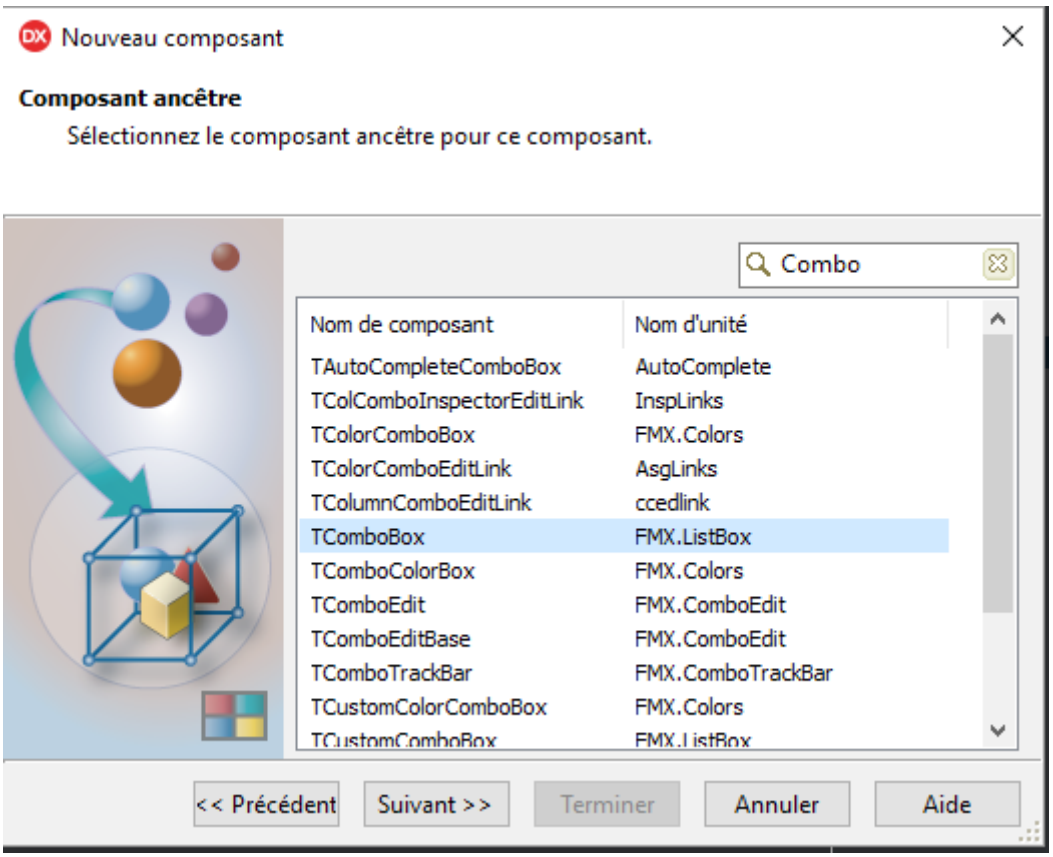
II-B - Deuxième solution : créer un nouveau composant

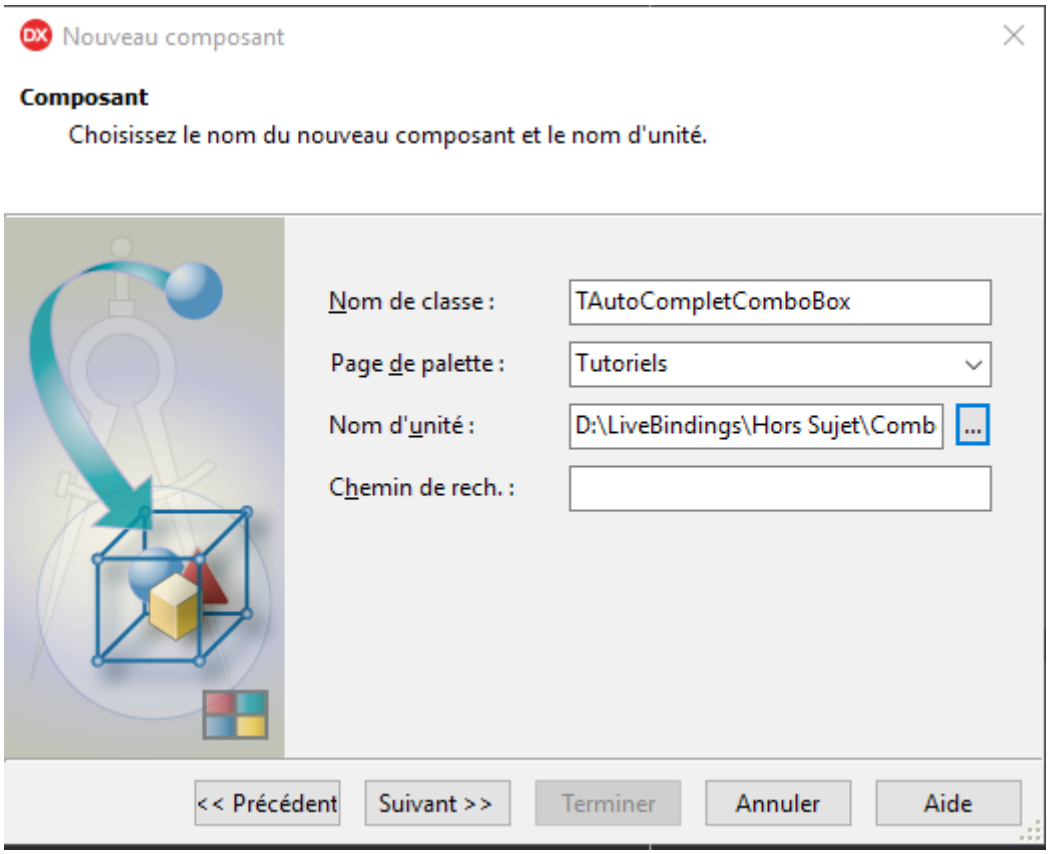
Dès que l'on parle de créer un nouveau composant, le doute, je n'ose écrire la peur, s'installe. Un débutant se dira que c'est compliqué, un programmeur confirmé y verra d'autres inconvénients, je cite : « Perte de temps », « casse-tête n'en valant pas la peine », etc.

