Introduction à

Borland C++ Builder 6
Version professionnelle

Bruno-Laurent GARCIA
Loïc YON

Août 2006

Avertissement

Ce document a été rédigé avec l'environnement logiciel suivant :

- Windows XP SP2
- Borland C++ Builder 6 version professionnelle édition française, mise à jour 4
- Interbase 7.1

Quelques références bibliographiques

- Manuscrit original de Bruno-Laurent Garcia
- Le manuel du développeur, tomes 1 & 2
- Le guide de prise en main
- Différents sites Internet cités dans le document et le site institutionnel borland.com
- L'aide en ligne

Table des matières

Table des illustrations

1. C++ Builder : un environnement RAD fondé sur le C++

C++ Builder est un environnement de développement proposé par Borland fondé sur le C++. Pour cet environnement de développement, Borland a repris les recettes développées avec succès pour ses produits phares orienté Pascal : Delphi ; notamment l'interface et les bibliothèques de composants. La version 6 a vu le jour en 2002[1] et après de nombreuses mises à jour a été remplacée par une nouvelle version baptisée Borland C++ Builder 2006.

1.1 Philosophie

C++ Builder est un outil RAD ou *Rapid Application Development*, c'est-à-dire tourné vers le développement rapide d'applications sous Windows. C++ Builder permet de réaliser de façon très simple l'interface des applications et de relier aisément le code utilisateur aux événements Windows, quelle que soit leur origine (souris, clavier, système).

Pour ce faire, C++ Builder repose sur un ensemble très complet de composants (visuels et non visuels) prêts à l'emploi. La quasi-totalité des contrôles Windows (boutons, boîtes de saisie, listes déroulantes, menus, …) y est représentée. Les caractéristiques de ces composants sont éditables directement dans une fenêtre spéciale intitulée **inspecteur d'objets**. Un clic de souris sur les composants permet d'associer du code à l'objet.

La version 6 de Borland C++ propose deux hiérarchies de composants :

- La bibliothèque VCL ou *Visual Component Library* dédiée à Windows
- La bibliothèque CLX pour des composants multiplateformes aussi bien Windows qu'UNIX. La portabilité est assurée par l'encapsulation de la bibliothèque QT de Trolltech.

Leur racine est commune : elles dérivent toutes deux de **`TObject`** et les hiérarchies sont similaires (regroupement sémantiques). Elles sont toutes deux écrites en Pascal Objet. Si mes informations sont bonnes, la librairie CLX n'a pas survécu à la dernière mouture du Builder à cause notamment de nombreux bogues résiduels.

Il est possible d'ajouter à l'environnement de base des composants fournis par des sociétés tierces[2] et même d'en créer soi-même.

Un outil RAD, c'est également un ensemble de squelettes de projets qui permettent de créer plus facilement une application SDI ou MDI, une DLL, une application Console, etc. A chacun de ces squelettes est habituellement associé un expert qui par une série de boîtes de dialogues permet de fixer une partie des options essentielles à la réalisation du projet associé.

Le développement est facilité grâce au débogueur intégré et à l'aide omniprésente.

1 Pour mémoire, Microsoft sort le Service Pack 1 de Windows XP à la fin de l'année 2002.

2 JEDI Visual Component Library (http://homepages.borland.com/jedi/jvcl/)

1.2 Limitations

Tout d'abord, il faut savoir que la technologie RAD ne s'applique qu'au squelette ou à l'interface d'une application. Bien entendu, toute la partie spécifique à votre projet reste à votre charge.

Du point de vue de la portabilité, le code C++ de Builder n'est pas compatible C++ ANSI. Ceci est dû, entre autres, à la gestion des bibliothèques de composants et en particulier des propriétés et des closures. Rapidement, une propriété d'un objet est un moyen d'accéder directement à un attribut de l'objet. L'utilisation de cette propriété masque le fait que les méthodes de lecture/écriture de l'attribut (les fameuses méthodes set et et), ou **accesseurs**, sont appelées. Ce concept, créé initialement par Microsoft pour les composants OLE est très bien intégré en Delphi. Le mot-clé `__property` permet de définir une propriété et n'appartient pas au C++ normalisé. Tout code faisant appel à la VCL est de fait non portable.[3]

En revanche, tout code qui n'est pas directement lié à l'interface (et à VCL) peut être écrit en C++ ANSI. Plus que jamais, il convient de respecter la décomposition Interface/Données/Méthodes pour l'écriture d'une application C++ Builder. Pour de plus amples renseignements, on pourra se référer au chapitre 13 du guide du développeur concernant l'implémentation Pascal des composants VCL.

La VCL encapsule l'API Windows. Même s'il est toujours possible de l'utiliser directement, son utilisation peut être rendue plus complexe. Certains messages Windows sont aussi inaccessibles car cachés par des API de plus haut niveau.

L'environnement de développement Microsoft en C++ respecte un peu mieux le modèle de développement Document/Interface/Contrôleur même s'il semble plus difficile à prendre en main.

[3] Pour passer du code C++ Builder à un code ISO, on pourra lire l'article http://www.geocities.com/gengiskanhg.geo/GFDL/Borland_CPP_to_ISO_CPP_translation_guide.html

2. L'environnement de développement C++ Builder

Nous allons commencer la prise en main du produit en décrivant l'environnement de développement.

2.1 L'interface

La figure 1 représente un exemple typique de l'interface au cours d'une session de travail.

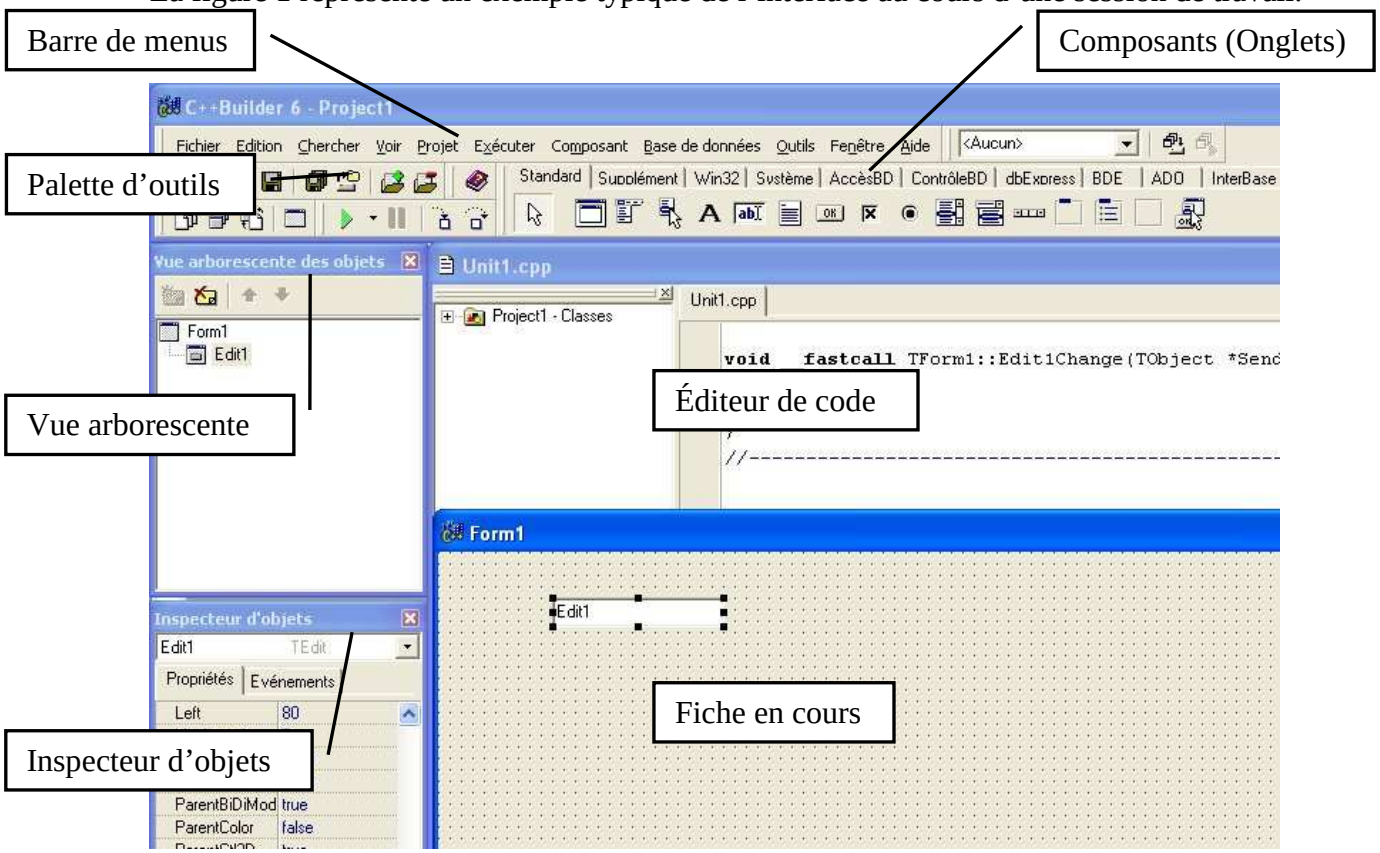


Figure 1 : Interface de C++ Builder

L'interface peut paraître déroutante car elle composée de différentes fenêtres qui ne recouvrent pas tout l'écran. Les applications qui ont été préalablement lancées sont toujours visibles.

On peut distinguer plusieurs zones distinctes :

- La barre de menu
- La barre d'outils qui se décompose elle-même en deux parties :
  - La palette d'outils pour les opérations classiques
  - La palette de composants rangés par catégories (onglets)
- Une fiche ou *Form* en Anglais qui représente l'interface en cours de création. Si l'application comporte plusieurs fiches, elles sont cachées et disponibles par le menu et le raccourci clavier F12. Du fait de l'environnement RAD, elles représentent à l'identique les composants à l'exécution (en dehors des composants non visuels).

- L'inspecteur d'objets qui donne les caractéristiques de l'objet sélectionné dans la fiche, tant au niveau des propriétés (attributs) que des événements.
- L'éditeur de code avec affichage automatique du code lié à l'objet sélectionné dans la fiche. A chaque fiche correspond deux fichiers : un fichier entête (.h) et un fichier code (.cpp) éditables.

D'autres fenêtres suivant le contexte peuvent être disponibles : arborescence des objets, débogueurs, experts.

2.2 Les composants de C++ Builder

Par défaut, C++ Builder utilise un compilateur C++, un éditeur de liens, un compilateur de ressources et un gestionnaire de projets intégrés.
Il est toutefois possible de spécifier la volonté d'utiliser les outils en ligne de commande livrés avec C++ Builder ou bien d'autres outils tiers. Cette dernière possibilité est très utile lorsque l'on veut utiliser des modules compilés avec d'autres langages. Il est fortement conseillé de lire les chapitres correspondants dans le manuel du développeur.

2.3 Création d'une application simple

C++ Builder permet de créer différents types de module très simplement en utilisant des experts. Pour créer un application toute simple, il suffit de sélectionner

(M) Fichier > Nouveau > Application

Un nouveau projet est alors créé. Ce projet contient une nouvelle fiche : Form1 et deux fichiers sources Unit1.cpp et Unit1.h. La terminologie est empruntée au Delphi et tous les éléments sont bien sûr renommables.

Il est très fortement recommandé de sauvegarder ce nouveau projet après sa création avec la commande

(M) Fichier > Enregistrer le projet sous

Le projet en lui-même est sauvegardé (fichier .bpr) après les différents fichiers *.cpp et*.h.

Si l'on regarde les différents fichiers créés par cette manipulation, on s'aperçoit que C++ Builder a créé automatiquement de nombreux autres fichiers :

- .ddp :
- .dfm : fichier texte décrivant les objets des fiches
- .res : fichier compilé de ressources

2.4 Création d'une application console

Une application console est une application un peu particulière : une telle application s'exécute et s'affiche dans une invite de commande sans interface graphique utilisateur.

(M) Fichier > Autre (O) Nouveau > Expert Console

Il est cependant possible d'utiliser les composants de la VCL ou de la CLX. Ce choix doit être fait à la création du projet sous peine d'obtenir des erreurs à la compilation. Les applications console doivent gérer toutes les exceptions afin que des boîtes de dialogue ne s'affichent.

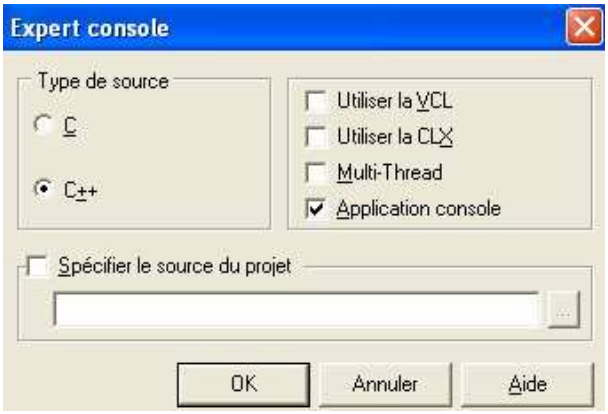


Figure 2 : Options de l'expert Console

Les applications console nous sont utiles pour découvrir le langage C++ en écrivant des exemples qui ne nécessitent pas d'interface graphique.

Il ne faut pas oublier que les fonctions standards sont dans un espace de nommage particulier **`std`** et qu'il est nécessaire de le préciser soit au travers de la clause **`using namespace std`** , soit accolé à chacune des instructions de l'espace, par exemple **`std ::cout`**.

2.5 L'inspecteur d'objets et les propriétés

L'inspecteur d'objets est une fenêtre à deux volets respectivement spécialisés dans l'édition des valeurs des propriétés des composants et l'intendance de leurs gestionnaires d'événements.

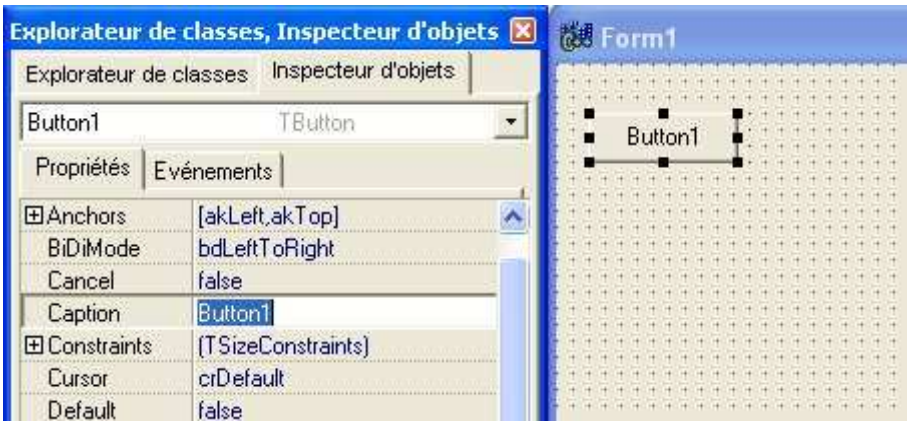


Figure 3 : Edition des propriétés dans l'inspecteur d'objets

La figure ci-dessus montre l'aspect de l'inspecteur d'objets lors de la modification des propriétés d'un bouton.
Selon le type de la propriété, l'édition se fera sous forme d'une boîte d'édition de texte simple, dans une liste à dérouler ou même une fenêtre spécialisée.

L'inspecteur d'objets peut être déplacé à l'écran et inséré par exemple dans l'éditeur de code. Si vous le perdez, appuyez sur la touche F11 pour le retrouver.

2.6 La propriété Name

Cette propriété est primordiale dans la mesure où elle permet d'accéder aux composants à l'intérieur du programme. Par défaut, lorsque l'on ajoute un nouveau composant à une fiche, C++ Builder lui confère un nom automatique du genre `TypeNuméro` qui est peu explicite. Par exemple, à la figure précédente, le premier bouton créé s'appelle `Button1`.

Il est préférable de respecter certaines conventions de nommage pour ses composants. Par exemple, un nom de composant doit indiquer :

- Le type du composant
- la fonction

L'usage veut qu'en général, la fonction se retrouve dans le nom du composant et que le type soit indiqué par un préfixe. Voici quelques exemples à titre informatif :

- `b`/`bn` : bouton d'action
- `br` : bouton radio
- `cc` : case à cocher
- `pl` : panel
- `lb` : label
- `gb` : boîte de groupe
- `edit` : boîte d'édition
- `me` : mémo

Ces conventions peuvent varier en fonction de la charte de développement de l'entreprise.

2.7 Manipuler des événements

Voici un exemple de manipulation d'événements au travers de l'inspecteur d'objets :

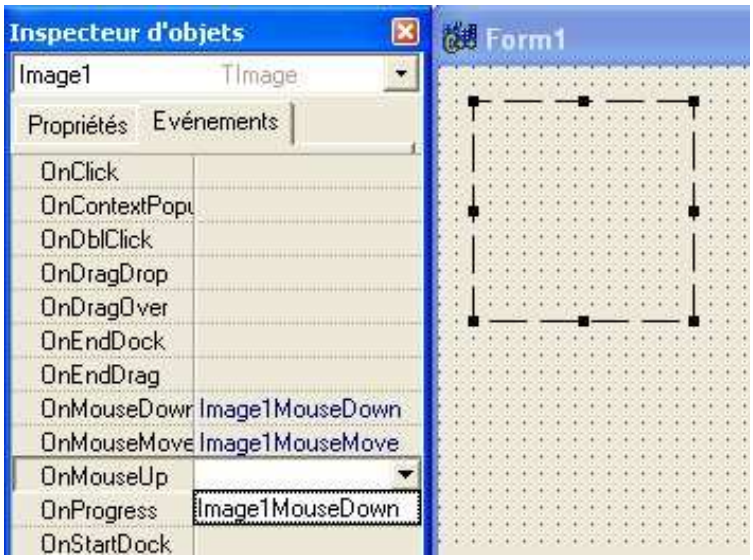


Figure 4 : Inspecteurs d'objet et événments

Les gestionnaires d'événements sont toujours des méthodes de la fiche. En effet, avec C++ Builder, les événements générés par les contrôles sont toujours renvoyés vers la fiche. Ce mécanisme (connu sous le nom de notification au parent) est très pratique car il permet de simplifier considérablement la tâche du programmeur : tous les événements de la fiche sont générés au même niveau.

