1 La programmation Delphi

1.1. Présentation de DELPHI

1.1.1 Philosophie

Environnement de développement « visuel » :

- permet de réaliser des applications Windows :
  - sur Windows 98, NT et 2000,
  - simple et rapide avec son EDI (sans définir ni variable, ni méthode) par simples « click » :
  - pour créer une interface graphique d'application,
- facilite le codage (il n'y a plus qu'à remplir des zones de texte),
- met tous les outils accessibles par menus et icônes,
- utilise le langage Object Pascal = dérivé orienté objet de Pascal 7,
-

Développement rapide d'applications :

- Delphi est un « RAD »,
- Propose une bibliothèque importante de composants prédéfinis => PPO,
- Permet l'élaboration d'interfaces graphiques complexes par le jeu de la construction,
- Les composants proposés :
  - sont des VCL (visual composant library) compilés pour un emploi direct (à rapprocher des VBX sous Visual Basic),
  - encapsulent quasiment complètement l'accès à Windows : ils cachent au programmeur toutes les difficultés inhérentes à ce type de programmation (manipulation des différents handles, DC, paramètres, événements, etc ...).

Langage Objet :

- Basé sur un véritable langage objet,
- Permet de créer ses propres composants,
- De modifier le comportement des composants fournis.
- Permet de réaliser la "programmation orientée objet" (POO).

- Delphi permet la réalisation d'applications Windows sans avoir à connaître au préalable les mécanismes fondamentaux de la programmation orientée objets.
- On peut aborder ce type de programmation ultérieurement, lorsque la prise en main du produit est déjà réalisée, que le besoin s'en fait sentir… ou jamais.

L4G :

- s'approche des langages de 4ème génération (L4G),
- en plus, permet de créer des applications exécutables compilées,

Accès aux bases de données :

- Delphi propose un moteur de système de gestion de base de données (le même que celui du SGBD/R "micro" Paradox du même éditeur) appelé BDE (Borland Database Engine).
- Ce moteur permet de créer, modifier et gérer très facilement des "tables" contenant des données, avec toutes les fonctions de contrôles de données évolués propres aux SGBD.

- De fait, Delphi permet de constituer des applications de gestion de bases de données puissantes au même titre que les progiciels spécialisés actuels (tels Paradox ou Access) tout en offrant des possibilités de programmation nettement plus importantes.
- Un utilitaire spécifique permet de réaliser et de tester des requêtes SQL qu'il est ensuite possible d'intégrer directement au code.
- Dans sa version professionnelle, Delphi permet de réaliser des applications "client/serveur" en utilisant un véritable serveur SQL plus complexe, appelé Interbase, ou en accédant aux données gérées par les principaux serveurs SQL du monde professionnel.

L'aide :

- Il est livré avec plusieurs "aides en ligne" au format Windows (.hlp) - écrites en français - pratiquement indispensables pour réaliser efficacement les applications,
- Les aides contiennent l'ensemble des informations nécessaires (types de paramètres, type de retour, …).

1.1.2 Principes

Les applications DELPHI fonctionnent en mode événementiel :

- Les différentes fonctionnalités du programme sont activées lorsque certains "événements" surviennent (et seulement lorsque ceux-ci surviennent).
- Ces événements sont déclenchés par le système mais ils sont plus généralement activés par l'utilisateur par ses actions sur la souris ou au clavier.

Delphi permet de réaliser "à la souris" des applications complètes. On parle alors de "développement visuel d'applications".

La simplicité de développement n'est qu'apparente :

- Il est effectivement possible de créer certaines applications sans avoir à écrire une seule ligne de code,
- Il ne faut pas oublier que le développement d'applications plus ambitieuses reste complexe et parfois fastidieux.
- Delphi aide le programmeur dans toutes les phases du développement mais Delphi s'adresse avant tout à des programmeurs,
- Il est illusoire de chercher à créer quoi que ce soit si l'on pense pouvoir s'abstraire des règles de programmation habituelles :
  - analyse préalable conséquente,
  - respect de règles précises,
  - qualité de la programmation,
  - un effort d'apprentissage conséquent permettant une utilisation optimale des différentes ressources, utilitaires, etc ...