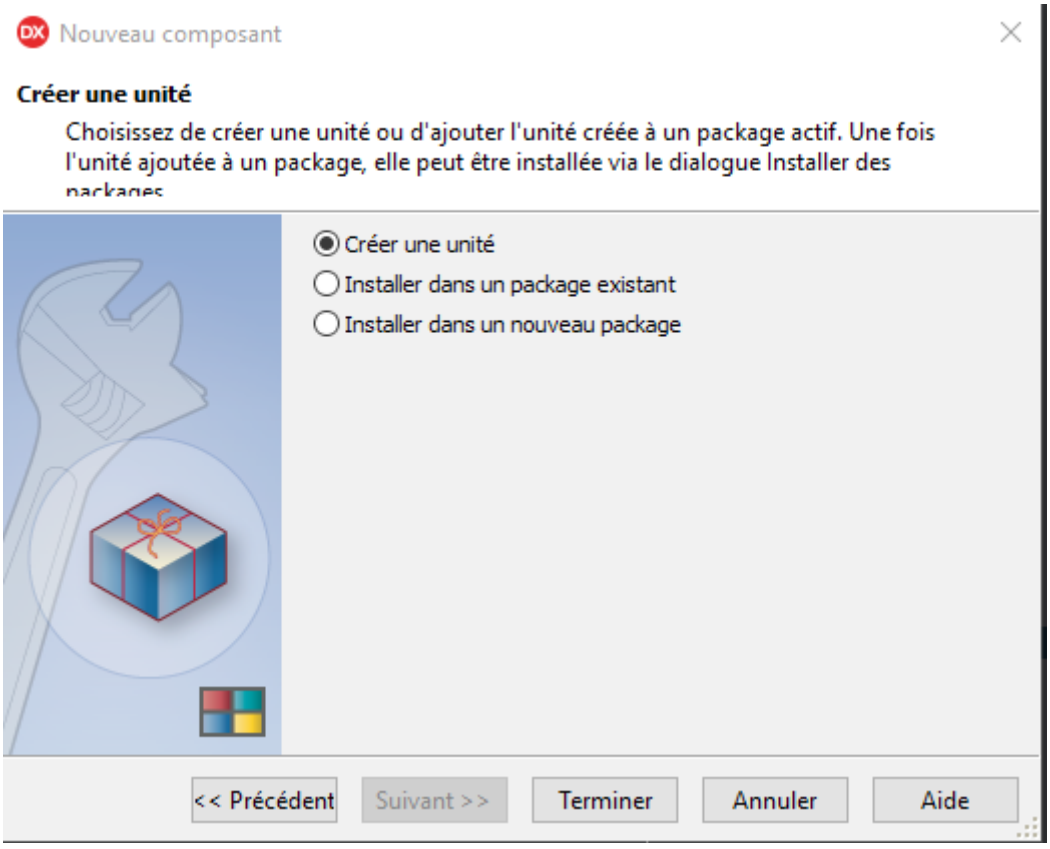
Si je ne peux rien répondre au programmeur confirmé, je peux au moins rassurer le débutant ; Non, ce n'est pas si compliqué que ça en à l'air et la première section a déjà bien débroussaillé l'horizon. En gros il s'agit de reprendre dans une unité le code indiqué au-dessus, de saupoudrer d'un peu de propriétés et d'ajouter l'enregistrement du composant dans l'IDE.

Comment faire sans stress ? Sans aucune hésitation en passant par l'assistant de création de composant (dans le menu : l'option 'Composant' puis 'Nouveau composant…').

*N'hésitez pas à consulter **la documentation***







En trois étapes j'obtiens une première unité squelette.

Avant d'aller plus avant, j'aimerais expliquer pourquoi j'ai préféré passer par la création d'un simple source plutôt que de demander à créer un nouveau package. Comme toute création de composant est à faire précautionneusement, je préfère déboguer mon code et le meilleur moyen pour cela est bien de créer le composant à l'exécution avant d'installer le composant. Je reporte donc à plus tard la création du paquet.

```
unit AutoComplete;

interface

uses
  System.SysUtils, System.Classes, FMX.Types, FMX.Controls, FMX.ListBox;

type
  TAutoCompleteComboBox = class(TComboBox)
  private
    { Déclarations privées }
  protected
    { Déclarations protégées }
  public
    { Déclarations publiques }
  published
    { Déclarations publiées }
  end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('Tutoriels', [TAutoCompleteComboBox]);
end;

end.
```

Après réflexion, je vais d'ores et déjà rajouter quelques propriétés. La réponse rapide que j'ai apporté mettait en lumière les variables privées ***Chronomometre*** et ***Keys***, le délai de frappe modifiable serait un plus, et enfin j'envisage également l'ajout du mode de recherche (début de chaîne ou contenu dans) sensible à la casse ou non.