Néanmoins, il est parfois gênant. Prenons l'exemple de la gestion d'un groupe de boutons radios. Typiquement, les boutons seront regroupés dans une fenêtre de groupe. Avec la version précédente de Borland C++, il était possible de rediriger les événements générés par les boutons radio vers la fenêtre de groupe et de les traiter ainsi avec une seule méthode. Désormais tous les événements étant automatiquement redescendus au niveau de la fiche, il faudra gérer individuellement les événements ou ruser comme un sioux.

Une même méthode peut gérer plusieurs événements si son prototype le permet.

Notons que la liste de paramètres d'un gestionnaire d'événements contient toujours au moins un paramètre nommé Sender, de type **`TObject*`** et qui contient l'adresse du composant ayant généré le message. D'autres paramètres peuvent être présents, par exemple :

- Les positions de la souris
- L'état des touches de modification du clavier

Pour créer une nouvelle méthode de gestion d'événement, il suffit de double cliquer dans l'espace vide à droite du nom de l'événement, une méthode avec le nom par défaut est alors créée. Son appellation reprend en partie le nom ou le type du contrôle générant l'événement et la dénomination de l'événement. Vous pouvez également choisir le nom vous même.

Pour affecter une méthode déjà existante à un gestionnaire, il suffit de puiser dans la liste déroulante.

2.8 C++ Builder et les exceptions

Le comportement de C++ Builder vis à vis des exceptions peut paraître parfois déroutant. En effet, la mise en place d'un gestionnaire d'exceptions, par exemple, pour l'exception **`EConvertError`** se traduit d'abord par l'affichage d'un message du genre :

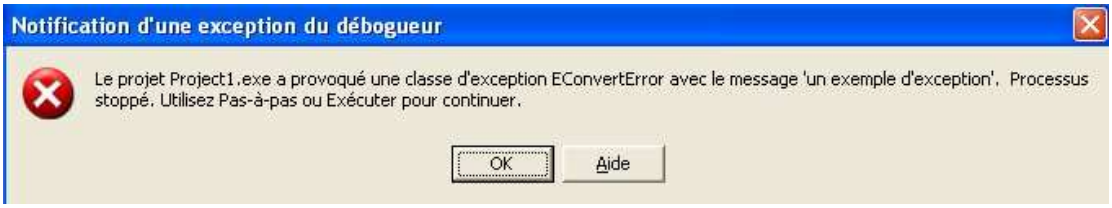


Figure 5 : Fenêtre d'interception d'une exception

L'exception a été provoquée par l'instruction

```
throw EConvertError("un exemple d'exception");
```

Ce qui se traduit en clair par la phrase suivante : Le débogueur intégré à C++ Builder a intercepté l'interruption avant de vous passer la main. Notez au passage que la fenêtre contient le message d'explication inclus dans toute exception C++ Builder : il s'agit du texte entre guillemets.

Le plus important est de savoir que ce comportement n'existe que dans l'EDI. En effet, en exécution indépendante, ce message n'apparaîtrait que si l'exception déclenchée n'était pas traitée ; si vous fournissez un handler d'exception, celui-ci serait activé normalement. En mode Console, toutes les exceptions doivent être traitées pour ne pas voir apparaître une telle boîte de dialogue.

Dans de nombreux cas, ce fonctionnement de l'EDI est plus une gène qu'un atout. Il est toutefois possible de le désactiver grâce à la boîte de dialogue – présentée ci-dessous – directement issue du menu (M) Outils > Options du débogueur. Le comportement par défaut semble désormais être la désactivation de cette possibilité.

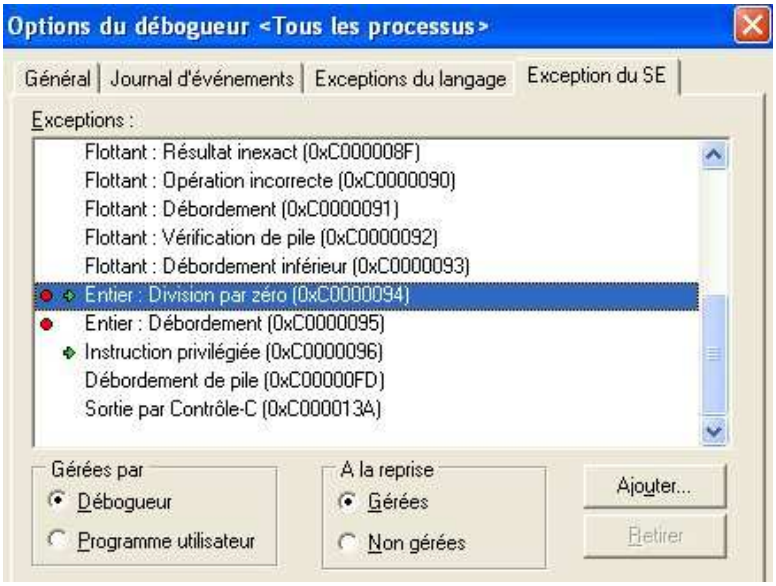


Figure 6 : Options du débogueur de l'EDI de C++ Builder

En plus de fournir de nombreuses options de configuration du débogueur, cette page nous permet de spécifier le comportement des exceptions lorsqu'un programme C++ Builder est lancé depuis l'EDI.

Ainsi, toutes les catégories marquées d'un petit rond (de couleur rouge !) – voir la figure précédente – sont gérées, d'abord par l'EDI et dans un second temps par le programmeur. Il est très facile de modifier ce comportement en sélectionnant le bouton radio « Programme utilisateur ».

En outre, si vous laissez l'EDI gérer votre interruption, il est possible de spécifier le comportement de votre programme à la sortie de la fenêtre de l'EDI en jouant sur les boutons radio « A la reprise ».

Par défaut, les exceptions de l'utilisateur sont rangées dans la rubrique « Exceptions C++ » alors que les exceptions levées par la VCL sont dans la catégorie « Exceptions Delphi » **et** dans la catégorie « Exceptions C++ ».

2.9 Utiliser le journal d'événements

La fenêtre du journal des événements est l'un des mécanismes de traçage des programmes les plus méconnus de Windows. C++ Builder nous permet de consulter l'état de cette fenêtre en l'activant dans le menu (M) Voir > Fenêtres de débogage > Journal d'événements.

La page de configuration des options du débogueur permet également de modifier le comportement de cette fenêtre. Focalisons nous donc sur ce dernier aspect :

Figure 7 : Options du journal d'événements

Deux grands types d'options sont gérables

- Les informations générales avec notamment la taille de l'historique
- Les catégories de messages tracés
    - Les points d'arrêt : à chaque fois que l'exécution d'un programme est stoppée à un point d'arrêt, un message est imprimé dans la fenêtre
    - Les messages de processus : ils concernent par exemple le chargement de DLL ou des informations de débogage au lancement du programme
    - Les messages de threads
    - Les messages de sorties : ils sont créés par l'utilisateur à l'aide de la commande API **`OutputDebugString`**. Le seul problème tient à la bufferisation de cette fenêtre. En effet, les messages envoyés par **`OutputDebugString`** ne sont pas affichés immédiatement, ce qui limite l'intérêt de cette fonctionnalité. En outre, cette fonction prend comme paramètre un **`char *`** et non pas une **`AnsiString`** comme la plupart des méthodes et fonctions de la VCL, ce qui rend le transtypage nécessaire.
    - Les messages de fenêtres : à chaque fois qu'une fenêtre reçoit un message, il y a un message contenant :
        - Le récepteur du message
        - Les paramètres (WPARAM et LPARAM)

Attention, si ce comportement peut se révéler très pratique, il peut créer des traces énormes. En effet, on imagine rarement le volume considérable de messages traités par les applications Windows.

3. Étude de la VCL

3.1 Organisation de la VCL

La VCL ou *Visual Component Library* livrée par Borland avec le C++ Builder ou Delphi est un ensemble de classes orientées vers le développement rapide d'application dédiée au system d'exploitation Windows. Toutes les classes présentes partagent un ancêtre commun : la classe **`TObject`**. Elles possèdent également une caractéristique particulière : elles ne peuvent pas posséder d'instances statiques : seules les instances dynamiques créées avec **`new`** sont acceptées. Ceci est nécessaire pour assurer la compatibilité avec Delphi qui ne reconnaît que les instances dynamiques. Toutes les classes de la VCL sont implémentées en Pascal Objet. En fait, la librairie d'exécution de C++ Builder est celle de Delphi, ce qui implique un certain nombre de gymnastiques pour assurer une édition de liens correcte.

3.2 Les composants

Les composants sont des instances de classes dérivant plus ou moins directement de **`TComponent`**. Si leur forme la plus habituelle est celle des composants que l'on dépose d'une palette vers une fiche, ils englobent plus généralement la notion de brique logicielle réutilisable. Bien qu'elle ne soit pas déclarée virtuelle pure[4] la classe **`TComponent`** n'est pas destinée à être instanciée. Compulsons sa documentation ;nous y apprenons que la plupart des méthodes sont protégées, c'est à dire inaccessibles à l'utilisateur, c'est une technique courante en Pascal Orienté Objet : définir le cadre de travail à l'aide de méthodes virtuelles protégées, lesquelles seront déclarées publiques dans les classes dérivées.

Autre aspect particulièrement intéressant : la présence des méthodes **`AddRef`**, **`Release`** et **`QueryInterface`** ce qui dénote l'implémentation de l'interface OLE **`IUnknown`**. Ainsi, lorsque l'on transformera un composant VCL en composant ActiveX, la gestion **`d'IUnknown`** sera directement prise en compte au niveau de **`TComponent`**. De la même manière, **`TComponent`** implémente l'interface principale d'automation **`IDispatch`** (méthodes **`GetIDsOfNames`**, **`GetTypeInfo`**, **`GetTypeInfoCount`** et **`Invoke`**). Pour plus de renseignements, se référer au guide du développeur concernant l'utilisation des technologies Microsoft OLE.

Pour finir, notons que les composants que vous créez avec Delphi ou C++ Builder sont compatibles avec les deux environnements : autrement dit, il est tout à fait possible d'utiliser dans C++ Builder un composant créé avec Delphi et réciproquement.

[4] En effet, la classe **`TComponent`**, à l'instar de toutes les autres classes de la VCL est implémentée en langage Pascal Objet, lequel s'accommode assez mal de la notion de classe virtuelle pure.

3.3 Les contrôles

On appellera Contrôle, tout objet instance d'une classe dérivant de **`TControl`**. Les contrôles ont pour caractéristique particulière d'être des composants à même de **s'afficher** sur une fiche dans leurs dimensions d'exécution. Par exemple, les boutons (**`TButton`**), les étiquettes (**`TLabel`**) ou les images sont des contrôles. En revanche, les menus (**`TMenu`**) ou les boîtes de dialogue communes (**`TCommonDialog`**) de Windows qui sont représentées par une icône sur les fiches ne sont pas des contrôles.

En terminologie C++ Builder, les contrôles se séparent en deux grandes catégories :

- les contrôles fenêtrés
- les contrôles graphiques

3.3.1 Les contrôles fenêtrés

Comme leur nom l'indique, les contrôles fenêtrés sont fondés sur une fenêtre Windows. Ceci leur confère plusieurs caractéristiques :

- Ils disposent d'un **`handle`** de fenêtre. Un **`handle`** est un numéro unique alloué par le système à toute ressource telle que les fenêtres, les polices, les pinceaux ou les brosses.
- De ce fait, chaque contrôle fenêtré monopolise une ressource fenêtre : il ne pourra y en avoir qu'un nombre limité présent dans le système à un instant donné.
- Leur état visuel est sauvegardé par le système. Lorsqu'ils redeviennent visibles, leur apparence est restaurée automatiquement sans que le programmeur n'ait à s'en soucier. Conséquence négative : il est plus lent de dessiner dans un contrôle fenêtré que dans un contrôle graphique.
- Ils peuvent recevoir la focalisation, c'est à dire intercepter des événements en provenance du clavier
- Ils peuvent contenir d'autres contrôles. Par exemple, les boîtes de groupe (**`TGroupBox`**) sont des contrôles fenêtrés.
- Ils dérivent de la classe **`TWinControl`** et, la plupart du temps, de la classe **`TCustomControl`** laquelle dispose d'une propriété **`Canvas`** permettant de dessiner facilement dans la zone client du contrôle. Si vous devez dessiner dans la zone client d'un contrôle sans que celui-ci nécessite la focalisation, alors, il faudra mieux utiliser un contrôle graphique moins gourmand en ressources système.

La plupart des éléments actifs dans une interface sont des contrôles fenêtrés. En particulier, les fiches sont des contrôles fenêtrés.

3.3.2 Les contrôles graphiques

Contrairement aux contrôles fenêtrés, les contrôles graphiques ne s'appuient pas sur une fenêtre. Ils ne disposent donc pas d'un **`handle`** de fenêtre et ne peuvent recevoir d'événements en provenance du clavier (pas de *focalisation*). Ils ne peuvent pas non plus contenir d'autres contrôles, toutes ces fonctionnalités étant réservées aux contrôles fenêtrés

(voir paragraphe précédent). Du coup, ils sont moins gourmands en ressources que les composants fenêtrés.

En outre, l'état visuel (ou apparence graphique) d'un contrôle graphique n'est pas gérée par le système : il sera plus rapide de dessiner dans un contrôle graphique que dans un contrôle fenêtré car dans ce cas, seul l'affichage est concerné. Il n'y a pas de sauvegarde de l'état dans la mémoire du système.

Lorsqu'un contrôle graphique est masqué puis réaffiché, le système lui envoie un événement **`WM_PAINT`** lui indiquant qu'il doit mettre à jour son apparence visuelle. Le programmeur doit donc intercepter cet événement (méthode **`OnPaint()`**) pour redessiner la zone client de son composant. Pour cela, il dispose de la propriété **`Canvas`** qui fournit une plate-forme de dessin des plus agréables à utiliser. Il faut également savoir que le gestionnaire d'événements **`OnPaint()`** est appelé par la méthode virtuelle **`Paint()`** directement déclenchée par l'événement Windows **`WM_PAINT`**. De fait, la création d'un nouveau contrôle graphique passe le plus souvent par la redéfinition de cette méthode.

La plupart des composants purement orientés vers l'affichage sont des contrôles graphiques. Citons par exemple **`TLabel`** (affichage pur et simple d'un texte), **`TImage`** (affichage d'un graphique) ou **`TBevel`** (affichage d'une ligne, d'une forme en creux ou en relief).

3.4 Les boîtes de dialogue standards de Windows

Les boîtes de dialogue standards de Windows sont des objets partagés par toutes les applications et permettant d'effectuer des opérations de routine telles que la sélection d'un nom de fichier, la configuration de l'imprimante ou le choix d'une couleur. C++ Builder encapsule la plupart d'entre elles dans des classes non visuelles dont les composants sont regroupés au sein de la palette « Dialogues ». Les classes associées dérivent de **`TCommonDialog`**, classe servant à établir le lien entre les classes de C++ Builder et les ressources Windows incluses dans la DLL **`COMMONDLG.DLL`**.

Ces composants ne sont pas des contrôles : ils n'apparaissent pas sous leur forme définitive lorsqu'ils sont insérés sur une fiche mais plutôt au travers d'une icône. Cette dernière n'apparaît pas lors de l'exécution, au contraire de la boîte qui elle sera invoquée à l'aide de sa méthode **`Execute()`**. L'icône posée sur la fiche ne sert qu'à réserver une ressource Windows et permet de modifier les propriétés associées à la boîte. La figure suivante illustre la palette dialogues :

Les différentes boîtes de dialogue disponibles sont, de gauche à droite : **`TOpenDialog`**, **`TSaveDialog`**, **`TOpenPictureDialog`**, **`TSavePictureDialog`**, **`TFontDialog`**, **`ColorDialog`**, **`TPrintDialog`**, **`TPrinterSetupDialog`**, **`TFindDialog`**, **`TReplaceDialog`**.

Mis à part **`TFindDialog`** et **`TReplaceDialog`**, ces boîtes de dialogue sont toutes modales. Cela signifie qu'elles monopolisent la souris et le clavier tant que l'utilisateur ne les a pas fermées. La variable **`TModalResult`** ou la méthode **`execute()`** indiquent à l'invocateur avec quel bouton l'utilisateur à fermé la boîte. Les constantes (aux noms très explicites) sont les suivantes : **`mrNone`**, **`mrOk`**, **`mrRetry`**, **`mrCancel`**, **`mrIgnore`**, **`mrNo`**, **`mrYes`**, **`mrAbort`**, **`mrAll`**, **`mrNoToAll`**, **`mrYesToAll`**.

Insistons sur le fait que l'utilisation de ces classes dépend fortement de la DLL standard de Windows **`COMMDLG.DLL`**, leur aspect et leur comportement peuvent donc varier légèrement d'un système à un autre.

3.4.1 Les boîtes de dialogue de manipulation de fichiers

Il est particulièrement pratique d'utiliser les boîtes de dialogue communes de Windows pour récupérer le nom d'un fichier à ouvrir ou à enregistrer car elles permettent, en particulier, une gestion intégrée de la navigation dans l'arborescence des répertoires. Les classes **`TOpenDialog`** et **`TSaveDialog`** sont très similaires dans le sens où elles reposent sur les mêmes propriétés. Elles ont d'ailleurs une structure de données identique. Voici un exemple de boîte d'ouverture :

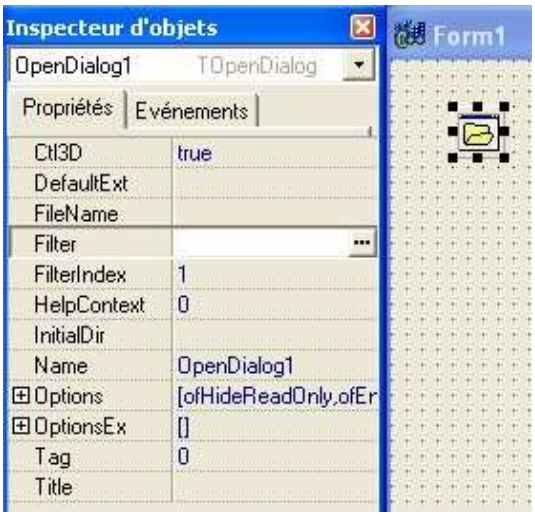


Figure 8 : Exemple d'une boîte d'ouverture de fichier

Quelques propriétés méritent l'attention :

- **`FileName`** contient, avant l'exécution, le nom du fichier à ouvrir par défaut et au retour de l'exécution, le nom résultat de la recherche.
- **`Filter`** contient la liste des formats de fichiers (spécifiés par leurs extensions) que l'application est susceptible d'ouvrir. En l'absence de toute indication, on suppose que tous les fichiers (*.*) sont ouvrables.
- **`InitialDir`** est le répertoire initial à l'ouverture
- **`Title`** est le titre de la boîte de dialogue

La chaîne des filtres a un format très particulier :

Chaîne de description | liste des extensions associées | description | extensions | ….
Fichiers texte|*.txt|Tous les fichiers (*.*)|*.*

En cliquant sur les trois petits points, une boîte spéciale apparaît qui facilite la saisie des filtres. Avec l'exemple précédent, voici la boîte que l'on obtient :

Figure 9 : Boîte d'édition de la propriété Filter

Les deux boîtes (ouverture et fermeture) se distinguent essentiellement par les options qui sont disponibles (précisément dans la propriété **`Options`**). Par exemple, la boîte d'ouverture (**`TOpenDialog`**) peut présenter une case à cocher proposant d'ouvrir un fichier en lecture seulement.
Il existe deux versions spécialisées dans l'ouverture et la sauvegarde de fichiers image qui ne posent pas de problème spécifique.