Delphi peut donc être vu comme un "piège à programmeurs du dimanche" qui croient que la facilité d'emploi d'un environnement définit la puissance et la qualité du logiciel réalisé. Les désillusions surviennent rapidement…

1.1.3 Mode de programmation

Programmer une application graphique qui fonctionne en mode événementiel est très différent de réaliser une application classique selon les règles strictes de la programmation structurée :

- en mode événementiel, il faut réaliser des fonctionnalités pratiquement indépendantes les unes des autres sans qu'il soit possible de définir l'ordre dans lequel elles seront exécutées.

- Le nombre de fonctionnalités est souvent très important et peut être difficile à appréhender dans son ensemble.

Il est donc prudent :
de commenter ses sources,
d'utiliser certaines règles de dénomination des différentes variables
pour que l'ensemble du projet ne devienne pas difficile, voire impossible, à gérer.

A l'intérieur de chaque fonctionnalité, la programmation structurée reprend ses droits.
L'algorithmie de chaque fonctionnalité peut aller de simple à complexe.

1.1.4 Les versions

Delphi existe en trois versions :

- La version standard (S) :
  - Environnement de développement complet,
  - N'offre pas tous les outils et composants.
- La version professionnelle (P) :
  - contient davantage d'outils.
  - Permet de :
    - programmer des applications Internet et Intranet,
    - Permet de développer des contrôles ActiveX,
  - Offre plus de fonctions de base de données.

- La suite Client/Serveur (C/S) :
  - Permet le développement des applications client-serveur complètes,

	Environnement de développement à la pointe de la technologie	Delphi STD	Delphi Pro	Delphi Ent
	CodeInsightTM : modèles de code, complément de code et de paramètres	X	X	X
	Exploration de projet permettant de mieux comprendre votre code et parcourir la VCL		X	X
	Cadres pour construire et réutiliser des composants composites		X	X
	Concepteur de module de données avec vues arborescentes et diagrammes de données		X	X
	Expert Panneau de configuration pour personnaliser les propriétés de vos applications		X	X
	Débogage évolué			
	Conseils, actions et groupes de points d'arrêt pour un meilleur contrôle	X	X	X
	inspecteur de débogage et vue d'inspecteur		X	X
	Vue FPU et prise en charge MMX pour un débogage de bas niveau amélioré		X	X
	Débogage distant et interprocessus pour un développement distribué évolué		X	X
	Bibliothèque de composants visuels avec une réutilisabilité maximale			
	Plus de ____ composants standard	85+	150+	200+
	Catégories de propriétés pour améliorer l'apprentissage et la productivité	X	X	X
	Expert d'applications console pour écrire rapidement des applications simples	X	X	X
	WebExtra : prise en charge des images JPEG et de la compression ZLib		X	X
	Amélioré : éditeur de propriétés et code source de la VCL		X	X
	Composant Internet Explorer pour l'intégration d'un navigateur		X	X
	TeamSourceTM pour l'adaptation du RAD à toute votre équipe de			X