Après ce recensement, j'insère donc les lignes de codes suivantes dans la partie privée.

```
private
  FChronometre: TDateTime;
  FKeys: string;
```

En préalable à la déclaration des nouvelles propriétés que je vais ajouter, je déclare un nouveau type pour définir les modes de recherche.

```
type
  TSearchMode = (smStarts,smContains);
```

Enfin, dans la partie publiée j'ajoute les lignes suivantes.

```
published
  property Delai: Integer read GetDelai write SetDelai default 500;
  property ModeRecherche : TSearchMode read FSearchMode write SetMode default TSearchMode.smStarts;
  property CasseSensible : Boolean read FCaseSensitive write SetCaseSensitive default false;
```

L'utilisation de la combinaison de touche Alt+C va me faciliter la tâche.

```
Après Alt+C
  unit AutoComplete;
```

Après Alt+C

```
interface

uses
  System.SysUtils, System.Classes, //  System.UITypes,
  FMX.Types, FMX.Controls, FMX.ListBox;

type
   TSearchMode = (smStarts,smContains);

   TAutoCompleteComboBox = class(TComboBox)
   private
     FChronometre: TDateTime;
     FKeys: string;
     FDelai: Integer;
     FSearchMode : TSearchMode;
     FCaseSensitive: Boolean;
     procedure SetMode(const Value: TSearchMode);
     procedure SetCaseSensitive(const Value: Boolean);
   protected
     procedure SetDelai(AValue: Integer);
     function GetDelai: Integer;
   public
   published
     property Delai: Integer read GetDelai write SetDelai default 500;
     property ModeRecherche : TSearchMode read FSearchMode write SetMode default
 TSearchMode.smStarts;
     property CasseSensible : Boolean read FCaseSensitive write SetCaseSensitive default false;
   end;

procedure Register;

implementation

procedure TAutoCompleteComboBox.SetCaseSensitive(const Value: Boolean);
begin
  FCaseSensitive := Value;
end;

procedure TAutoCompleteComboBox.SetDelai(AValue: Integer);
begin
 FDelai := AValue
end;

procedure TAutoCompleteComboBox.SetMode(const Value: TSearchMode);
begin
FSearchMode:=Value;
end;

function TAutoCompleteComboBox.GetDelai: Integer;
begin
   SetDelai(FDelai); // vérifier délai raisonnable  0 et 4 s
   Result := FDelai;
end;

procedure Register;
begin
   RegisterComponents('Tutoriels', [TAutoCompleteComboBox]);
end;

end.
```

Je vais maintenant ajouter le moteur de l'auto complétion, tel que déjà entre-aperçu dans la partie **II.A**, seule différence : prendre en compte mes différents modes de recherche.

Dans la partie protégée je rajoute la déclaration de la procédure ***KeyDown*** qui écrasera celle héritée.

protected

```
procedure KeyDown(var Key: Word; var KeyChar: System.WideChar; Shift: TShiftState); override;
```

Toutefois, pour alléger le code, je vais séparer la partie recherche dans une chaîne de la procédure et mettre celle-ci dans une fonction, renvoyant l'index de l'élément de liste trouvé.

public
```
function Recherche(AText: string; AShowDropDown: Boolean = True): Integer;
```

Pourquoi dans la partie publique ? Parce que, même si je n'en vois pas encore l'utilité, il me sera possible d'utiliser cette fonction à l'exécution.

J'utilise à nouveau la combinaison de touches Alt+C pour obtenir les lignes de code dans la partie ***implementation.***

```
function TAutoCompleteComboBox.Recherche(AText: string;
   AShowDropDown: Boolean = True): Integer;
begin

end;

procedure TAutoCompleteComboBox.KeyDown(var Key: Word; var KeyChar:System.WideChar;  Shift:
 TShiftState);
begin
  inherited;
end;
```

À partir de là c'est plus une question de codage qu'autre chose, voici donc ma vision du processus.