3.4.2 La boîte de sélection de couleurs

La figure suivante montre les propriétés les plus intéressantes de la boîte de sélection de couleurs de C++ Builder (classe **`TColorDialog`**).

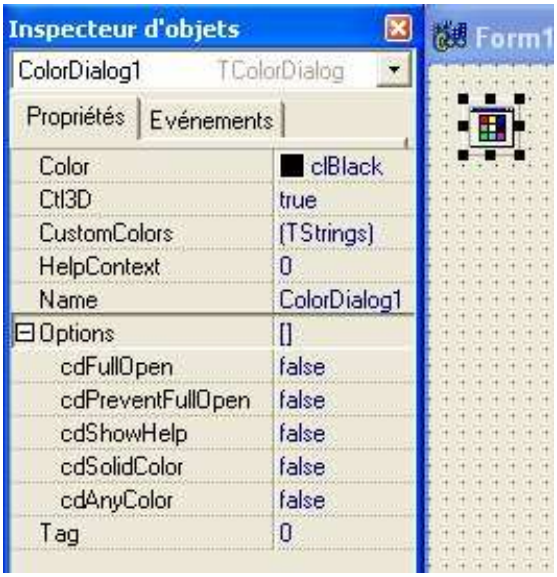


Figure 10 : Propriété de la boîte de choix de couleur

Certaines propriétés méritent que l'on s'y intéresse

- **`Color`** qui représente la couleur initiale, puis la couleur choisie par l'utilisateur
- **`CustomColors`** qui contient les couleurs utilisateur

Les options sont particulièrement intéressantes car elles conditionnent toute l'exécution : on peut par exemple proposer à l'utilisateur de créer ses propres couleurs :

- **`cdFullOpen`** permet d'ouvrir directement la boîte de dialogue en mode complet. Sinon la boîte s'ouvre en mode standard.
- **`cdPreventFullOpen`**. Lorsque cette option est activée, il est impossible de passer en mode complet.
- **`cdShowHelp`**. Si cette option est activée, un bouton permettant d'activer un texte d'aide est présent.

- **`cdSolidColor`** et **`cdAnyColor`** sont deux options qu'il est intéressant d'activer si la carte vidéo de l'utilisateur ne supporte pas le mode *true colors*. Lorsque **`cdSolidColor`** est activée, la sélection d'une couleur obtenue par tramage renvoie vers la vraie couleur la plus proche. Si **`cdAnyColor`** est activée, alors il est possible de créer de nouvelles couleurs par tramage.

3.4.3 La boîte de sélection de polices de caractères (fonte)

Cette boîte, très simple, permet de sélectionner une police parmi celles installées dans Windows. Toutes les informations sélectionnées sont regroupées dans la propriété **`Font`** qui elle même peut se déplier pour donner des informations aussi diverses que : le nom de la police, le jeu de caractères employé, le crénage, le style, la couleur ou, tout simplement, la hauteur des caractères.

La figure suivante montre la propriété **`Font`** déployée au maximum. On aperçoit le nom de la police (**`Name`**), la taille des caractères (**`Size`**), l'enrichissement de style (**`Style`**). Une boîte de choix standard de Windows permet de changer la police par défaut (en cliquant sur les trois points)

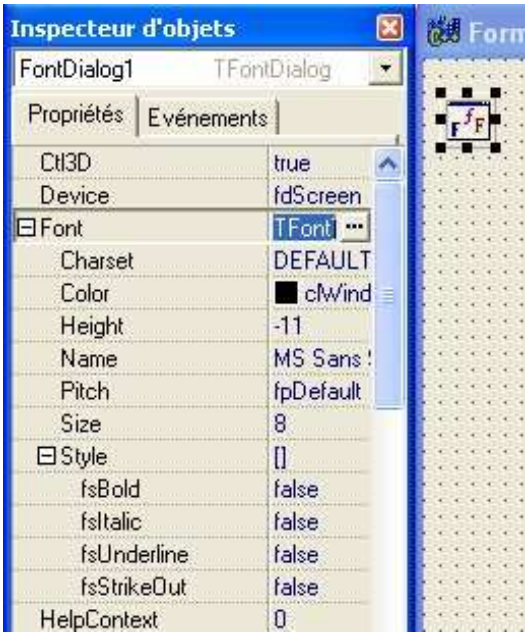


Figure 11 : Propriétés de la boîte de sélection de polices

Les options de la boîte elle même permettent de limiter l'utilisateur dans ces choix (par exemple, ne l'autoriser qu'à choisir des fontes non proportionnelles) ou de contrôler l'aspect visuel de la boîte.

3.4.4 Les boîtes de Recherche et Recherche/Remplacement

Ce sont les boîtes les plus compliquées à utiliser. Tout d'abord, ce sont les seules pour lesquelles le comportement n'est pas modal et binaire :

En effet, elles restent ouvertes tant que l'utilisateur appuie sur le bouton « Suivant ». En outre, elles n'ont pas de bouton « Ok ». Le seul moyen permettant de les fermer consiste à utiliser le bouton « Annuler ».

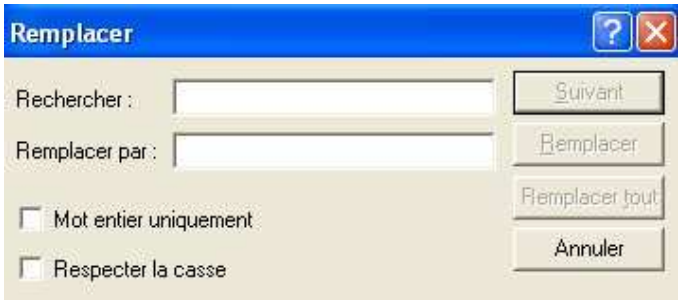


Figure 12 : Exemple de boîte Chercher/Remplacer

En outre, elles n'effectuent pas directement la recherche ! Elles ne permettent que de saisir le texte à rechercher (ou le texte de remplacement) et de sélectionner des options. Le corps de la recherche reste à la charge du programmeur.

Les options de ces boîtes de dialogue sont aisées à comprendre, elles conditionnent l'affichage des diverses possibilités et le résultat de la sélection de l'utilisateur.

Le résultat de la boîte comprend, entre autres : la recherche de mots complets (**`frWholeWord`**), la prise en compte de la casse (**`frMatchCase`**), le sens et la portée de la recherche (**`frDown`**, **`frHideUpDown`**, etc). Par défaut, les boîtes s'affichent avec l'étendue totale de leurs possibilités. Toutefois, l'utilisateur peut les limiter en activant des options dont le nom comprend **`Hide`** ou **`Disable`**. Par exemple **`frDisableMatchCase`** désactive la case permettant de sélectionner le respect de la casse des mots recherchés.

Comme la recherche elle-même, la prise en compte des options de recherche est entièrement à la charge du programmeur. On ne répètera jamais assez que ces boîtes ne sont que des outils de saisie des options!

La boîte de recherche simple fournit un événement nommé **`OnFind()`** notifiant au programmeur l'appui sur la touche « Suivant » de la boîte. La boîte de remplacement lui ajoute un événement nommé **`OnReplace()`**.

Un exemple d'utilisation consiste à utiliser ces événements en conjonction avec la méthode **`FindText()`** d'un composant **`TRichEdit`**.

3.4.5 Exercice d'application

Nous vous proposons de tester l'utilisation des différentes boîtes de dialogue sur le texte contenu dans un composant **`TRichEdit`**. Ce composant est très proche de **`TMemo`** dans le sens où il reprend la plupart de ses fonctionnalités en ajoutant celle de lire un fichier **`.rtf`** et de le formater correctement. Malheureusement, il ne propose ni propriété ni méthode permettant d'accéder à la mise en forme de chaque caractère. En revanche, il est possible de formater globalement le texte.

Les fonctionnalités suivantes devront être accessibles à partir d'un menu :

- Charger le texte dans le composant (méthode **`LoadFromFile`**() de la propriété **`Lines`**) à partir d'une fichier sélectionné à l'aide d'une boîte d'ouverture de fichier. Les extensions à considérer des fichiers sont : les fichiers textes (*.txt), les fichiers au format rtf (*.rtf) et tous les fichiers.
- Sauvegarder le texte dans un fichier sélectionné par la boîte de sauvegarde (méthode **`SaveToFile`**() de la propriété **`Lines`**)
- Manipuler l'apparence visuelle du texte du composant riche avec le changement de couleur et le changement de police de caractères
- Effectuer des recherches et des remplacements avec les bonnes boîtes
- Quitter l'application

Pour aller plus loin, on peut proposer de sauvegarder le texte s'il a été modifié par des opérations de recherche/remplacement ou par une saisie utilisateur avant d'en charger un nouveau ou de quitter l'application. L'événement **`OnCloseQuery`**() de la fiche est alors à considérer.

Rien de cet exercice n'est vraiment compliqué, il y a juste beaucoup de code à écrire. Une partie de la solution de cet exercice se trouve dans le guide de prise en main.

3.5 Les boîtes combo

Les boîtes combo permettent de rassembler en un seul composant les fonctionnalités d'une zone de saisie et d'une boîte de liste moyennant quelques limitations :

- La zone d'édition ne peut pas avoir de masque de saisie
- La boîte de liste est à sélection simple

Par conséquent, les propriétés sont hybrides de celles de la boîte de liste et de la zone d'édition, pour citer les plus importantes :

- **`Items`** : liste des éléments présents dans la liste déroulante
- **`ItemIndex`** : numéro de l'élément sélectionné. Le texte de l'élément sélectionné est accessible via : **`Items->Strings[ItemIndex]`** ou plus simplement via **`Text`** car le texte de l'élément sélectionné est recopié dans la zone d'édition
- **`Text`** : texte sélectionné dans la liste ou saisi dans la zone d'édition
- **`Sorted`** : à l'instar des boîtes de liste « normales », il est possible de stipuler que les différents éléments de la liste soient triés dans l'ordre lexicographique croissant.
- **`Style`** : propriété déterminant le style de la boîte combo
    - **`csSimple`** : la liste est visible en permanence
    - **`csDropDown`** : il s'agit de la présentation habituelle d'une boîte combo où l'affichage de la liste s'obtient par activation d'un bouton placé à droite de la zone d'édition.
    - **`csDropDownList`** : identique à la précédente à la différence que l'on ne peut sélectionner qu'un élément déjà présent dans la liste. Cela permet de créer une liste fixe que l'on fait dérouler à l'aide du bouton.
    - **`csOwnerDrawFixed`** : l'affichage de chaque cellule de la liste est sous la responsabilité du programmeur qui doit fournir une réponse à l'événement **`OnDrawItem()`** appelé pour dessiner chacun des éléments de la liste. Ceci permet, par exemple, de créer une liste de couleurs ou de styles de traits. Dans

ce cas, les différents éléments de la liste sont tous de la même hauteur, laquelle est fixée par la propriété **`ItemHeight`**.

- **`csOwnerDrawVariable`** : identique à la précédente à la différence que chaque élément peut avoir une hauteur différente. L'événement **`OnMeasureItem()`** est lancé juste avant **`OnDrawItem()`** afin de fixer la hauteur de l'élément à dessiner.

La possibilité offerte au programmeur de contrôler totalement l'affichage de chacun des éléments de la liste est d'autant plus intéressante que cela n'empêche pas pour autant la saisie de nouveaux éléments dans la zone de saisie.

Il est important de noter que la saisie d'un nouvel élément n'entraîne pas pour autant son inclusion dans la liste des possibilités. Cette dernière est laissée à la responsabilité du programmeur. Une fois de plus, cela s'avère particulièrement judicieux car on a alors la possibilité de « filtrer » les inclusions.

A la conception, il est possible de fixer le texte initial d'une boîte combo grâce à sa propriété **`Text`** mais pas en spécifiant un numéro d'index dans la liste. Ceci est seulement réalisable par programmation, par exemple dans l'événement **`OnCreate()`** de la liste.

3.5.1 Exercice

On désire réaliser une boîte combo qui n'accepte d'inclure dans sa liste que des nombres compris entre 0 et une limite indiquée dans une zone d'édition. Réaliser cette interface sachant que la validation de la valeur se fait par l'appui sur un bouton.

Puisque la validation de la saisie se fait grâce à un bouton, nous allons nous concentrer sur le code de l'événement **`OnClick()`** associé. On peut lire la valeur de **`TEdit`** et de **`TComboBox`** par la propriété **`Text`**. La méthode **`AddItem()`** permet d'ajouter un élément à un **`TComboBox`**.

3.6 Les menus

Il y a grossièrement deux grandes catégories de menus : le menu principal d'une application, qui s'affiche en haut de la fenêtre principale et qui varie finalement très peu au cours de l'exécution et les menus contextuels, activés par le bouton droit de la souris ou la touche spéciale du clavier de Windows. Ils partagent néanmoins de très nombreuses caractéristiques.

Deux composants permettent de créer des menus principaux :

- **`TMainMenu`** de la palette « Standard »
- **`TActionMainMenu`** de la palette « Supplément »

Les avantages des uns et des autres sont discutés au paragraphe 3.10 concernant la gestion des actions utilisateur.

Cette section est dédiée au composant **`TMainMenu`**.

3.6.1 Mise en place d'un menu principal

Tout commence par l'insertion d'un composant non visuel **`TMainMenu`** sur la fiche correspondant à la fenêtre principale de votre application. Ensuite, il suffit de double cliquer sur l'icône de celui-ci ou d'activer le « Concepteur de menus » dans son menu contextuel pour entrer dans l'éditeur de menu (commun au menu principal et aux menus contextuels).

Par définition, le menu principal se loge toujours en haut de la fenêtre principale, au dessus des composants possédant l'alignement **`alTop`** s'ils sont présents (des barres d'outils, par exemple). En mode conception, il n'apparaît qu'après la première activation du concepteur de menus.

La propriété principale de chaque composant menu est **`Items`**, propriété regroupant l'ensemble des éléments présents dans le menu, chaque élément étant lui même un objet de la classe **`TMenuItem`**.

3.6.2 L'éditeur de menus

La figure suivante illustre le fonctionnement de l'éditeur de menus. Celui-ci permet de construire un menu de façon très intuitive. Comme le montre l'inspecteur d'objets, tout élément de menu est doté d'un identificateur particulier. Ceci a pour conséquence positive qu'il est très facile d'accéder à chacun d'entre eux et comme aspect malheureux la saturation de l'espace de nommage des identificateurs lorsque les menus regroupent de nombreux éléments. Notons toutefois qu'il est possible, et c'est un cas unique, de laisser la propriété **`Name`** en blanc pour un élément auquel l'on n'accèdera pas par programmation !

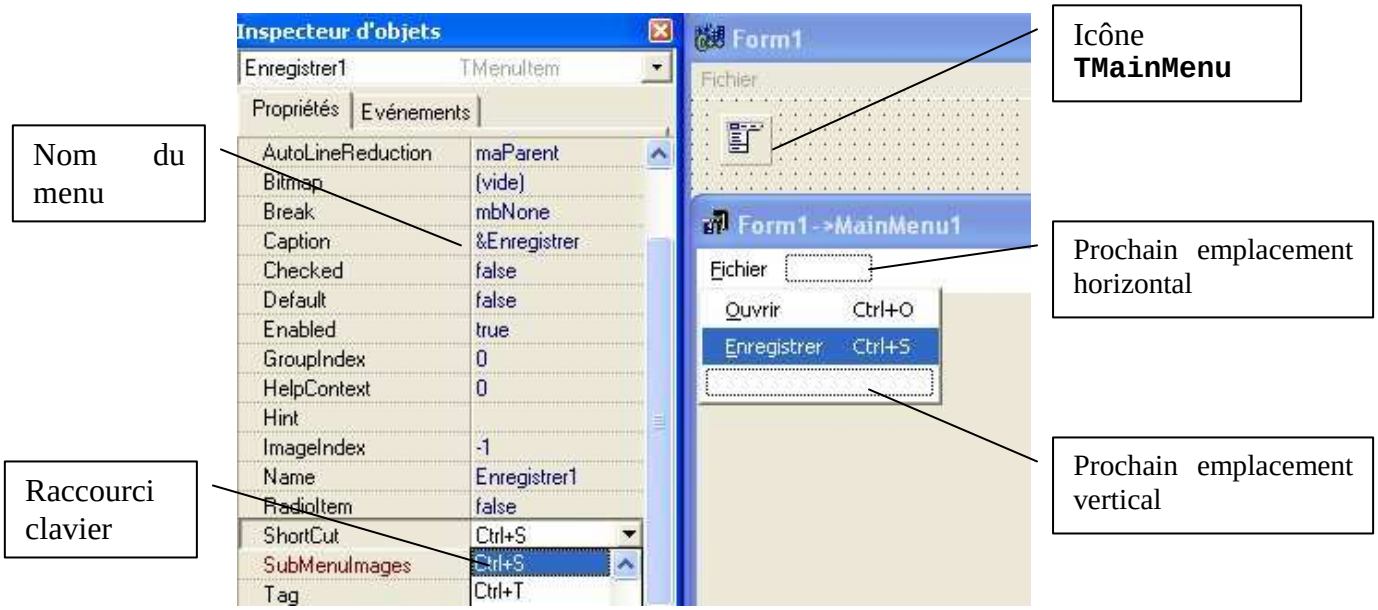


Figure 13 : L'éditeur de menus

Quelques petites indications s'imposent :

- Comme pour les boutons ou les labels de tout poil, il est possible d'utiliser le caractère esperluette (**&**) pour signaler la lettre active d'un élément de menu. Si l'élément en

question fait partie de la barre horizontale du menu principal, la combinaison **`Alt-lettre-active`** permettra d'accéder directement au menu.