	développement			
	Compléments au développement			
	CD-ROM de JBuilderTM 2 et C++BuilderTM 3		X	X
	InstallShield Express et Resource Workshop		X	X
	Développement à haute productivité pour Internet			
	Internet Explorer	X	X	X
	WebBroker : livrez les applications de bases de données Web les plus rapides		X	X
	ActiveForms pour la construction d'applications Web		X	X
	Composants Internet natifs FastNet (ftp, smtp, pop3, http, etc)		X	X
	InternetExpressTM : clients légers complets pour le Web			X
	prise en charge de la prévisualisation HTML 4 plus expert d'application Web			X
	MIDASTM : DataBroker distant pour partitionner facilement les applications			X
	prise en charge évoluée de fournisseur et résolveur maître/détail			X
	Données XML pour simplifier l'échange de données			X
	Clients Dynamic HTML 4/XML complets pour le Web			X
	Outils de développement de bases de données intégrés : connectez-vous à toutes vos données d'entreprise			
	Pilotes Access, FoxPro, Paradox et dBASE et connectivité ODBC		X	X
	Prise en charge d'InterBase 5.5 et d'InterBase local		X	X
	InterBase NT (licence 5 utilisateurs) pour développer et tester des applications SQL multi-utilisateurs		X	X
	Liaison de données de contrôle ActiveX et expert d'objet COM		X	X
	Prise en charge de MS SQL Server			X
	ADOExpress pour l'accès à tous les types d'informations			X
	Prise en charge d'Oracle8i avec champs imbriqués (ADT), champs tableaux et champs références			X
	tableau croisé Decision Cube et code source de Decision Cube			X
	Pilotes SQL Links natifs pour Oracle, SQL Server, InterBase, Sybase, Informix et DB2			X

1.1.5 Installation

La version de Delphi 3 peut occuper jusqu'à 110 Mo en installation complète,
La version de Delphi 5 peut occuper jusqu'à 190 Mo en installation complète,

Fichiers programme : 125 Mo,
Fichiers partagés : 48 Mo,
BDE et SQL : 13 Mo,
Module base de données : 4 Mo

1.2 L ’environnement de développement

1.2.1 L’interface graphique

L'environnement de développement intégré comporte :

- une barre de menu,
- une barre d'icônes,
- des fenêtres permettant de réaliser la totalité des opérations de développement.