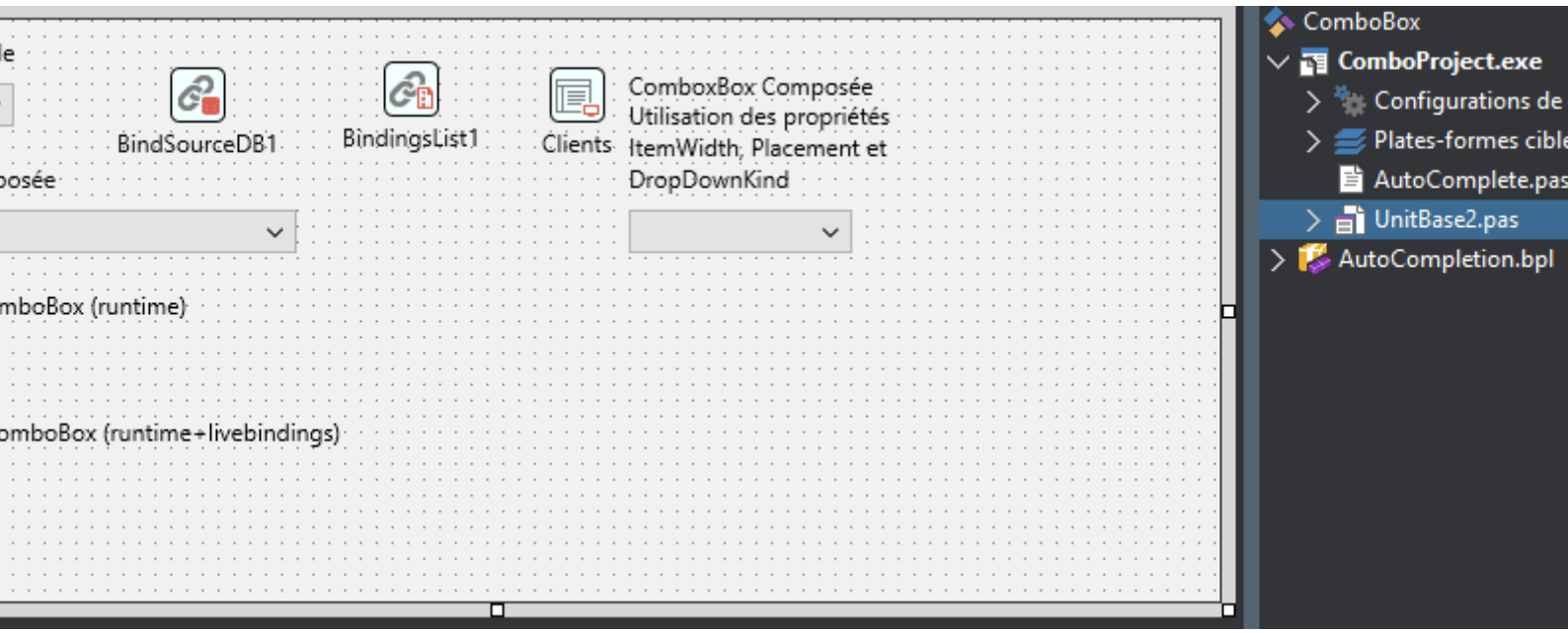
procedure KeyDown
```
procedure TAutoCompleteComboBox.KeyDown(var Key: Word; var KeyChar:System.WideChar;  Shift:
 TShiftState);
begin
   // test des caractères saisis
   if CharInSet(keychar,[chr(48)..chr(57),chr(65)..chr(90),chr(97)..chr(122)]) then
   begin
      // vérification du délai
      if MilliSecondsBetween(TimeOf(Now), FChronometre)<=FDelai
      then FKeys := FKeys + KeyChar
      else FKeys := KeyChar;
     // recherche du texte et positionnement dans la liste
     itemIndex:= Recherche(FKeys);
     // réinitialisation du chronomètre
     if ItemIndex<>-1 then
     begin
       FChronometre := TimeOf(Now);
     end;
   end
   else inherited;
end;
```

Fonction Recherche
```
function TAutoCompleteComboBox.Recherche(AText: string;
   AShowDropDown: Boolean = True): Integer;
var found : Boolean;
begin
 // au besoin, force l'affichage de la liste
 if (not Self.DroppedDown) and (AShowDropDown) then Self.DropDown;
 found :=False;
 // recherche dans la liste en fonction des options
 for result := 0 to Self.Items.Count - 1 do
   begin
    if FSearchMode=TSearchMode.smStarts then
      begin
      if FCaseSensitive
          then found:=StartsStr(FKeys, Self.Items[Result])
          else found:=StartsText(FKeys, Self.Items[Result]);
        end
```

Fonction Recherche

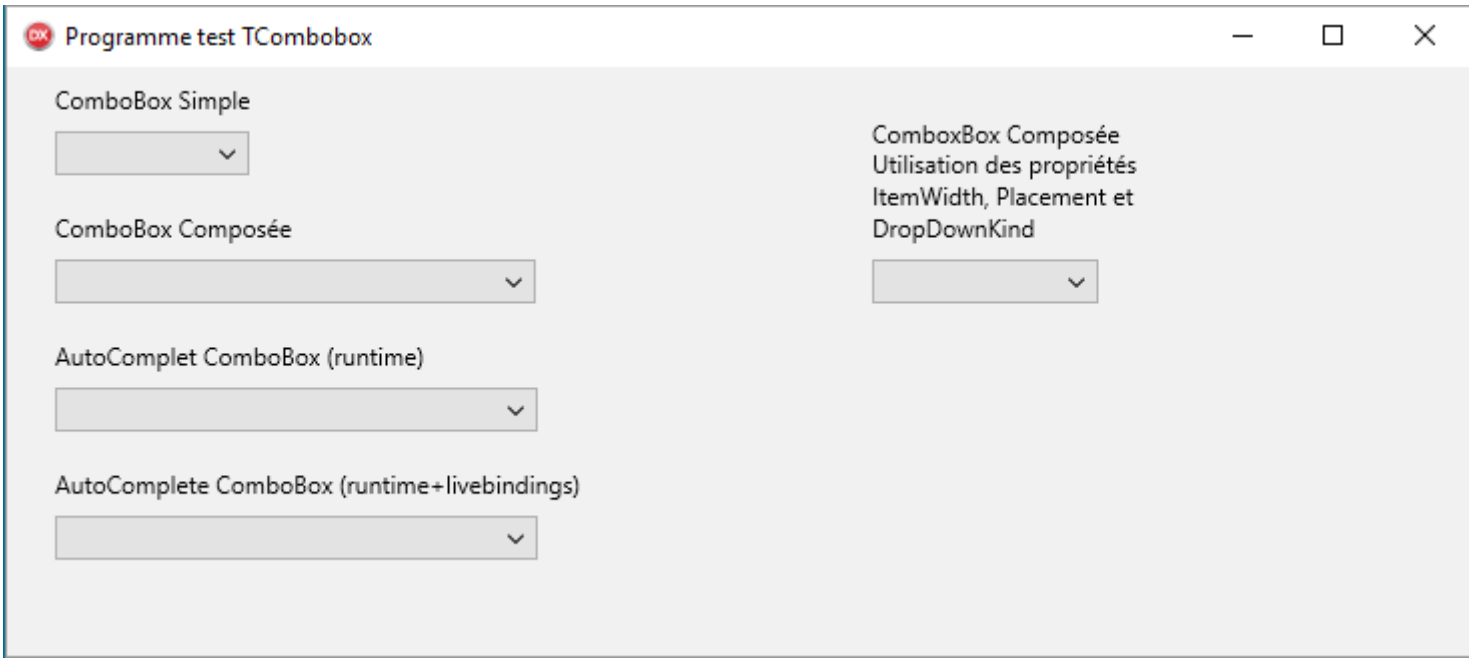
```
      else begin
        if FCaseSensitive
            then found:=ContainsStr(Self.Items[Result],FKeys)
            else found:=ContainsText(Self.Items[Result],FKeys);
         end;
      if found then Break;
    end;
  if not found then Result := -1;
end;
```