- Lorsque vous saisissez le **`Caption`** d'un élément, le concepteur le valide et en ajoute automatiquement un nouveau du même type. Il ne faut pas s'inquiéter de la présence des ces éléments fantômes présents en mode conception : ils n'apparaîtront pas lors de l'exécution.
- Lorsque l'on désire ajouter un séparateur dans un menu « vertical »,il suffit de taper le caractère tiret – dans son **`Caption`**.
- « Griser » un menu s'obtient en basculant sa propriété **`Enabled`** à **`false`**.
- Positionner à **`false`** la propriété **`Visible`** d'un menu permet de le rendre invisible et donc totalement inactif. Nous verrons que cette propriété est particulièrement utile car la structure d'un menu est figée après la conception : il n'est pas possible d'ajouter ou de supprimer un élément dans un menu lors de l'exécution. Nous pourrons toutefois utiliser la propriété **`Visible`** pour simuler un ajout ou une suppression.
- On pourra accéder au menu contextuel (Figure 14) pour créer ou supprimer un item de menu ou bien encore créer un sous menu à « droite ».
- Il suffit de glisser-déplacer un item de menu pour changer sa place. On ne peut plus simple, non ?

Pour associer une action à un menu, il suffit de double cliquer sur la propriété **`Action`**. Il faut ensuite écrire le code associé. Pour une centralisation efficace des actions, il peut être utile d'utiliser des mécanismes comme les gestionnaires d'actions que l'on étudiera ultérieurement.

3.6.2.1 Les modèles de menus

C++ Builder propose d'utiliser des modèles de menus déjà prêts. Vous pouvez y accéder par le menu contextuel attaché au bouton droit de la souris dans le concepteur de menus. La figure suivante illustre cette possibilité :

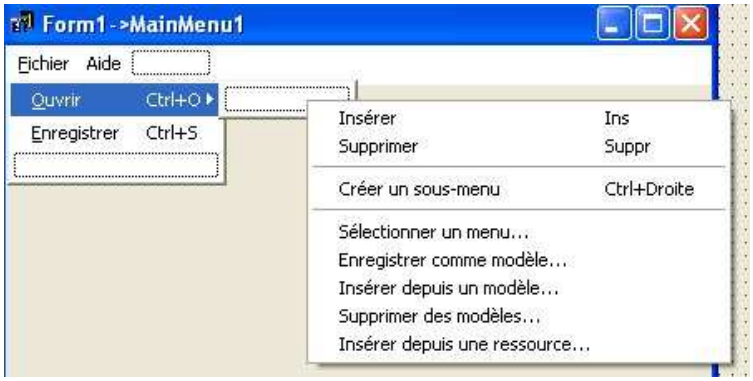


Figure 14 : Menu conceptuel de l'éditeur de menu

Les modèles de menus sont très efficaces car ils reprennent des menus canoniques et très complets. Il vous suffira alors d'ajouter les fonctionnalités qui vous manquent et de supprimer (ou rendre invisibles) celles que vous ne souhaitez par voir apparaître. La figure suivante montre la liste des modèles de menus disponibles par défaut :

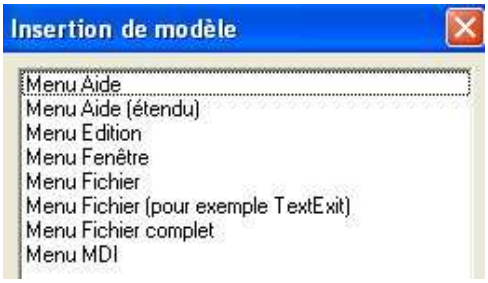


Figure 15 : Modèles de menus par défaut

Comme vous pouvez le constater, cette liste contient même un menu MDI complet prêt à l'emploi! La figure suivante montre un menu d'aide directement obtenu après insertion du modèle Aide.

3.6.2.2 Les sous-menus

Il est très simple de créer un sous-menu attaché à un élément quelconque : il suffit d'activer l'option correspondante dans le menu contextuel (ou d'activer le raccourci Ctrl-Flèche Droite).

Notez que, dans l'inspecteur d'objets, rien ne permet de différencier un item auquel est attaché un sous menu d'un autre, ce qui peut paraître surprenant car ils ont un comportement tout à fait différent et sont représentés avec une petite flèche vers la droite. La Figure 14 montre la création d'un sous-menu.

3.6.2.3 Les menus à comportement radio

Il est possible de conférer à certains des éléments d'un menu un comportement rappelant celui des boutons radio dans le sens où seul l'un d'entre eux sera précédé d'un rond noir. Cela permet, par exemple, d'affecter des options.

Pour transformer un ensemble d'éléments en éléments radio, il faut commencer par positionner leur propriété **`RadioItem`** à **`true`** et leur affecter à tous une même valeur de propriété **`GroupIndex`**, laquelle doit être obligatoirement supérieure à 7 afin d'éviter des conflits avec les valeurs prédéfinies de groupes d'éléments de Windows. La propriété **`Checked`** permet de spécifier lequel de ces boutons sera activé par défaut. Attention, contrairement aux boutons radio, cliquer sur un élément ne l'active pas immédiatement : ceci doit être fait dans le gestionnaire de l'événement **`OnClick()`**.

3.6.2.4 L'utilisation des menus

Les éléments de menu n'ont qu'un seul événement : **`OnClick()`** auquel on affecte une méthode répondant à l'événement « activation du menu ». Il est nécessaire d'affecter la propriété **`Checked`** d'un élément radio dans le gestionnaire si vous voulez déplacer la marque radio dudit élément.

3.6.3 Un exemple de modification par programmation : la liste des derniers fichiers ouverts

Comme nous l'expliquions quelques lignes auparavant, la structure (nombre des éléments de chaque partie de menu) est figée lors de la conception : la propriété **`Items`** est en lecture seulement. Il n'est donc pas possible d'ajouter ou de retirer un élément du menu en cours d'exécution. Heureusement, il est possible de jouer sur la propriété **`Visible`** de chaque élément.

Nous allons profiter de cette aubaine pour construire une liste des quatre derniers fichiers ouverts ayant les propriétés suivantes :

- Lorsque la liste n'est pas vide, elle apparaît en bas du menu fichier, juste avant l'élément Quitter, divisée du reste des éléments par des séparateurs
- Si la liste est vide, elle est totalement invisible
- L'élément le plus ancien est remplacé par le nouveau lorsque la liste est pleine

Afin d'obtenir ce que nous désirons, il est possible d'utiliser la séquence d'instructions :

1. Nous commençons par réserver les emplacements nécessaires comme le montre la figure suivante : quatre emplacements vides et un séparateur

Figure 16 : Exemple de programmation de menus

2. Dans un second temps, nous basculons la propriété **`Visible`** des quatre emplacements et d'un des séparateurs à **`false`**. Attention ! si vous ne spécifiez pas de **`Caption`** pour un élément, par défaut son nom restera en blanc, effet dont nous ne voulons pas. Aussi, soit vous donnez un **`Caption`** qui sera de toute façon remplacé par la bonne valeur au moment de l'affichage, soit vous fournissez un nom au composant.
3. Afin de pouvoir accéder facilement aux quatre emplacements, nous ajoutons dans les déclarations privées de la classe **`Fiche`** :
    - un tableau de quatre **`TMenuItem`** qui sera affecté aux adresses des 4 éléments dans le constructeur
    - une variable entière indiquant le nombre d'éléments présents dans la liste
4. Il ne reste plus qu'à rendre visibles les éléments et à leur affecter le bon **`Caption`**.

L'exemple qui suit remplit tout simplement cette liste grâce à un bouton et une zone d'édition.

```
// Variables affectées individuellement aux éléments de menu
// destinés à la liste des derniers fichiers ouverts
      TMenuItem *MenuFich1;
      TMenuItem *MenuFich2;
      TMenuItem *MenuFich3;
      TMenuItem *MenuFich4;
private: // Déclarations de l'utilisateur
      // Déclaration d'une constante indiquant le nombre d'éléments de la
      // liste
      enum {NB_FICHIERS=4};
      // Tableau permettant de manipuler facilement les éléments de menu
      // destinés à la liste des derniers fichiers ouverts
      TMenuItem *MenuFichiers[NB_FICHIERS];
      // Variable comptant le nombre d'éléments présents
      int nbFichiers;

__fastcall TFPrinci::TFPrinci(TComponent* Owner): TForm(Owner)
{
      // Remplissage (bourrin) du tableau des éléments
      MenuFichiers[0]=MenuFich1;
      MenuFichiers[1]=MenuFich2;
      MenuFichiers[2]=MenuFich3;
      MenuFichiers[3]=MenuFich4;
      nbFichiers=0;
}
//---------------------------------------------------------------------
void __fastcall TFPrinci::bnElementClick(TObject *Sender)
{
      // Affichage du séparateur
      N2->Visible=true;
      // Mise en place du Caption et affichage de l'élément
      MenuFichiers[nbFichiers]->Caption=editElement->Text;
      MenuFichiers[nbFichiers]->Visible=true;
      // Mise en place du cyclage
      nbFichiers++;
      nbFichiers%=NB_FICHIERS;
}
```

Comme souvent, ce code répète plusieurs fois la même opération (mettre la propriété **`Visible`** des éléments à **`true`**) car cela coûterait aussi cher d'effectuer le test de visibilité.

3.6.4 Les menus contextuels

Les menus contextuels ou *popup* sont en tout point semblables (du moins pour ce qui est de leur conception) au menu principal d'application. La seule différence réside dans leur activation qui est contrôlée par deux propriétés.

- **`AutoPopup`** est une propriété de **`TPopupMenu`** qui indique si ce menu est activable par un clic sur le bouton droit de la souris (ou l'enfoncement de la touche idoine du clavier) ou s'il est uniquement accessible par programmation (méthode **`Popup()`**). Cette propriété est par défaut à **`true`**, ce qui correspond bien à l'utilisation la plus fréquente de ces menus.
- **`PopupMenu`** est une propriété définie par tous les contrôles et qui indique l'identificateur du menu surgissant actif dans ce contrôle. Ainsi, il est possible de spécifier un menu contextuel par contrôle. Si cette propriété n'est pas renseignée, alors le menu contextuel est celui du parent.

3.7 Les boîtes déroulantes

Le fonctionnement des ascenseurs est quelque peu étrange en C++ Builder. Nous allons donner un aperçu des possibilités qu'ils offrent.

3.7.1 Généralités

Les boîtes déroulantes **`TScrollBox`** (sur la palette « Supplément ») permettent de faire défiler des composants à l'aide d'ascenseurs. Pour bien comprendre leur fonctionnement, il nous faut définir un peu de vocabulaire à l'aide de la figure suivante :

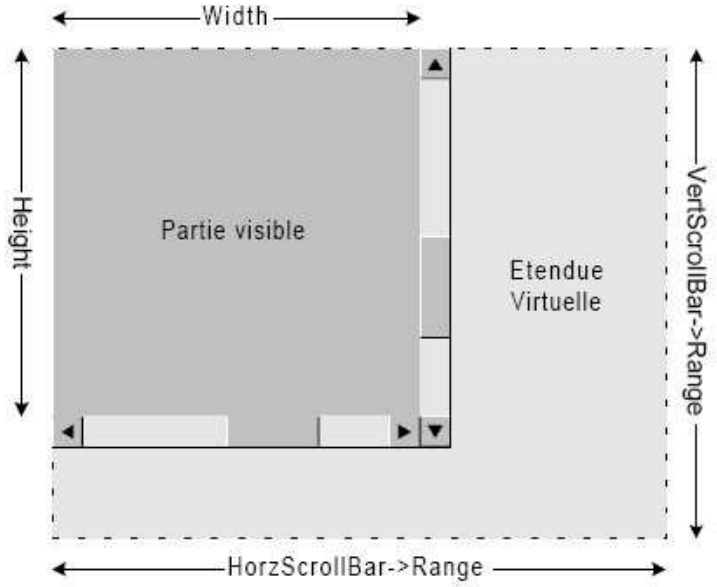


Figure 17 : Partie visuelle et étendue virtuelle d'un TScrollBox

La partie « visible » d'un composant **`TScrollBox`** est spécifiée, comme à l'accoutumée par ses propriétés **`Width`** et **`Height`**. Le nouveau concept est celui d'étendue virtuelle. En effet, le défilement sert à afficher les parties cachées de l'étendue virtuelle du composant. La largeur et la hauteur de cette dernière sont respectivement indiquées par les propriétés **`Range`** des ascenseurs horizontaux et verticaux lesquels sont stockés dans les propriétés **`HorzScrollBar`** et **`VertScrollBar`** de la **`TScrollBox`**.

L'étendue virtuelle peut varier au cours de la vie de l'application. Lorsqu'elle devient inférieure ou égale à la partie visible, les ascenseurs sont cachés.

Autre chose importante à savoir : modifier la position des ascenseurs ne génère pas d'événement particulier. Vous ne pouvez donc pas savoir en temps réel si leur position a changé. En conséquence, vous devez prévoir l'affichage complet de la zone virtuelle.

3.7.2 Que peut-on mettre dans une TScrollBox ?

Potentiellement, il est possible d'intégrer dans une **`TScrollBox`** n'importe quel type de contrôle, ce qui peut s'avérer pratique si l'on doit créer une interface graphique à la main en fonction d'un fichier de configuration.

Supposons par exemple, que vous écriviez une application dont certaines options booléennes sont passées dans un fichier. Vous souhaitez associer une case à cocher à chacune de ces options. Comme vous ne connaissez pas leur nombre à l'avance, vous êtes obligés de les construire à la main lors de l'exécution de votre programme.

Afin de ne pas surdimensionner votre fenêtre, il est possible de les placer dans une **`TScrollBox`**. Si vous activez la propriété **`AutoScroll`**, l'espace virtuel sera étendu automatiquement lorsque vous ajouterez des contrôles dans votre boîte déroulante. De toute façon, il vous est toujours possible de modifier les **`Range`** à votre guise.

Le plus souvent, les **`TScrollBox`** sont associées à un contrôle graphique ou un contrôle fenêtré unique dans le but d'afficher un document de grande taille. C'est ce que l'on appelle une vue.

3.7.3 Exercice : affichage d'un dessin avec facteur de zoom

L'exemple que nous allons traiter utilise un **`TImage`** placé dans une **`TScrollBox`** pour afficher un dessin de taille variable en fonction d'un certain facteur de zoom. Afin de ne pas compliquer inutilement l'exercice, nous afficherons une suite de 25 cercles concentriques.

Le facteur de zoom (variant entre 1 et 20) est indiqué par un composant **`TTrackBar`** lequel modélise un curseur placé le long d'une règle. Les propriétés les plus intéressantes sont :

Position	Valeur du curseur
Min	Valeur minimale du curseur, associée à la position « tout à gauche »
Max	Valeur maximale du curseur, associée à la position « tout à droite »
Frequency	Nombre d'unités séparant deux graduations sur la règle

On démarre avec une taille d'image de 50x50 correspondant au facteur de zoom=1. A chaque fois que le facteur de zoom change, vous modifierez la taille du **`TImage`**, (simplement en multipliant la taille de base par le facteur de zoom), vous modifierez l'étendue virtuelle de la **`TScrollBox`** si cela est nécessaire et mettrez à jour l'affichage des cercles concentriques. Il est possible de dessiner dans l'objet **`TImage->Canvas`**.

Pour bien exécuter cet exercice, il est préférable d'instancier manuellement le composant **`TImage`** d'affichage du dessin. En effet, lorsque l'on modifie les propriétés **`Width`** et **`Height`** d'un **`TImage`**, cela n'a pas de répercussion directe sur la taille de la zone cliente (**`ClipRect`**), laquelle est dans une propriété en lecture seulement. En effet, cela reviendrait à réallouer la zone de tampon mémoire associée à la **`TImage`**.

L'interface à créer ressemble à :

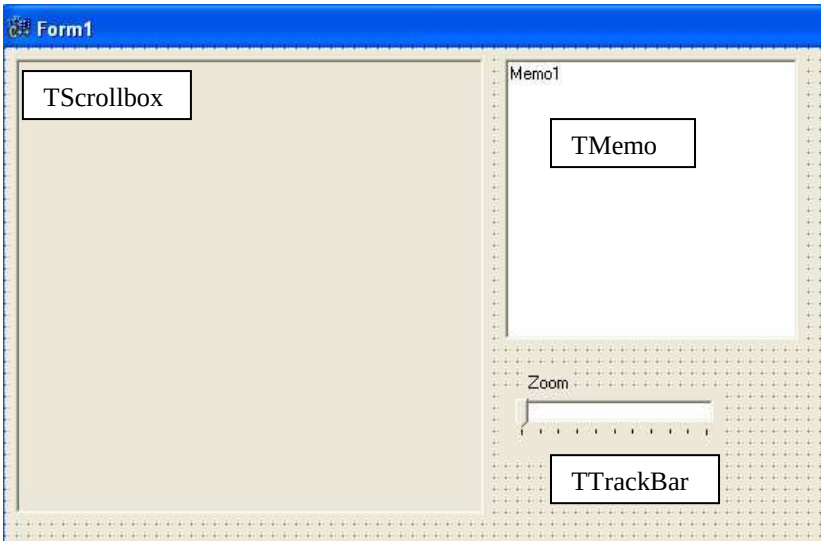


Figure 18 : TScrollBox et Zoom sur une zone à dessiner

Le composant **`TMemo`** sera utile pour faire une trace du programme Le composant **`TTrackBar`** sera responsable des changements de facteur de zoom.