Une fois l'unité réalisée et sauvegardée, je passe aux tests. Il est temps que je dévoile mon petit programme de test que vous pourrez retrouver en téléchargement ici.



Design

Bien sûr au design, les deux boîtes que je vais tester ne vont pas apparaître.



Exécution

Quelques explications de code s'imposent, surtout en ce qui concerne le test avec un remplissage fait par l'intermédiaire des **LiveBindings**.

FormCreate

```
procedure TForm11.FormCreate(Sender: TObject);
var AAutoComplete,BAutoComplete : TAutoCompleteComboBox;
begin
// Création Composant AutoComplete Test
// remplissage par code
AAutoComplete:=TAutoCompleteComboBox.Create(Self);
with AAutoComplete do
 begin
   Parent:=Self;
   Position.X:=24;
   Position.Y:=160;
   Width:=241;
   CasseSensible:=False;
   Delai:=500;
   ModeRecherche:=TSearchMode.smContains;
   Clients.Active:=True;
   Clients.First;
   while not Clients.EOF do
     begin
       Items.Add(format('%.0f %s %s',[ClientsSS_NUMBER.asFloat,
                                      ClientsLAST_NAME.AsString,
                                      ClientsFIRST_NAME.asString]));
       Clients.Next;
     end;
   Clients.Active:=False;
 end;
// Simulation LiveBindings Test
BAutoComplete:=TAutoCompleteComboBox.Create(Self);
with BAutoComplete do
 begin
   Parent:=Self;
   Position.X:=24;
   Position.Y:=224;
   Width:=241;
   CasseSensible:=False;
   Delai:=500;
   ModeRecherche:=TSearchMode.smContains;
 end;
```

FormCreate

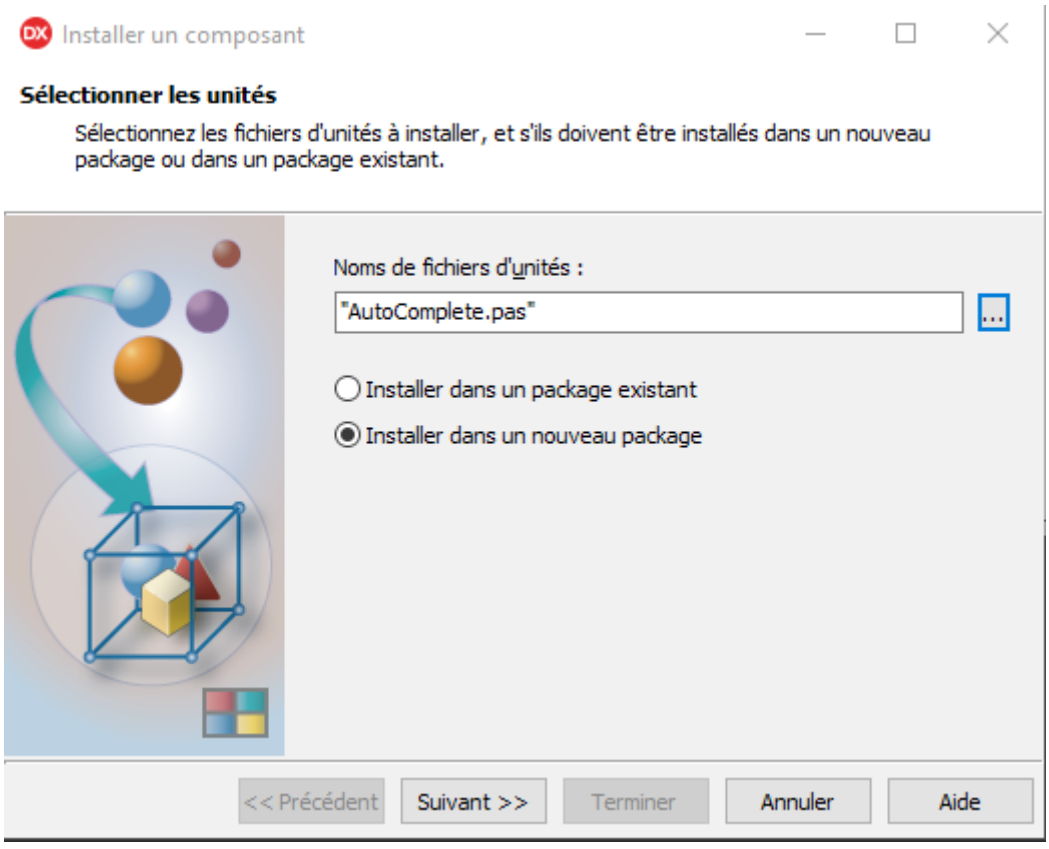
```
with TLinkFillControlToField.Create(Self) do // pas de synchronisation
// with TLinkListControltoField.Create(Self) do // avec synchronisation
  begin
       Control := BAutoComplete;
       Track := True;
       FillDataSource := BindSourceDB1;
       FillDisplayFieldName := 'SS_NUMBER';
       FillDisplayCustomFormat := '%s+'#39' '#39'+Dataset.LAST_NAME.Text+'#39'
 '#39'+Dataset.FIRST_NAME.text';
       AutoFill := True;
  end;
Clients.Active:=True;
end;
```

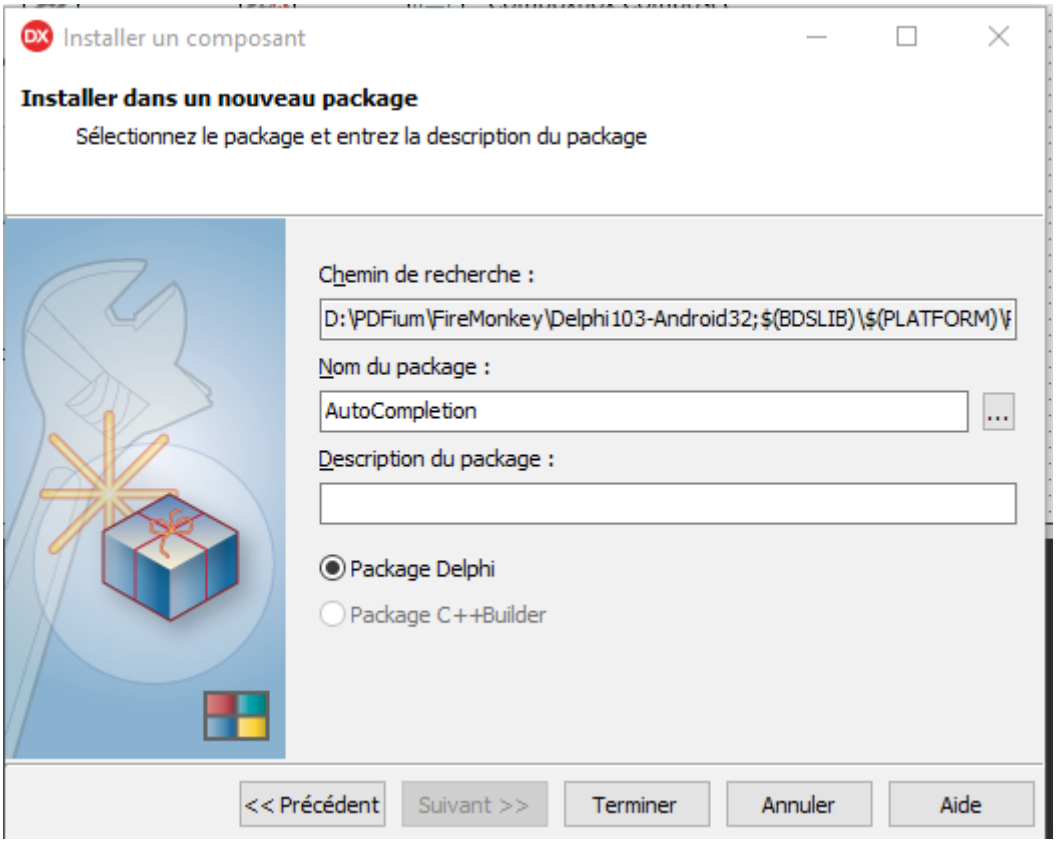
En effet tester la création d'un composant utilisant les **LiveBindings** interdit toute possibilité d'utiliser le concepteur visuel de liaison ou même l'ajout d'une liaison au **TBindingsList** au cours du design. J'utilise alors la technique exposée dans le tutoriel **LiveBindings et POO** pour créer le lien au cours de l'exécution.

Dernier point qui a son importance, vous remarquerez qu'après le remplissage par code du premier composant via un classique boucle, l'ensemble de données est refermé. Il est très important de le faire et de le ré-ouvrir après la création par programmation du lien entre la seconde boîte de recherche et les données.

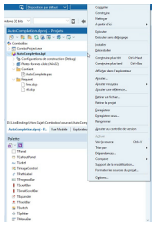
Cliquer sur ce lien pour lancer l'animation

Tests concluants, il ne me reste plus qu'à créer un nouveau paquet ou ajouter le source de mon composant dans un paquet existant et, objectif final, l'installer. Pour cela, je fais à nouveau appel à l'assistant de création de composant. Cette fois-ci j'utilise les options du menu : Composant/Installer un composant…





Suggestion d'amélioration du composant : travailler sur le filtrage de touches.