Vous pouvez suivre les instructions suivantes :

1. Détruire le composant **`TImage`** en place (**`delete()`**)
2. Créer un composant **`TImage`**, le seul paramètre à passer au constructeur est le propriétaire, c'es-à-dire la fiche principale
3. Modifier les propriétés fondamentales de la **`TImage`**
   - **`Parent`** : identificateur de la **`TScrollBox`** ☚
   - **`Width`** et **`Height`** : taille de l'image à afficher
   - **`Top`** et **`Left`** : positionnés à zéro. On peut aussi utiliser la méthode **`SetBounds()`**.
4. Afficher le dessin

Cette technique, au demeurant fort simple souffre d'un handicap colossal : il faut allouer un **`TImage`** capable de contenir tout le dessin ce qui demande des ressources mémoire conséquentes. Aussi, pour des documents de grande taille, il faudra recourir à une autre technique qui elle n'utilise plus des **`TScrollBox`** mais des **`TScrollBar`**.

Pour aller plus loin dans l'exercice :

- Centrez l'image dans la **`TScrollBox`** si elle est plus petite que la taille statique de la **`TScrollBox`**,
- Ajoutez un composant zone d'édition affichant en permanence le facteur de zoom et permettant de le saisir. On effectuera tous les contrôles d'erreur nécessaires (valeurs en dehors de [Min, Max], saisie de valeur non numérique, etc.)
- Essayez d'atteindre les limites d'allocation de C++ Builder et fixez la valeur supérieure du zoom en conséquence.

3.8 Les ascenseurs simples

3.8.1 Généralités

Les ascenseurs simples sont encapsulés dans le composant **`TScrollBar`** (palette « Standard »). Une valeur numérique indiquant la position de leur « curseur » au sein de leur « étendue » leur est associée :

Les propriétés fondamentales sont les suivantes :

- **`Min :`** Valeur minimale de l'étendue du curseur
- **`Max :`** Valeur maximale de l'étendue du curseur
- **`Position :`** Valeur courante du curseur
- **`Kind :`** Spécifie si l'ascenseur est horizontal (valeur **`sbHorizontal`**) ou vertical (**`sbVertical`**)
- **`SmallChange :`** Valeur d'incrémentation du curseur lorsque l'utilisateur utilise les flèches.
- **`LargeChange :`** Valeur d'incrémentation du curseur lorsque l'utilisateur clique dans la barre d'ascenseur ou utilise les touches PgUp (Page haut) ou PgDown (Page bas)

Contrairement aux ascenseurs des **`TScrollBox`**, les **`TScrollBar`**s génèrent un événement lorsque l'on modifie leur valeur, il s'agit de **`OnChange()`**.

3.8.2 Exercice

Reprendre l'exercice précédent mais en réalisant l'affichage dans un **`TImage`** fixe et en faisant défiler l'image avec des ascenseurs simples.

Il est relativement simple d'adapter le code de l'exercice précédent à celui-ci. Aussi, je vous conseille de dupliquer tout le code du précédent projet dans un nouveau répertoire et de modifier l'application existante.

- Supprimer la **`TScrollBox`**
- Ajouter un composant **`TImage`** que l'on aura pris soin de poser sur un **`TPanel`**
- Ajouter deux composants **`TScrollBar`**, un horizontal et un vertical situés de part et d'autre du **`TImage`** (propriété **`Kind`**).

La gestion du zoom et du défilement est totalement différente dans le sens où, d'un côté, on construit complètement l'image un document et les barres de défilement ne font que faire glisser une loupe dessus alors que de l'autre côté, c'est le programmeur qui gère le défilement en récupérant la position des barres de défilement pour construire l'image de la partie visible d'un document.

Cette solution est un peu plus compliquée à mettre en œuvre mais beaucoup plus souple au sens où elle autorise des documents d'une taille de visualisation virtuellement illimitée.

En effet, dans le cas précédent, on utilise un composant **`TImage`** dans lequel toute la scène est dessinée. Aussi, la taille de se composant croit-elle avec le facteur de zoom. Lorsque l'on utilise les barres de défilement simple, la taille du composant **`TImage`** ne varie pas.

3.9 Les barres d'outils de C++ Builder

Deux composants permettent de créer des barres d'outils :

- **`TToolBar`** de la palette « Standard »
- **`TActionToolBar`** de la palette « Supplément »

Les avantages des uns et des autres sont discutés au paragraphe 3.10 concernant la gestion des actions utilisateur.

Cette section est dédiée au composant **`TToolBar`**. Par défaut, un objet de classe **`TToolBar`** se place en haut de sa fenêtre parent. C'est un objet conteneur destiné à recevoir divers objets fils parmi lesquels on recense :

- tout type de contrôles
- des séparateurs
- des boutons gadgets

Ces deux dernières classes étant plus spécifiquement liées aux barres de contrôles ou autres palettes flottantes.

3.9.1 Insertion de contrôles standards dans une barre d'outils

Si les contrôles n'existent pas encore, il suffit de sélectionner la barre de contrôle avant de les placer, ils s'insèreront directement à leur place. On remarque, que, par défaut, les contrôles sont collés les uns aux autres constituant des groupes. Heureusement, il est possible de laisser de l'espace en insérant des séparateurs avec une commande de menu rapide (bouton de droite). Ceux-ci sont aisément repositionnables et redimensionnables. La présence de séparateurs constitue des groupes de contrôles.

En revanche, si les contrôles existent déjà, il faut les déplacer par des opérations Couper / Coller, le collage devant s'effectuer barre d'outils sélectionnée.

3.9.2 Les boutons gadgets

Les boutons gadgets ressemblent à s'y méprendre aux anciens turbo boutons de Delphi 2. Toutefois, il ne faut pas s'y tromper, ce ne sont pas des contrôles car ils ne reposent pas sur une fenêtre. Ils ne peuvent donc pas recevoir et émettre des messages Windows de leur propre chef mais doivent être utilisés au sein d'une fenêtre dérivant de **`TGadgetWindow`**, comme, par exemple, **`TToolBar`**.

C++ Builder masque le travail de la **`TGadgetWindow`** en proposant au développeur des gestionnaires d'événements au niveau de la fiche et qui semblent directement liés aux boutons gadgets.

Ils sont créés en utilisant la commande « Nouveau bouton » du menu contextuel (clic droit avec la souris) des objets de classe **`TToolBar`**. A l'instar des turbo boutons (qui étaient présents dans Delphi 2 pour les émuler) ils peuvent être de type commande ou settings et adopter un comportement radio dans un groupe. Ici un groupe est un ensemble contigu de boutons gadgets entouré de séparateurs.

Les différents bitmaps associés aux boutons présents dans une barre d'outils sont tous regroupés au sein d'une propriété de la barre d'outils : **`ImageList`**. Chaque bitmap est ensuite adressé dans cette propriété via son index.

Voici donc la marche à suivre pour créer une barre d'outils avec des boutons gadgets :

1. Créer tous les bitmaps en les enregistrant dans des fichiers .bmp. Vous disposez pour cela, soit de l'éditeur intégré, soit de tout utilitaire capable de générer des .bmp.
2. Rassembler tous les bitmaps dans un objet **`TImageList`** (les fichiers .bmp ne sont alors plus nécessaires)
3. Créer l'objet **`TToolBar`** et lui associer la **`TImageList`**
4. Créer les boutons gadgets, leur adjoindre un style et leur associer leur bitmap

Pour la marche à suivre détaillée, on pourra se référer au guide de prise en main du produit. Je vous conseille aussi la lecture du paragraphe suivant qui permet de mettre en place un gestionnaire d'actions communes au menu principal et à la barre d'outils.

3.10 Les actions utilisateurs

Nous avons vu comment faire le lien entre une action et un item du menu principal. Faire de même entre un bouton d'une barre d'outils et une action est trivial.

Cependant, si l'on veut gérer à la fois un menu principal et une barre d'outils dont certaines actions (et images !) sont identiques, comment faire pour ne pas dupliquer de code ? C++ Builder propose plusieurs mécanismes élégants pour répondre à cette question. Le premier mécanisme disponible avec toutes les versions s'appelle *l'éditeur de liste d'actions*. Le second, disponible uniquement avec les versions entreprise et professionnelle se dénomme *l'éditeur de gestionnaire d'actions*.

3.10.1 Le gestionnaire d'actions

Quels sont les avantages du gestionnaire :

- Il exploite la VCL et permet des fioritures comme la disparition des menus non utilisés (à la MS Office, capacité que je m'empresse de désactiver la plupart du temps)
- Il permet un développement plus intuitif et plus rapide que son homologue

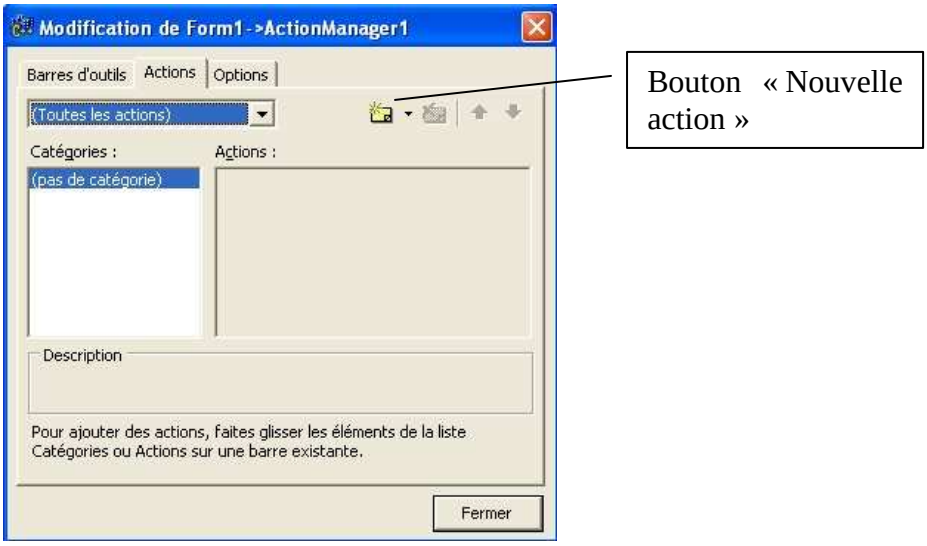


Figure 19 : Le gestionnaire d'actions

Voici la marche à suivre pour disposer d'un gestionnaire d'actions

1. Déposez un composant **`ActionManager`** sur la fiche (palette « Supplément »). Le composant est non visuel et apparaît donc comme une icône sur la fiche.
2. Si besoin, associez un composant **`TImageList`** à la propriété **`Images`** (pour une barre d'outils)
3. Double cliquez sur l'icône du gestionnaire pour l'ouvrir. Une boîte de dialogue, l'éditeur, apparaît alors.
4. Cliquer sur le bouton «Nouvelle action » pour ajouter une nouvelle action (ou utiliser le menu contextuel ou encore la touche Ins) standard ou non, en fonction de ce que vous voulez faire. Vérifiez que [pas de catégories] est sélectionné. Remplissez les propriétés suivantes
   - **`Caption`** : le fonctionnement est similaire au menu. Exemple : &Ouvrir
   - **`Category`** : Choisir une catégorie, par exemple, Fichier. Cela permet de regrouper les actions sous une même bannière.
   - **`Hint`** est le conseil d'aide
   - Donner éventuellement **`l'ImageIndex`** du composant **`TImageList`**
   - Donner un nom **`Name`** significatif à l'action, par exemple, FichierNouveau
5. Répéter l'opération 4 autant de fois que nécessaire.

Les actions standards sont des actions prédéfinies que l'on retrouve généralement dans toute application.

Pour associer une action ou une catégorie d'action à un menu principal ou une barre d'outils :

- Poser un composant **`TActionMenuBar`** ou **`TActionToolBar`** sur la fiche. Apparaît alors un menu **vide** (et non pas une icône)
- Faire glisser des catégories ou des actions du gestionnaire vers le menu. Le tour est joué.

3.10.2 La liste d'actions

La liste d'actions est disponible pour toutes les éditions de C++ Builder. Elle permet aussi de gérer les actions pour plusieurs composants.

Pour l'utiliser, il suffit de poser ce composant non visuel sur la fiche. **`ActionList`** est dans la palette « Standard ». Définir chacune des actions se fait de manière très semblable au gestionnaire d'actions.

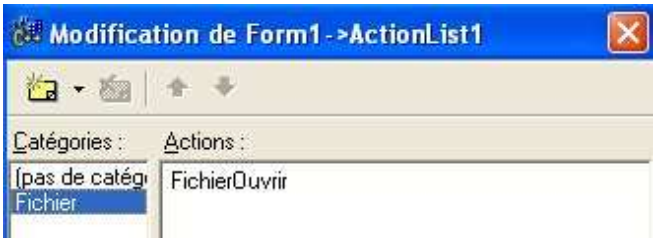


Figure 20 : L'éditeur de listes d'actions

Pour référencer une action de la liste dans un composant **`TMenuBar`** ou **`TToolBar`**, il suffit de renseigner la propriété **`Action`** des menus ou des boutons.

3.10.3 Quelle méthode choisir ?

Voilà un petit tableau pour vous aider :

Méthode	Composants	
Liste d'actions	**`TMenuBar`** **`TToolBar`**	☹ Il faut aussi dessiner les menus/barres ☺ Plus général ☺ Toute édition
Gestionnaire d'actions	**`TActionMenuBar`** **`TActionToolBar`**	☺ Création intuitive des menus/barres par glisser/déplacer ☺ Plus puissant (VCL) ☺ Réservé édition Prof. et Ent

4. Utiliser une base de données MySQL

C++ Builder est livré avec différents moyens d'interagir avec des bases de données : le Borland Database Engine ou DBE et dbExpress (encore connu sous le nom DataSnap Direct).

Le support du BDE (dédié aux Paradox et dBase) est arrêté depuis 2003. Borland considère en effet que cette interface est arrivée en fin de vie et que les développeurs doivent se tourner vers dbExpress.

4.1 Connexion avec dbExpress

Borland préconise l'usage de cette couche logicielle pour interagir avec un serveur SQL. Cependant, je n'ai pas réussi à la faire fonctionner.

Je place un composant **`TSQLConnection`** dans un module de données. Je précise que j'utilise une base MySQL en copiant la librairie libmysql.dll dans le répertoire du projet et j'obtiens tout de même le message « Impossible de charger la librairie ». J'ai essayé aussi de charger une autre librairie dbexpmysql.dll livrée avec C++ Builder.

Les pistes que j'envisage pour l'utilisation de cette couche logicielle :

- Utiliser un driver OBDC libre pour dbExpress appelé dbxoodbc. J'ai testé avec la version 2.010 de l'année 2003. La configuration du driver est possible. L'interaction de l'environnement est de la base se fait correctement (par exemple, le choix des tables pour le composant **`TTable`**). J'ai obtenu une erreur à l'exécution lors de la connexion.
- Tester une nouvelle version du driver sus-cité disponible à l'URL http://www.justsoftwaresolutions.co.uk/delphi/dbexpress_and_mysql_5.html
- Utiliser une ancienne version de la librairie libmysql.dll (une 3.23 par exemple alors que mon serveur de tests est équipé d'une version 4.1.9)
- Utiliser un pilote propriétaire, par exemple *microOLAP DBX for MySQL*.

Si les fichiers dll sont compatibles, il faut utiliser les composants **`TSQLConnection`** et **`TSQLQuery`** (et non pas **`TSQLDataSet`**).

Le mode de connexion dbExpress est unidirectionnel mais cette limitation peut être contournée par programmation, comme le montre l'article suivant

4.2 Connexion avec le BDE

Il faut donc répondre à la question : comment se connecter à un serveur MySQL alors que cela n'est pas prévu d'origine ? La réponse est simple : utiliser le mécanisme ODBC (*Open DataBase Connectivity*, une API standard d'interaction avec les bases de données dont le but est de créer des composants indépendants du langage, du système SGBD et du système d'exploitation. Nous allons d'abord voir comment configurer une telle liaison. Une

description plus précise de l'utilisation des composants de base de données sera faite au chapitre suivant.

4.3 Configurer une connexion ODBC

Il faut tout d'abord télécharger un pilote ODBC – ou connecteur - pour que Windows puisse utiliser une base MySQL. Le connecteur ODBC MySQL est aussi connu sous le nom de MyODBC. Une fois que c'est fait, il faut le configurer. Il est nécessaire d'accéder à la fenêtre de gestion des *Sources de données (ODBC)*.

- A partir de Windows XP
  (Démarrer) Paramètres > Panneau de configuration > Outils d'administration > Sources de données (ODBC)
- A partir de l'administrateur BDE
  (O) Configuration > Pilotes > ODBC > MySQL ODBC 3.51 Driver puis choisir Administrateur ODBC avec le menu contextuel (obtenu par clic droit)

Les étapes de configuration sont décrites par les captures d'écran suivantes. Cela fonctionne que la connexion au serveur soit locale ou distante.

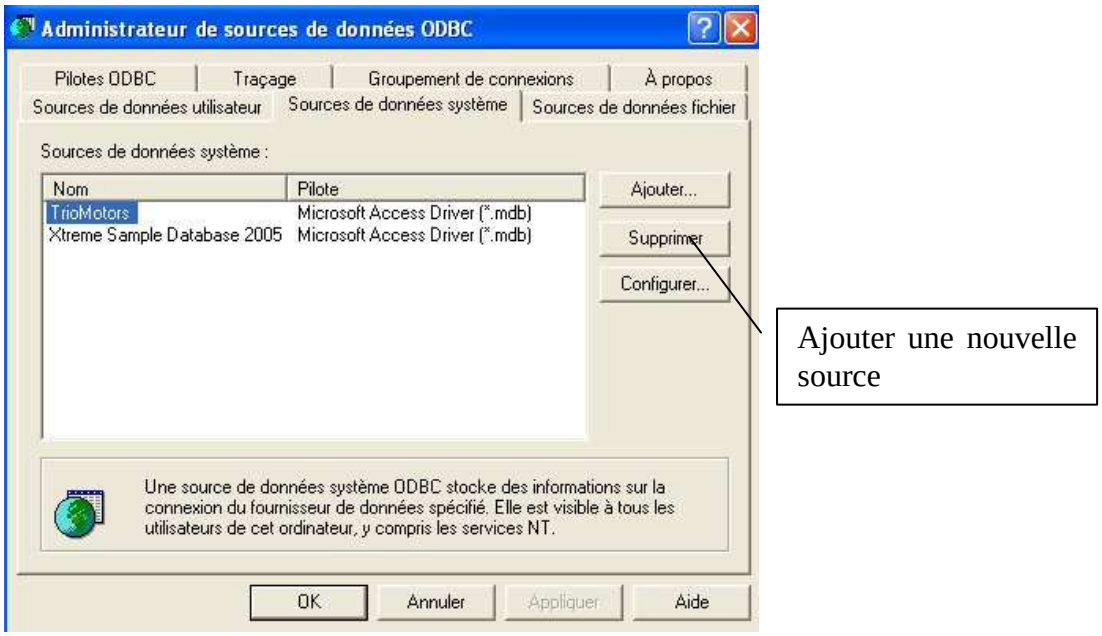


Figure 21 : Source ODBC - Créer une nouvelle source

Il faut tout d'abord créer une nouvelle source. Deux choix sont possibles en fonction de la visibilité de la nouvelle source : soit « utilisateur », soit « système ». Une source utilisateur n'est visible que pour l'utilisateur connecté. Une source système est visible pour tous les utilisateurs de la machine.