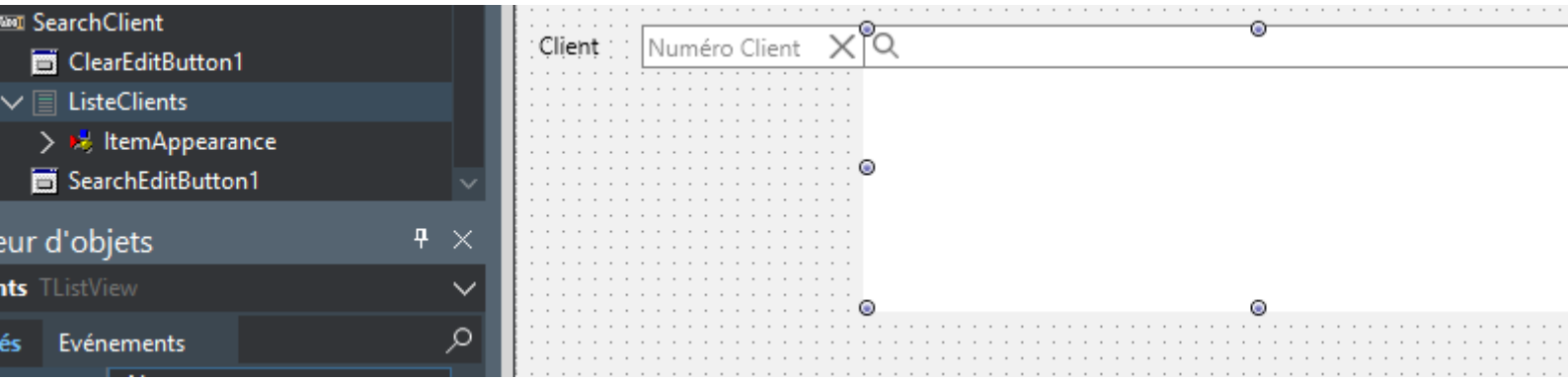
menu contextuel

III - Alternative

Toujours frileux ? Je vous propose une alternative qui ouvre de nombreuses perspectives tout en restant dans l'objectif premier : l'aide au choix.

L'origine de ce tutoriel est, en fait, **un billet de mon blog** où je déplorai les limites du **TComboBox.** Comme d'habitude, pressé par le temps, il me fallait trouver une alternative rapidement. En décomposant les fonctionnalités de la boîte de choix j'y trouve deux choses : une zone de saisie, quoique en lecture seule, et une liste. D'où l'idée d'associer un **TEdit** et un **TListView,** ce dernier ayant déjà fait l'objet de plusieurs communications de ma part aussi bien dans mon blog que via divers tutoriels.

Le design est donc simple, bien sûr la liste ne sera visible que si demandée.



Et le résultat au rendez-vous et ce avec quelques lignes de code seulement. Le remplissage de la liste, plus de 4000 éléments, se fait via Livebindings.

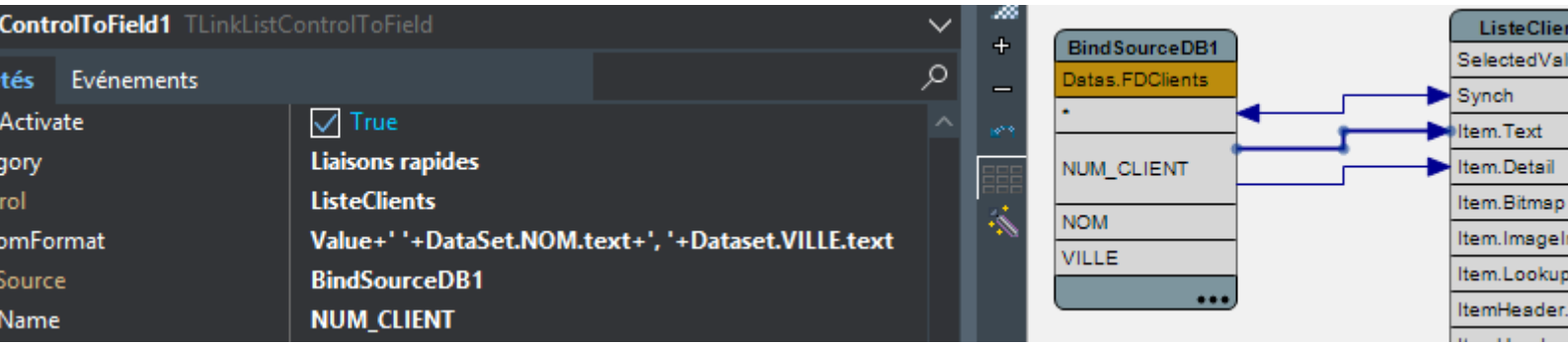
```
// Montrer la liste
procedure TForm1.SearchEditButton1Click(Sender: TObject);
begin
ListeClients.Visible:=True;
end;

// cacher la liste en cas de sortie (clic dans une autre zone de la forme
procedure TForm1.ListeClientsExit(Sender: TObject);
begin
ListeClients.Visible:=False;
end;

// gérer la sélection de l'élément
procedure TForm1.ListeClientsItemClick(const Sender: TObject;
  const AItem: TListViewItem);
begin
/// Récupération de la valeur
/// première solution, traitement du texte affiché
// SearchClient.Text:=LeftStr(AItem.Text,4);
/// deuxième solution, la valeur souhaitée est reportée, seule,
/// dans une zone de l'item de liste (cachée ou non)
// SearchClient.Text:=AItem.Detail;
/// troisième solution, rendu possible par la synchronisation
SearchClient.Text:=Datas.FDClientsNUM_CLIENT.AsString;
ListeClients.Visible:=False;
end;

// Abandon de la recherche par utilisation de la touche Escape
procedure TForm1.ListeClientsKeyDown(Sender: TObject; var Key: Word;
  var KeyChar: Char; Shift: TShiftState);
begin
if Key=vkEscape then
 begin
  ListeClients.Visible:=False;
  TEdit(ListeClients.Parent).SetFocus;
 end;
end;
procedure TForm1.ListeClientsExit(Sender: TObject);
begin
ListeClients.Visible:=False;
end;
```