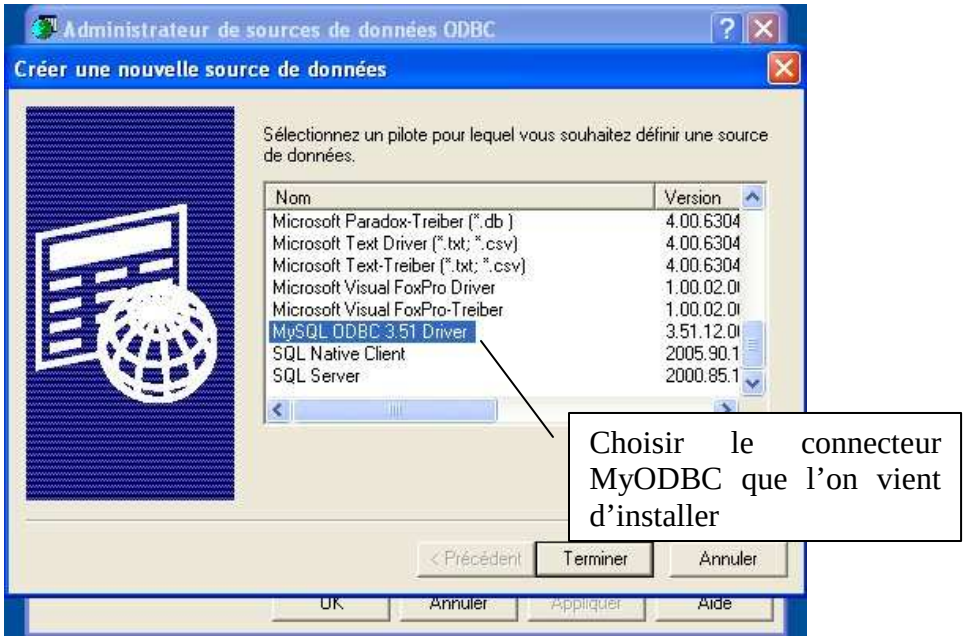


Figure 22 : Source ODBC - Choisir un type de source

Il faut ensuite renseigner les paramètres de la source :

- un DSN ou *Data Source Name*, sorte de raccourci pour désigner cette nouvelle source
- une description [facultative]
- le serveur, local ou distant
- le nom d'utilisateur
- le mot de passe
- le nom de la base

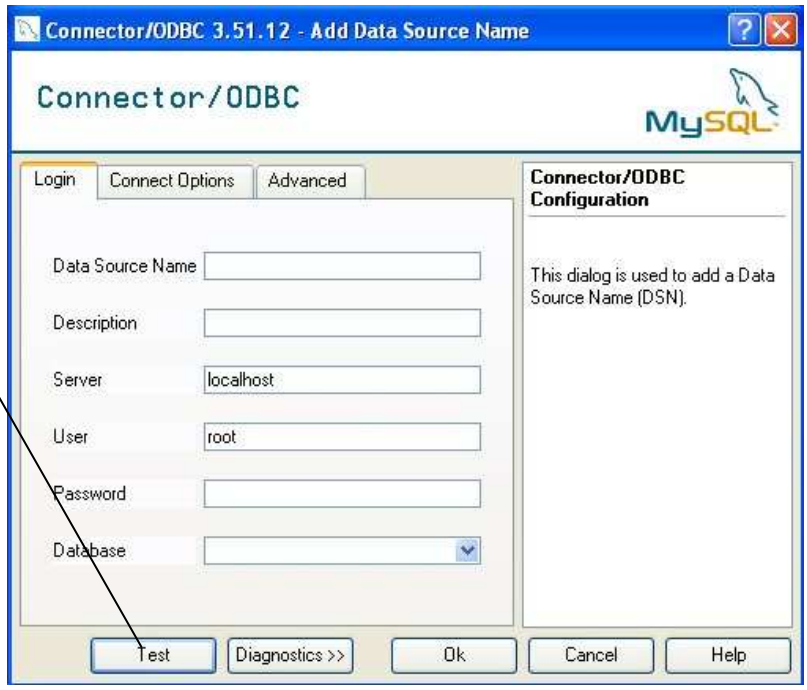


Figure 23 : Source ODBC - Paramètres (1)

La dernière étape est la création d'un alias de base avec l'Administrateur BDE, étape décrite au chapitre suivant.

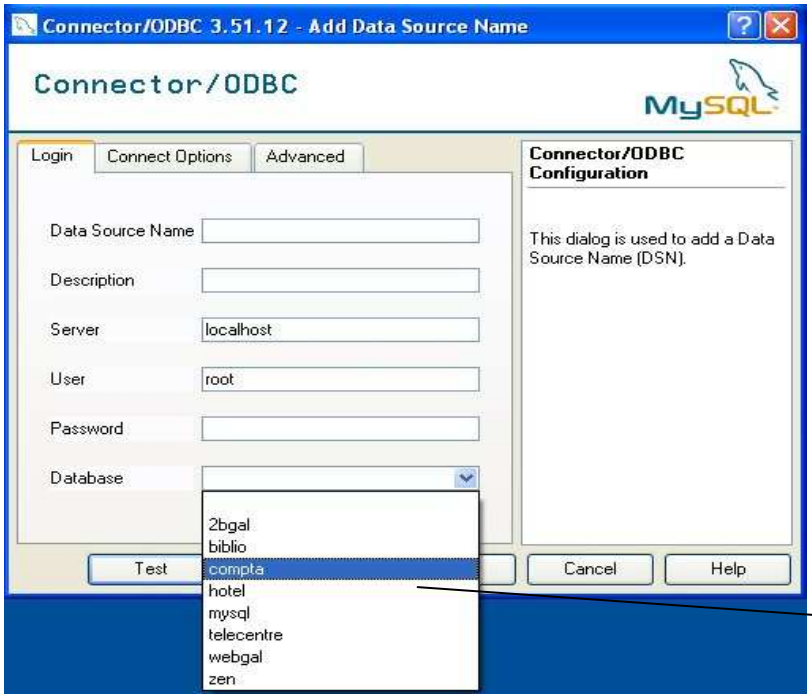


Figure 24 : Source ODBC - Paramètres (2)

Suivant les versions de MyODBC, certaines options doivent être sélectionnées :

- *Don't optimize column widths*
- *Return matching rows*

Il est aussi possible que les champs requêtes ne soient pas mis à jour lors des changements de structure de base et que seuls les index primaires de table soient reconnus.

4.4 Utiliser des composants ADO

On peut utiliser les composants de la palette « ADO » ou **ActiveX Data Objects .** Cette norme a été développée par Microsoft pour manipuler des données sans connaître la base de données sous-jacente. Leur fonctionnement est en tout point similaire à celui du BDE comme décrit au chapitre 5. La connexion ADO encapsule une connexion ODBC. Les étapes sont donc les suivantes :

1. Vérifier que le DSN de la connexion ODBC est correctement défini (voir la section 4.3)
2. Placer sur un module de données un composant **`TADOConnection`**
3. Placer ensuite un composant **`TADOTable`**
4. Ne pas oublier les composants **`TDataSource`**
5. Placer les composants visuels de bases de données sur la fiche.

Les figures suivantes reprennent les étapes de configuration des composants, notamment de **`TADOConnection`**.

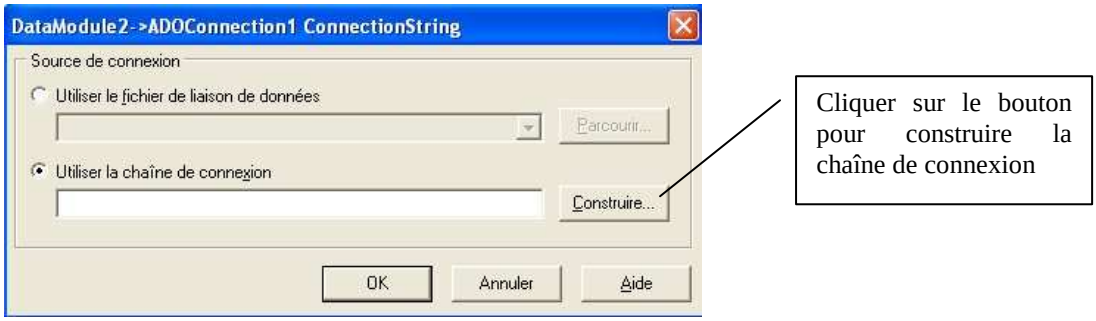


Figure 25 : Chaîne de connexion ADO

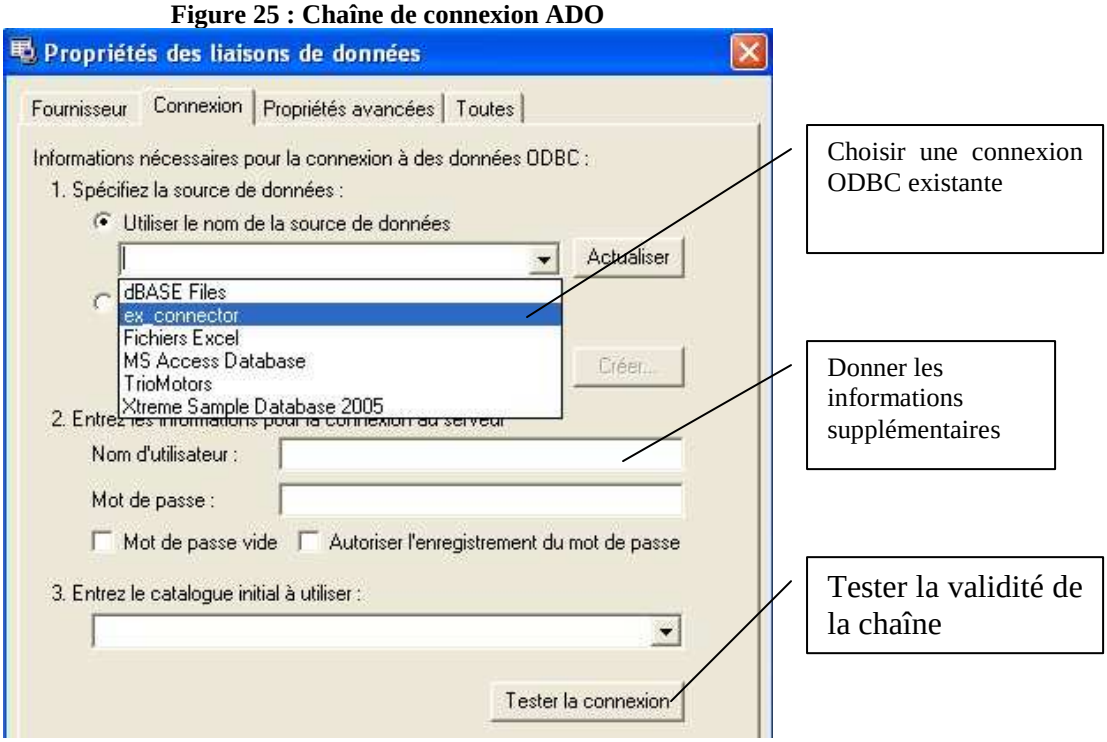


Figure 26 : propriété de la connexion ADO

La validité des informations saisies peut être contrôlée grâce au bouton « Tester la connexion ». Il est même possible de changer de base de données (grâce à la liste déroulante permettant de choisir le catalogue initial).

Le champ **`connection`** du composant **`TADOTable`** permet de choisir parmi les objets de type **`TADOConnection`** déjà définies. Il faut ensuite spécifier le champ **`TableName`**. La propriété **`Active`** permet d'activer la connexion ODBC.

4.5 Autres méthodes d'interaction

4.5.1 Utiliser des composants dédiés

La société CoreLab propose des composants VCL spécifiques pour se connecter à des bases de données MySQL : MyDAC.

4.5.2 Utiliser directement l'API MySQL

Un article est disponible sur Developpez.com rédigé par LFE qui décrit la manière de se connecter à une base de données MySQL en utilisant l'API fournie avec MySQL.

http://lfe.developpez.com/BCBmSQL/

5. Utilisation minimaliste des bases de données avec le BDE

Cette section n'a d'autre prétention que de présenter sommairement et par ordre chronologique les fonctionnalités minimales permettant de travailler avec des bases de données sous C++ Builder. Elles sont gérées par une couche logicielle intitulée Borland Database Engine ou DBE.

Nous allons décrire les fonctionnalités des bases de données au travers d'un exemple. Nous voulons gérer les écritures de comptes bancaires pour faire sa propre comptabilité.

5.1 Alias d'une base

La première étape consiste à créer une base de données par l'intermédiaire d'un **alias** de base de données.

Un alias de base de données est un **identificateur unique** permettant d'accéder à une base de données exploitable avec C++ Builder. Celle-ci peu être de plusieurs types en fonction des pilotes installés sur votre machine :

- Un répertoire regroupant des fichiers dBase ou Paradox (option par défaut, toujours présente). Il est désormais conseillé de convertir les bases de ces formats au format Interbase.
- Une base de données ACCESS (option)
- Une base de données SQL au format Interbase (option souvent présente)
- Un lien SQL vers un serveur Oracle, Sybase, DB2, SQL Server, etc (option)
- Tout lien ODBC reconnu par l'environnement Windows (toujours présent). La section précédente présente comment créer un lien ODBC pour une base de données MySQL.

La création d'un alias se fait très simplement avec un outil nommé « Administrateur BDE ». Dans notre cas d'école, nous allons créer un alias sur la base de données créée à la section précédente (avec le lien ODBC).

5.2 L'Administrateur DBE

L'Administrateur BDE est un outil à la fois très simple et très complet. Il est capable d'afficher soit l'ensemble des alias déjà créés (grâce à l'onglet « Bases de données ») soit des informations concernant le système telles que l'ensemble des formats de bases de données disponibles ou le format de certains éléments comme les nombres et les dates (grâce à l'onglet « Configuration »).

La fenêtre principale de l'administrateur BDE se décompose en deux volets :

- Dans le volet de gauche, vous sélectionnez un objet (un alias de base de données en mode « Base de données »)
- Dans le volet de droite, vous obtenez la liste de ses caractéristiques.

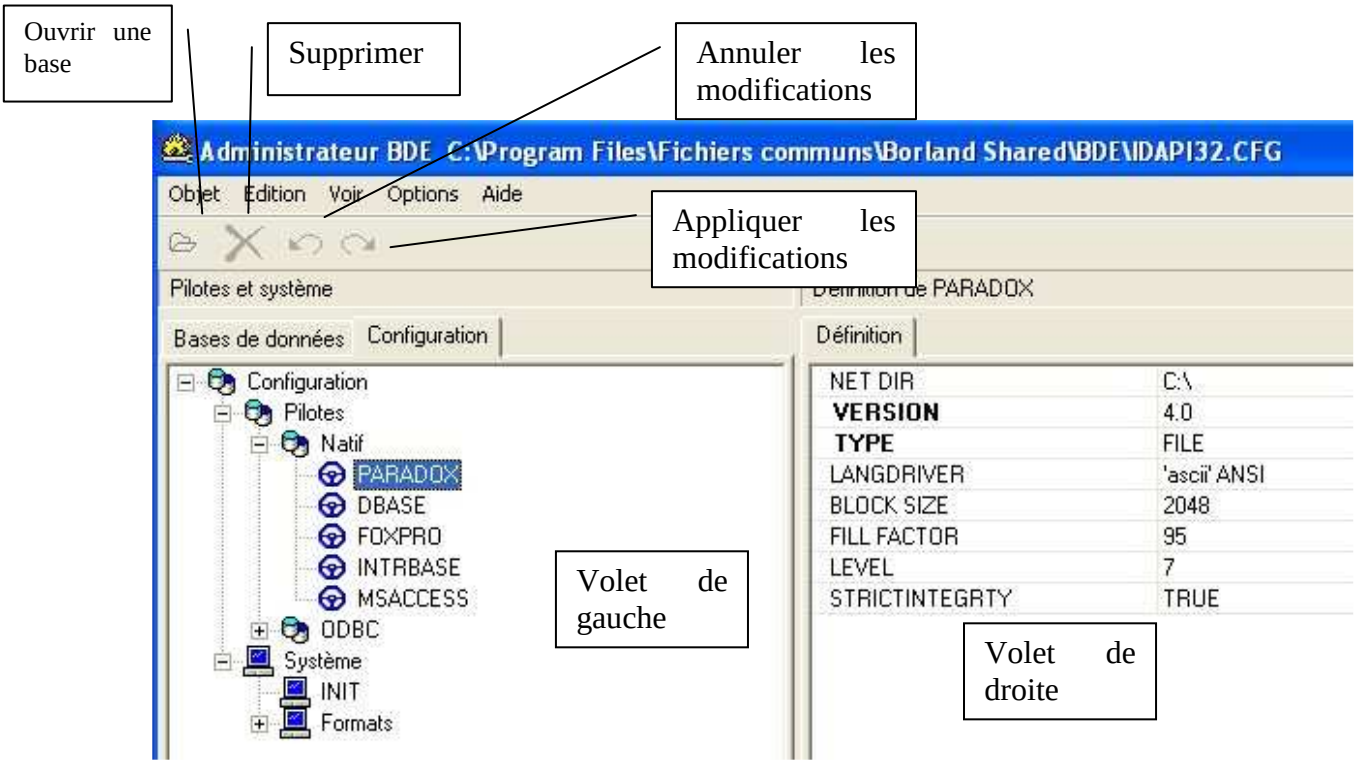


Figure 27 : l'administrateur BDE

Contrairement à ce que l'on pourrait penser un pilote natif n'est pas toujours plus rapide qu'un pilote ODBC. Si vous avez différentes possibilités, des tests de performance s'imposent.