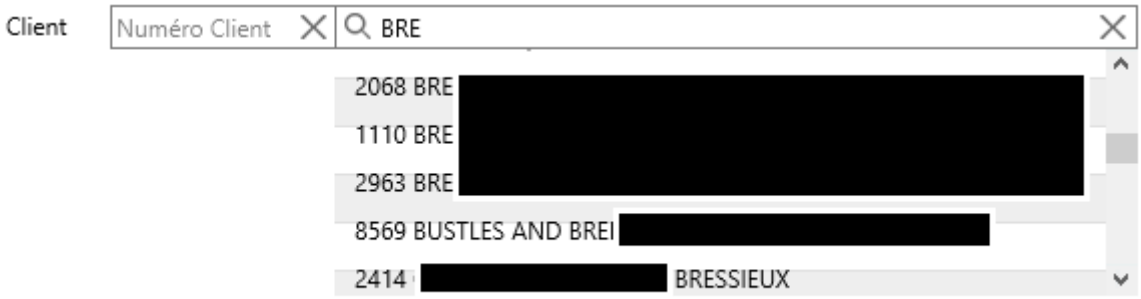
Toutefois, impossible d'intercepter l'utilisation la touche ***Escape*** à l'intérieur de la boîte de la recherche ! Pour pouvoir le faire il faudra, en premier lieu, indiquer l'utilisation de l'unité ***FMX.SearchBox***, puis associer l'évènement ***ListeClientsKeyDown*** à la boîte de recherche

(voir ce billet)

```
uses ... FMX.SearchBox;
```

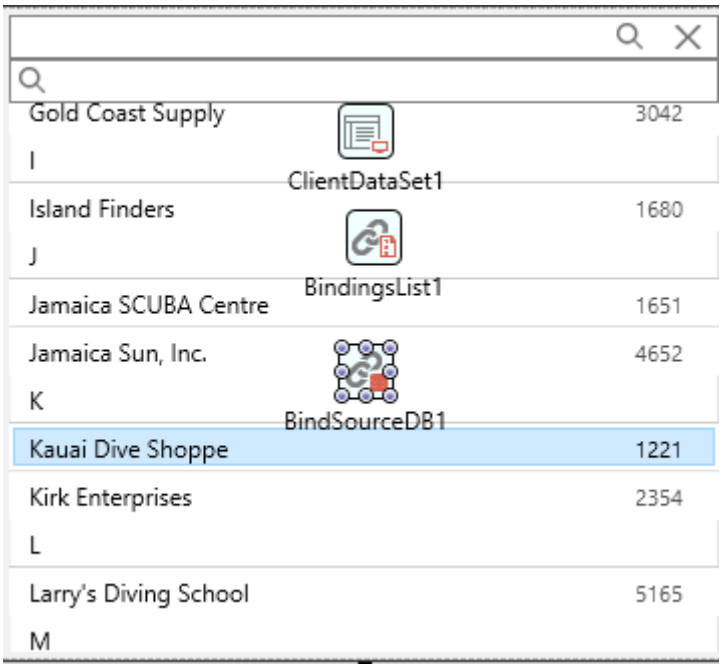
```
procedure TForm1.FormCreate(Sender: TObject);
begin
...
// association de l'évènement
if ListeClients.controls[1].ClassType = TSearchBox
  then  TSearchBox(ListeClients.controls[1]).OnKeyDown:=ListeClientsKeyDown;
end;
```





Vous excuserez la censure, l'image fournie ici est faite à partir de données d'entreprise.

L'utilisation de cette technique à un avantage certain, la liste peut être ainsi largement personnalisable par exemple en y ajoutant des groupes.



*Mettre ces deux composants ainsi qu'une partie du code dans un **cadre FireMonkey** pourrait également être une solution intéressante et permettrait de créer un pseudo-composant. Je vous invite à visionner le webinaire animé par **Patrick Prémartin** : **Créer des composants visuels sans faire de composant***

IV - Pour conclure

Je vous ai exposé trois méthodes pour améliorer l'utilisation de boîtes de choix. J'espère aussi avoir un peu vulgarisé la création de composant par héritage

IV-A - Références utilisées

Je ne pourrais citer toutes les questions posées dans les forums, ce qu'un moteur de recherche avec quelques mots clés pourra facilement vous ressortir. Je tiens cependant à signaler ces deux articles de Brovin Yaroslav : « Nouvelle approche de développement de contrôles FireMonkey 'Contrôle - Modèle – Présentation' » **Partie 1**, **Partie 2 TEdit avec auto complétion**

IV-B - Remerciements