5.3 Création de l'alias

La démarche est la suivante :

- Lancer l'utilitaire Administrateur BDE
- Sélectionner l'onglet « Base de données ». Vous pouvez consulter les informations des alias déjà créés pour se familiariser avec les informations.
- Pour créer un nouvel alias
  (M) Objet > Nouveau
- Sélectionner ensuite le type de base de données. Sélectionner pour notre exemple, le driver ODBC
- Donner les informations idoines pour le fonctionnement de la base.

Il faut absolument renseigner le nom de l'alias, le nom par défaut n'est guère engageant et le chemin des données.

Il n'y a plus qu'à valider en sélectionnant la flèche bleue. Une boîte de dialogue vous demande alors de confirmer. A partir de ce moment là, la base de données est accessible via C++ Builder.

Notez également qu'un utilitaire nommé « Module de base de données » décrit ultérieurement permet de créer des bases de données au format Paradox ou dBase ainsi que de tester des requêtes SQL simples. Bien que rudimentaire, cet outil permet de créer simplement des tables et de leur associer des clés primaires ou secondaires, des fichiers index ou des contraintes d'intégrité.

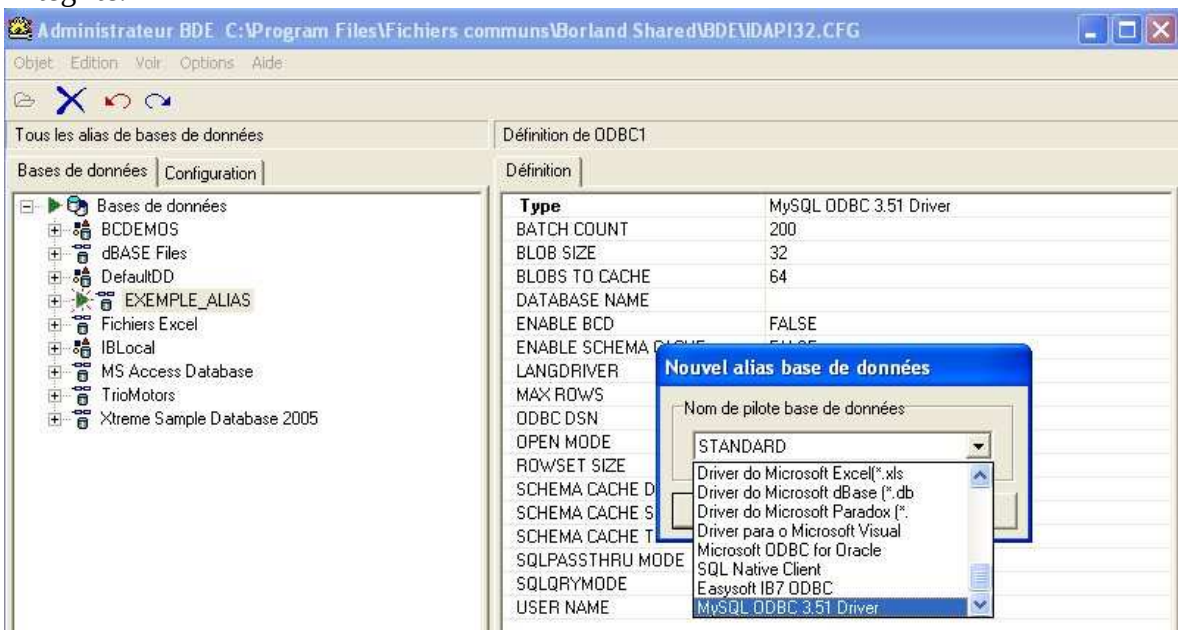


Figure 28 : Création de l'alias, sélection du type

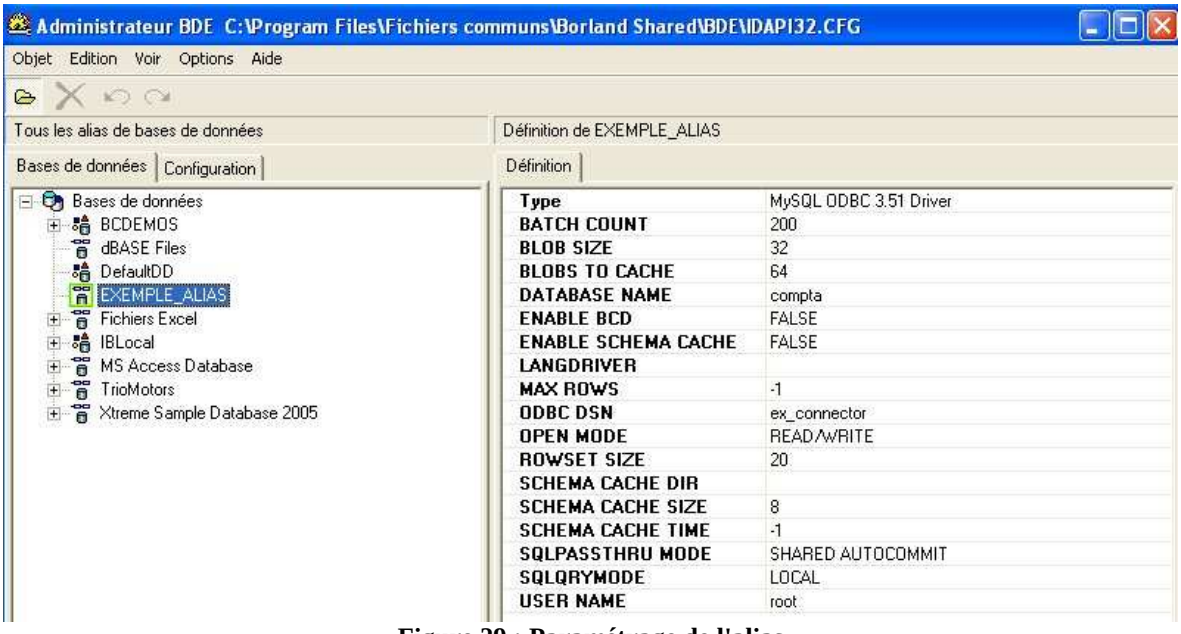


Figure 29 : Paramétrage de l'alias

5.4 Accès aux données

C++ Builder effectue une ségrégation forte entre les variables qui représentent les données en tant que document et les composants visuels qui permettent à l'utilisateur de les manipuler. C'est un exemple d'application du modèle Document / Visualisation.

Voici la description officielle de la gestion des bases de données telles que donnée dans l'aide en ligne :

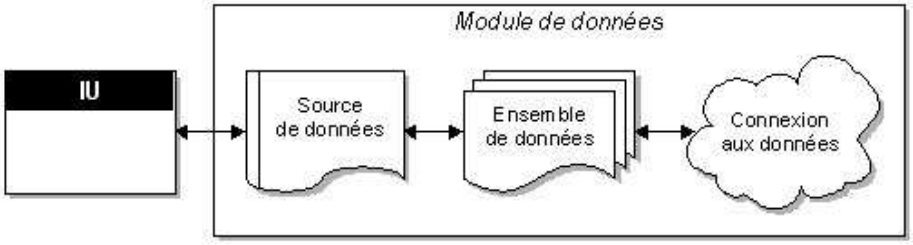


Figure 30 : Schéma général d'une application de base de données

Il est conseillé d'isoler l'**interface utilisateur** sur une fiche complètement indépendante du reste de l'application. Cela présente plusieurs avantages et notamment une plus grande flexibilité conceptuelle :

- les modifications que vous apportez à la gestion des informations de bases de données n'imposent pas la réécriture de l'interface utilisateur, tandis que celles que vous apportez à l'interface utilisateur ne vous obligent pas à modifier la partie de l'application qui utilise la base de données.
- La cohérence de l'interface utilisateur en développant des fiches communes à diverses applications
- le partage des fiches vous permet de développer des standards d'entreprise pour les interfaces des applications.

Le premier élément du module de données est une **source de données**. La source de données relie l'interface utilisateur à un ensemble de données qui représente les informations d'une base de données. Plusieurs contrôles orientés données disposés sur une fiche peuvent partager une même source de données. Dans ce cas, le contenu de chaque contrôle est synchronisé : lorsque l'utilisateur parcourt les enregistrements, les valeurs figurant dans les différents champs de l'enregistrement actif sont affichées dans les contrôles correspondants.

L'**ensemble de données** constitue le cœur de votre application de base de données. Ce composant représente un ensemble d'enregistrements de la base de données sous-jacente. Ces enregistrements peuvent être les données d'une seule table de base de données, un sous-ensemble des champs ou des enregistrements d'une table ou des informations émanant de plusieurs tables jointes en une vue unique. L'utilisation d'ensembles de données protège la logique de votre application de la restructuration des tables physiques de la base de données. Lorsque la base de données sous-jacente change, vous pouvez être amené à modifier la façon dont le composant ensemble de données spécifie les données qu'il contient, mais le reste de votre application peut continuer à fonctionner sans subir de modifications.

Différents types d'ensembles de données utilisent différents mécanismes de **connexion** aux informations de la base de données sous-jacente. Ces différents mécanismes déterminent les variantes majeures de l'architecture des applications de bases de données que vous créez.

Il est préférable de regrouper les éléments d'accès aux données – qui sont des composants non visuels – dans une unité spécialisée appelée **Module de données** qui se manipule comme une unité classique.

(M) Fichier > Nouveau > Module de données

Notre exemple de base de données dispose des relations suivantes :

- *type_transaction*
- *ecritures*

La première chose à faire consiste à associer un composant **`TTable`** à chacune de ces relations. Pour cela, il sélectionner le composant **`TTable`** dans la palette « BDE » qu'il faut coller dans le module de données créée auparavant. L'inspecteur d'objets permet ensuite de personnaliser le composant :

- Champ **`Name`** pour donner un nom explicite à la relation
- Champ **`DatabaseName`** pour associer un alias de base de données
- Champ **`TableName`** pour donner le nom de la table à choisir dans une liste déroulante (une fois que le composant est connecté)
- Champ **`Active`** : ce drapeau permet d'ouvrir la table. L'index utilisé est fondé sur la clé primaire sauf si l'utilisateur définit un index secondaire (champ **`IndexName`**). Ne peut valoir *true* que si **`DatabaseName`** et **`TableName`** sont renseignés. Toute modification sur la structure de la table entraîne irrémédiablement le passage de la propriété à *false*.
- Le champ **`IndexName`** représente l'index actif sur la table. Par défaut, c'est-à-dire si le champ est vide, l'index est la clé primaire. Le nouvel indice se choisit dans une liste déroulante.
- Le champ **`FileDefs`** contient la définition de chacun des champs de données de la table. Une boîte de dialogue spécifique s'affiche pour les lister.

Il est possible d'inclure d'autres composants non visuels dans un module de données. Citons, en particulier :

- Les composants représentatifs d'une base de données **`TDatabase`**. Ceux-ci ajoutent une couche supplémentaire entre les tables (ou les requêtes) et les alias de la base de données. Cela peut être utile si plusieurs bases partagent la même structure et que vous vouliez utiliser le même code indifféremment. La variable **`TDatabase`** pointera sur le bon alias de base et les composants **`TTable`** et **`TQuery`** référenceront le **`TDatabase`** directement.
- Les requêtes SQL **`TQuery`**. Les requêtes et les tables ont un ancêtre commun **`TDataSet`** dans le sens où ces composants ont pour résultat des ensembles de données que l'on pourra afficher. Les requêtes SQL sont paramétrées et il est possible de récupérer la valeur de ces paramètres directement dans certains composants ou par jointure sur une table ou le résultat d'une autre requête.
- Les procédures stockées SQL **`TStoredProc`** c'est-à-dire du code SQL précompilé pour une exécution plus rapide. Il s'agit habituellement de code de maintenance et non pas de code d'extraction.
- Les sources de données de type **`TDataSource`**. Ces composants, non visuels, établissent le lien entre les ensembles de données (tables ou requêtes) – en d'autres mots, la partie document du modèle C++ Buidler – avec les composants visuels chargés de permettre la manipulation des bases de données (la partie visualisation du modèle de données de C++ Builder). Bien que ce ne soit pas obligatoire, il est très

fortement conseillé de placer les composants **TDataSource** dans le module de données et pas très loin des **TDataSet** sur lesquels ils travaillent.

Il est possible d'avoir plusieurs sources de données pour le même ensemble de données. On dispose ainsi de plusieurs visualisations sur un même document. Le placement des objets sur le module de données n'ayant pas d'incidence sur le code produit, il vous appartient de le structurer de la manière la plus cohérente possible.

La propriété la plus importante des composants **TDataSource** est **DataSet**. En effet, celle-ci indique sur quel ensemble travaille la source de données. Vous pouvez spécifier dans cette propriété tout composant héritant de **TDataSet**, c'est-à-dire, en ce qui concerne les composants de base, **TTable** et **TQuery**.

5.5 Les contrôles orientés bases de données

Ce sont des versions spécialisées dans l'édition des bases de données des contrôles standards de Windows. On retrouve ainsi, par exemple, des boîtes d'édition, des listes ou des boutons radio orientés bases de données. Le contrôle le plus important est néanmoins celui qui permet de présenter une relation ou le résultat d'un requête sous forme tabulaire : **TDBGrid**. Tous les composants se trouvent dans la palette «ContrôleBD».

Les contrôles orientés bases de données se placent sur les fiches et non pas sur les modules de données : ils constituent la partie interface du modèle Document/Visualisation de C++ Builder.

Tous sont reliés aux composants non visuels grâce aux sources de données. Afin d'accéder à ces dernières, il ne faut pas oublier d'inclure l'entête du module de bases de données dans la fiche de présentation.

(M) Fichier > Inclure l'entête unité (Alt+F11)

Si rien ne se passe malgré tout, vérifiez aussi que vous disposez bien des composants **TDataSource**. Ceux-ci sont indispensables pour relier les composants non visuels et les composants visuels orientés bases de données.

5.5.1 Présentation tabulaire d'une table ou d'une requête : TDBGrid

Nous donnons ici un exemple de présentation d'une table mais ce qui suit s'applique à tout composant de type **TDataSet**, une requête par exemple. Le composant **TDBGrid** présente les données sous leur forme la plus naturelle, celle d'un tableau où chaque colonne correspond à un champ et chaque ligne à un tuple.

Voici la marche à suivre pour créer une représentation d'un **TDataSet** :

- Placer un composant **TDBGrid** sur la fiche
- Associer la propriété **DataSource** une source de données présente dans le module de données. Attention à l'inclusion des fichiers entêtes.

Aussitôt cette dernière opération réalisée, la grille se remplit des données présentes dans la relation. Si rien ne se passe, il faut vérifier la propriété **`Active`** de la relation à représenter. Le tableau crée autant de lignes et de colonnes qu'il y a de tuples et de champs. Chaque colonne possède un titre dont la valeur est prédéfinie au titre du champ. Voici un exemple de composant **`TDBGrid`** :

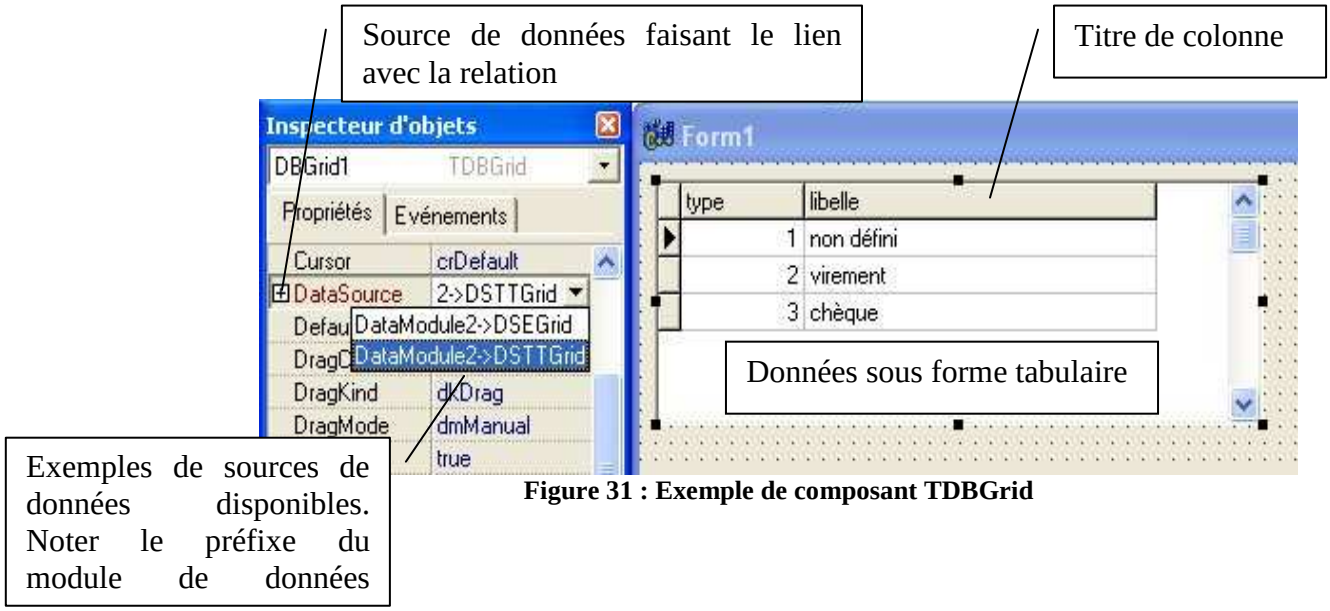


Figure 31 : Exemple de composant TDBGrid

La propriété la plus importance d'un composant **`TDBGrid`** est **`Columns`** qui est principalement constituée d'un tableau de **`TColumn`** dont chaque élément est représentatif d'une colonne du tableau.

A sa création, un composant **`TDBGrid`** est constitué de colonnes dynamiques, c'est-à-dire de l'ensemble des champs de la relation avec les formats prédéfinis. Il est toutefois possible de créer des colonnes statiques permettant, entre autres des présentations plus sophistiquées des données. Toutefois, il faut garder à l'esprit que les colonnes dites dynamiques évoluent en même temps que l'ensemble de données. Cela est particulièrement utile lors de l'ajout ou de la suppression de champs. Il faut modifier « à la main » les colonnes statiques.

Le meilleur moyen de créer des colonnes statiques consiste à double cliquer sur la grille pour faire apparaître la liste des colonnes statiques initialement vide. On peut ensuite à partir du menu contextuel obtenu par clic droit « Activer tous les champs » pour ajouter à cette liste, l'ensemble courant des champs de la relation.

Les différents boutons de la fenêtre des colonnes statiques permettent de supprimer celles que l'on ne veut pas afficher, ou bien d'ajouter des colonnes si l'on a modifié la structure de la table ou que l'on a précédemment supprimées. Pour changer l'ordre des champs, il suffit de les faire glisser à la position souhaitée.

Une fois la liste des colonnes statiques créée, il est possible de manipuler chaque colonne dans l'inspecteur d'objet comme le montre la figure suivante :

Liste des colonnes statiques

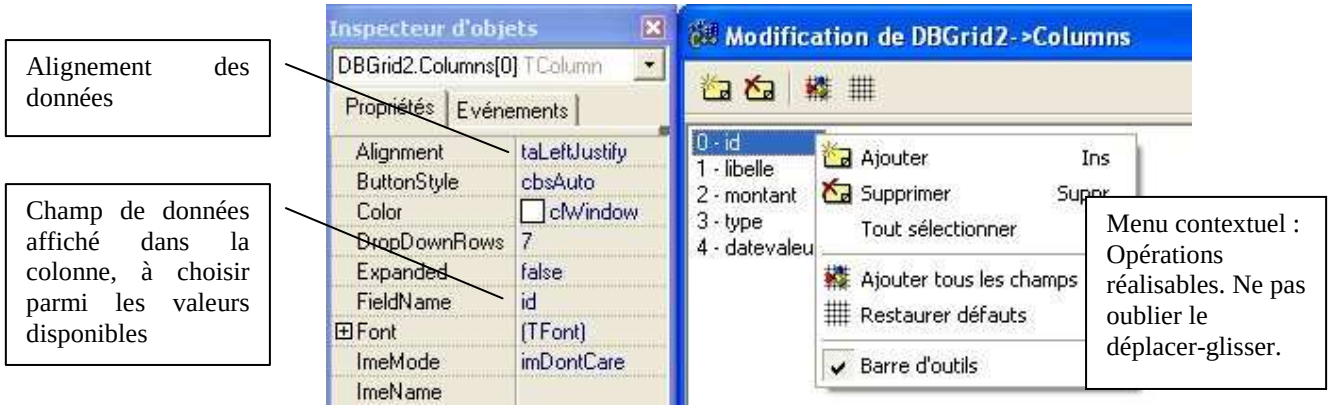


Figure 32 : Manipulation des colonnes d'un TDBGrid

La propriété **`Title`** est très complexe et comprend tout ce qui est nécessaire à la manipulation d'un titre, y compris une propriété **`Font`** permettant de spécifier un style de caractères différent de celui de la colonne. En particulier, il est possible d'aligner différemment les données de la colonne et leur titre ou de choisir une police de caractères différente. Initialement, le champ **`Title`** reprend le nom du champ de données, ce qui n'est pas toujours agréable.

A titre d'exercice, changez le titre des colonnes de manière à ce qu'il reprenne le nom du champ en gras et en minuscules.

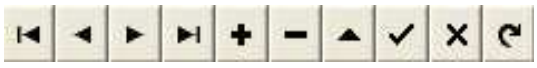
5.5.2 Les autres contrôles

Les autres contrôles permettent essentiellement de modifier ou d'ajouter de nouveaux éléments dans la base de données. Par exemple, les composants listes orientées bases de données autorisent l'utilisateur à changer la valeur d'un champ dans un tuple existant ou en cours de création.

A l'usage, il s s'avèrent relativement dangereux car ils effectuent des opérations d'insertion ou de prise en compte automatique des modifications particulièrement « nocives ». Nous ne préconisons l'usage que de deux composants de saisie orientés données :

- **`TDBLookupComboBox`** (étudié un peu plus loin)
- **`TDBnavigator`**

Les composants navigateur ou **`TDBNavigator`** permettent à un utilisateur de se déplacer dans un ensemble de données et d'y effectuer des modifications si cela est possible. Sous la forme la plus complète, ce composant se présente comme suit :



Les actions réalisables sont de gauche à droite : Aller au premier tuple, Tuple précédent, Tuple suivant, Dernier tuple, Supprimer le tuple courant, Modifier les données, Prendre en compte les modifications, Annuler les modifications et Rafraîchir les données.

5.5.3 Liste et fonctions des composants orientés bases de données

A FAIRE
Ils se trouvent sur la palette « ContrôleBD ».

5.6 Synthèse sur l'utilisation des bases de données avec le BDE

Avant de passer aux choses sérieuses : avec notamment des exemples de programmation. Il est nécessaire de rappeler comment utiliser une base de données :

1. Créer un alias de base de données
2. Créer un module de données
3. Ajouter les composants des relations qui vous intéressent
4. Ajouter les composants non visuels **`TDataSource`**
5. Ajouter les composants visuels orientés bases de données sur la fiche

5.7 Manipulation élémentaire sur les bases de données

C++ Builder autorise en direct quelques opérations élémentaires de l'algèbre relationnelle, à savoir :

Opération	Vocabulaire et commande C++ Builder
Jonction	Ajout de champ « Référence » dans un ensemble de données
Sélection	Filtrage ou création de relations Maître/Détail
Projection	Création de colonnes statiques et suppression des champs non désirés dans les **`TDBGrid`** (déjà vu)

5.7.1 Réalisation de jonctions

Dans notre exemple, nous aimerions voir apparaître en toutes lettres le type de transaction pour les écritures en lieu et place de leurs numéros de références dans la représentation tabulaire de la relation *écritures*. Pour cela, nous allons ajouter des champs dits de référence dans la relation. Ceux-ci ne sont pas autre chose que des jonctions limitées.

L'accès aux champs d'un **ensemble de données** (**`TTable`** par exemple) est faussement similaire à celui des colonnes d'un **`TDBGrid`**, aussi un avertissement solennel s'impose ! Il faut tout de suite que vous fassiez la différence entre les champs présents dans un **`TDataSet`** et la liste des colonnes de présentation présentes dans un **`TDBGrid`**. Le premier représente la liste des champs présents dans l'ensemble de données, que ce soient des champs **endogènes** (présents dans le fichier pour une table ou dans la définition pour une requête) ou des champs **exogènes** ajoutés par jonction ou calcul. Le second ne définit que des options d'affichage de données. Le problème vient du fait que le vocabulaire est très similaire.

Une fois de plus, il nous faut distinguer entre champs dynamiques et statiques. Afin d'accéder à la liste des champs statiques présents dans un ensemble de données, il est nécessaire de double cliquer sur son icône, ce qui finit d'accentuer la confusion possible avec la liste des colonnes d'un **`TDBGrid`**.

Le menu contextuel possède deux options fort intéressantes :

- « Ajouter des champs » permet d'ajouter dans la liste des champs statiques un champ de donnée présent dans la structure du **`TDataSet`**.
- « Nouveau champ » permet d'ajouter un nouveau champ par jonction ou calcul.

Précisons immédiatement que nous aurons le droit d'ajouter plusieurs types de champs :

- Les champs de données sont les champs endogènes de la table, cette option ne présente que peu d'intérêt car vous pouvez faire la même chose avec le menu « Ajouter champs »
- Les champs calculés permettent d'effectuer des opérations arithmétiques ou lexicographiques sur les différents champs. Par exemple, vous pourrez effectuer une addition entre deux champs numériques ou une concaténation de deux champs chaînes de caractères.
- Les champs de référence permettent d'effectuer une jonction c'est-à-dire une mise en correspondance des valeurs de deux champs dans deux tables différentes avec extraction d'un résultat.

Ajoutons un champ nommé TYPE_TRANSACTION spécifiant le type de transaction à partir de son numéro. La jonction doit s'effectuer entre les champs TYPE de « type_transaction » et TYPE de « ecritures » avec LIBELLE de « type_transaction » pour résultat. Pour parler avec le langage des bases de données, TYPE est la clef primaire de la relation « type_transaction » alors que TYPE est une clef étrangère (ou externe) de « ecritures ».

Il faut tout d'abord choisir « Nouveau champ » dans le menu contextuel. La Figure 33 décrit la boîte de dialogue qui s'affiche.

Les premières précisions à apporter sont :

- **Le nom du champ** que vous désirez ajouter. C++ Builder gère pour vous le nom du composant **`TField`** généré pour représenter votre nouveau champ.
- **Le type de données du champ** et éventuellement sa taille. Dans le cas d'un champ calculé tout type et autorisé. Dans le cas d'un champ de données ou dans le cas d'un champ référence en provenance d'un autre ensemble de données, il faut faire attention à n'utiliser que des types compatibles avec les données sources, par exemple, des données numériques entre elles
- **La taille éventuelle du champ.**

Continuons notre jonction en choisissant de créer un champ de type **référence**. Il faut préciser les éléments suivants :

- Le Champs clé : c'est le champ de jonction dans la table courante (clé étrangère)
- L'ensemble de données : c'est la table sur laquelle on effectue la jonction
- Clés de référence : c'est le champ de jonction dans la table externe (clé primaire)
- Champ résultat : champ résultat de la jonction (dans la table externe désignée par l'ensemble de données)

Le champ nouvellement créé peut alors se traiter comme s'il avait toujours appartenu à la table (champ endogène) à quelques exceptions près, comme, par exemple, le filtrage.

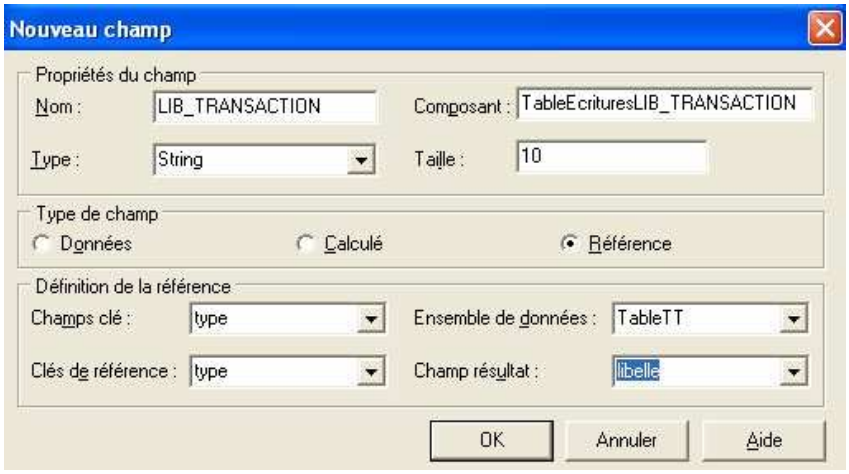


Figure 33 : Réalisation d'une jonction

Attention, si vous obtenez un message d'erreur concernant un ensemble de données ouvert, il vous faudra recommencer l'opération en pensant à « fermer » le composant auquel vous voulez ajouter un champ. Pour cela, il suffit de mettre sa propriété **`Active`** à *false*. Il faudra la réactiver après l'ajout du champ.
Si vous obtenez une erreur concernant un champ clé introuvable, pensez à l'ajouter à la liste des champs de la relation.

5.7.2 Le filtrage

Le filtrage est une opération qui permet de restreindre les tuples d'une table à ceux qui respectent une certaine condition nommée Filtre. Pour activer un filtre, il faut spécifier dans la propriété **`Filter`** une chaîne de caractères spécifiant les critères de filtrage et positionner la propriété **`Filtered`** à **`true`**. Ces propriétés étant modifiables à tout instant, cette opération pourra s'effectuer très facilement et sous le contrôle d'éléments d'interface.

Pour obtenir des exemples de filtrage, on pourra consulter l'aide en ligne de
`TDataSet::Filter`

Seuls les champs endogènes d'un ensemble de données peuvent être filtrés.

5.7.3 Création de fiches Maître/Détail

Supposez désormais que vous souhaitiez accéder aux données d'un ensemble en fonction du tuple sélectionné dans une grille. Le meilleur exemple est sans doute celui des nomenclatures où vous disposez des trois tables.

- La table « produits finis » représente l'ensemble des produits complexes
- La table « matières premières » représente l'ensemble des composants élémentaires que vous assemblez
- La table « composition » associe sur une ligne, un produit complexe, un composant simple et le nombre de composants simples entrant dans la composition du produit complexe.

Vous pouvez alors décider d'afficher dans un premier **`TDBGrid`** l'ensemble des produits complexes, puis, dans un second, la liste des pièces le composant. Il s'agit typiquement d'une liaison maître/esclave. La table « produits finis » (le maître) conditionnant l'accès aux données de « composition » (l'esclave).

Une fois les données créées, voici la marche à suivre :

- Préparation de la visualisation de la relation maître / esclave
  - Ajouter un nouveau composant **`TTable`** et une source de données à la relation « composition »
  - Ajouter un champ NOM_MATIERES à la table « composition » par jointure sur la table « matieres premières »
  - Associer une grille à la table des compositions et ne présenter que le nom de la matière et la quantité.
- Mise en place de la relation Maître / Détail. Celle-ci se fait essentiellement sur les propriétés de la table esclave
  - Sélectionner la propriété **`MasterSource`** de la **`TTable`** « composition » et lui donner pour valeur la source de données associée à la **`TTable`** maître.
  - Utiliser l'éditeur spécial pour choisir les **`MasterFields`**, opération qui est similaire à une jointure entre un champ de la table maître et un champ de la table esclave.
    - Choisir un index de « composition » pour obtenir le champ joint côté esclave (celui qui dénote le numéro du produit fini). Notez au passage que les relations maître/esclave ne peuvent avoir lieu sans indexation correcte de la table esclave.
    - Choisir le champ de jointure côté « produits finis » dans la case de droite (par exemple, la clé primaire)
    - Cliquer sur Ajouter et le tour est joué.

Voici une figure représentant l'éditeur spécial pour cet exemple :

Figure 34 : Editeur spécial pour la liaison Maître/Esclave

En guide d'exercice, recommencer « de mémoire » le processus de création d'une nouvelle base de données. Il n'est pas nécessaire de créer un nouvel alias ou une nouvelle connexion ODBC. Vous avez du remarquer qu'il était possible d'utiliser directement un connecteur ODBC dans un composant **`TTable`**. On peut aussi surcharger les paramètres d'un DSN ODBC pour le faire pointer vers une autre base. Pour cela, insérez tout d'abord un composant

TDabaBase. Les composants **TTable** devront pointer vers ce dernier et non pas directement sur le connecteur ou l'alias. Cliquez ensuite sur le bouton « Défauts » pour obtenir tous les paramètres avec leurs valeurs par défaut. Modifiez enfin ce qui vous intéresse.

5.7.4 Ajout d'un tuple dans une ou plusieurs tables

Pour ajouter un tuple dans une table, il suffit de passer la table en **mode insertion**, ce qui a pour effet de créer un tuple vierge. Dans un deuxième temps, on modifie les valeurs des champs par une instruction du style (méthode) :

```
IdentificateurTable->FieldByName("nom du champ") = valeur ;
```

Ou par la propriété :

```
IdentificateurTable->FieldValues["nom du champ"] = valeur ;
```

L'identificateur de table doit contenir un pointeur sur un composant **TDataSet**, une **TTable** par exemple ; éventuellement précédé par un pointeur sur le module de données.

Cette opération peut être réalisée directement en affectant un contrôle de données à un champ. Par exemple, une boîte d'édition de données **TDBEdit** est connectée à un champ endogène d'une table via sa source de données. Cependant, cette opération est assez dangereuse et il vaut mieux utiliser des champs simples et confier à un bouton genre « Valider » le soin d'incorporer des données dans la base.

Deux commandes principales permettent de passer en mode insertion : **Append** et **Insert**. Il est toujours préférable d'utiliser la première qui est moins traumatisante pour la base de données. En effet, il n'est guère intéressant de rajouter des tuples au milieu d'une base lorsque l'on peut les obtenir dans l'ordre que l'on veut en utilisant un index approprié, n'est ce pas ?

Le composant **TDBEdit** est fort simple. Il suffit de le connecter sur le bon champ via une source de données et un champ de cette source.

En exercice, ajouter un bouton qui permet d'accéder à l'insertion d'un nouveau produit dans la table « produits_finis ». L'insertion se fait par un composant **TDBEdit** et deux autres boutons : l'un pour valider (méthode **Post()** de **TTable**), l'autre pour annuler (méthode **Cancel()** de **TTable**).

Certains contrôles permettent de sélectionner des champs en effectuant une jonction, c'est le cas, par exemple des deux composants **TDBLookupListBox** et **TDBLookupComboBox**.

La **TDBLookupListBox** est plus difficile à utiliser car elle utilise deux ensembles de données différents mais devant appartenir à la même base de données. Voici les principales propriétés :

- Les propriétés **DataSource** et **DataField** sont respectivement reliées à la source de données et au champ dans lequel on souhaite écrire une valeur.
- Les propriétés **ListSource** et **ListField** sont liées à la source de données et au champ à lister.
- La propriété **KeyField** est un champ en provenance de **ListSource** qui correspond à la seconde partie de la jonction. Sa valeur sera recopiée dans celle de **DataField**. Ainsi, c'es **ListField** qui est présenté et **KeyField** la valeur qui est recopiée dans le champ en cours de modification.

Il est possible de désactiver temporairement l'affichage des contrôles orientés bases de données grâce à la méthode **`DisableControls()`** de l'ensemble de données (la méthode **`EnableControls()`** permet de les réactiver).

6. Piloter le port Série

Je ne sais pas comment piloter le port série avec Borland C++ Builder. Cependant, voilà quelques pistes que j'ai pu trouver. Aucune n'a été testée car je n'ai pas le matériel nécessaire.

6.1 Appel à l'API Windows

Les fonctions de windows.h qui peuvent servir sont les suivantes :

- **`CreateFile()`**
- **`GetCommState()`**
- **`SetCommState()`**
- **`WriteFile()`**
- **`SetCommTimeouts()`**
- **`ReadFile()`**
- **`CloseFile()/CloseHandle()`**

6.2 Utiliser des composants dédiés

Et voir comment, ils font !!

6.2.1 Composant ComPort Library

Ce composant est disponible sur SourceForge. Deux pages parlent de son installation et de son utilisation.

http://sourceforge.net/projects/comport/
http://petit.developpez.com/serie/install_tcomport/
http://petit.developpez.com/serie/cours_tcomport/

6.2.2 Composant AsyncPro

Il est lui aussi disponible sur SourceForge

http://sourceforge.net/projects/tpapro/