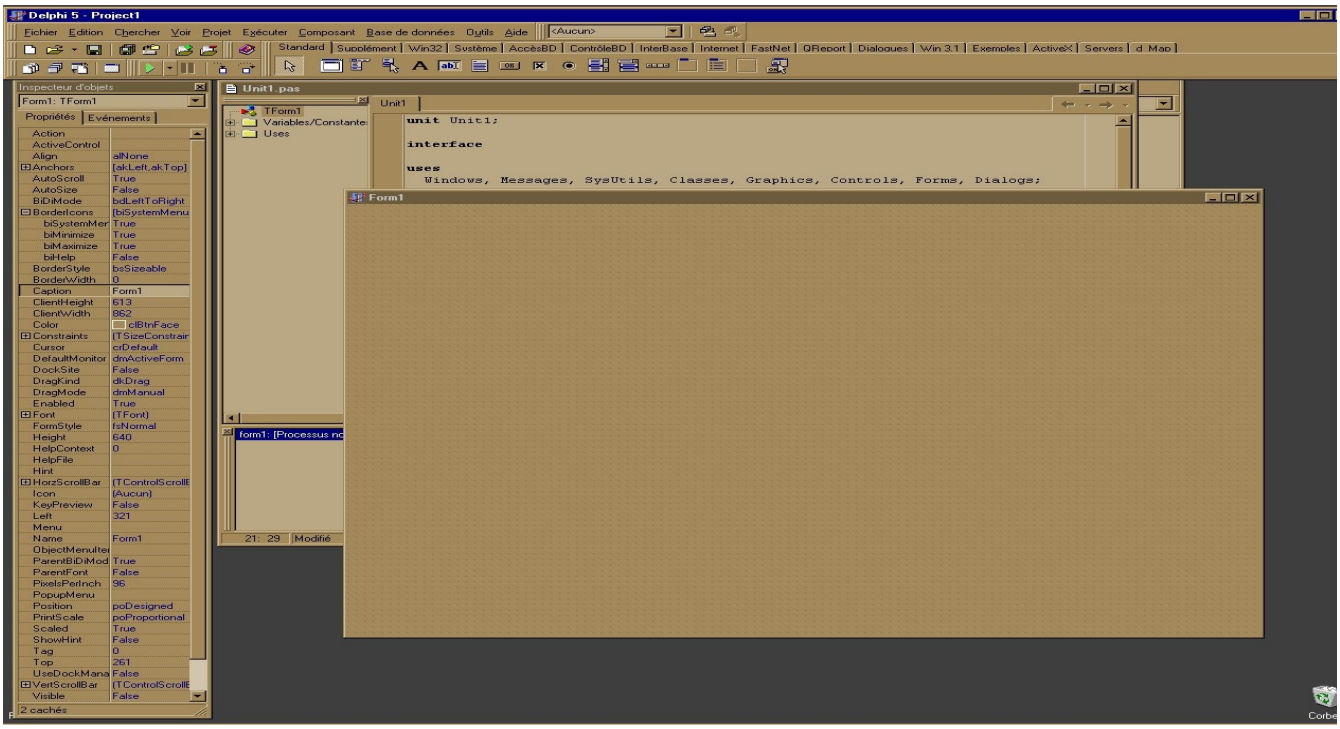
Description de l'écran principal

L'écran principal se compose de 4 fenêtres distinctes et indépendantes :

- Une fenêtre contenant les menus et les différentes icônes ;
- Une fenêtre "inspecteur d'objet" dont le rôle sera précisé ultérieurement ;
- Une fenêtre dans laquelle se réalisera la construction de l'interface graphique ;
- Une fenêtre, en grande partie cachée par la fenêtre précédente, qui est l'éditeur de code.

Composition de l'écran de développement :

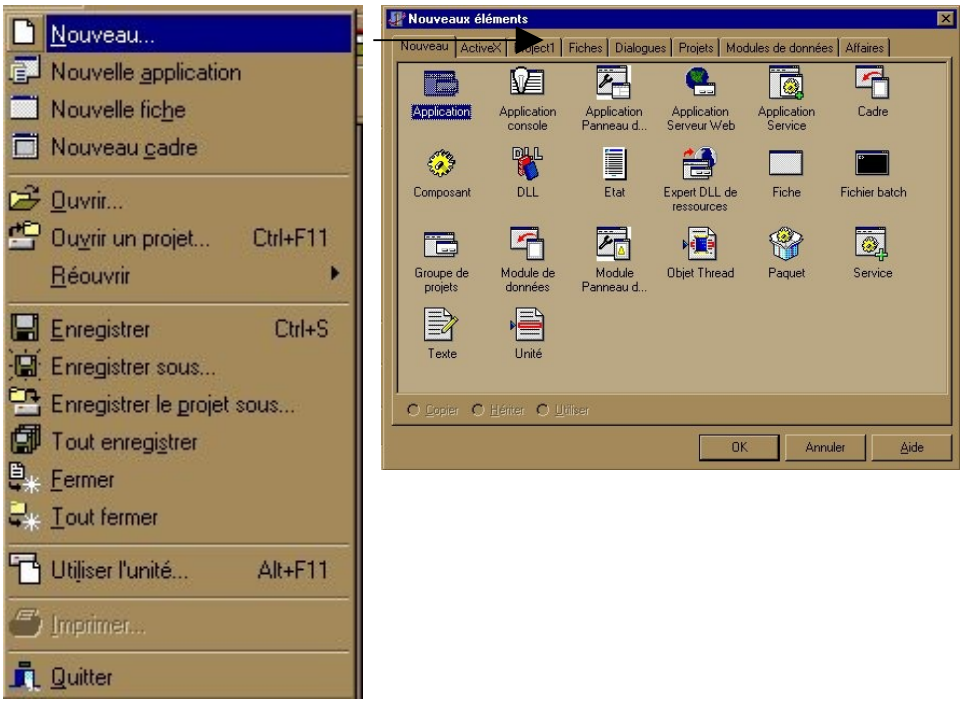
En haut la barre de menu et la barre d'icônes, à gauche, l'inspecteur d'objet, à droite la fenêtre de création de l'interface graphique qui cache en partie l'éditeur de code.



1.2.2 La Barre « Menu »

La barre de menu permet d'accéder à l'ensemble des fonctionnalités de l'environnement de développement. Les fonctions les plus utilisées se retrouvent, sous forme d'icônes dans la partie gauche de la barre d'icônes.

1.2.2.1 Menu " Fichier"

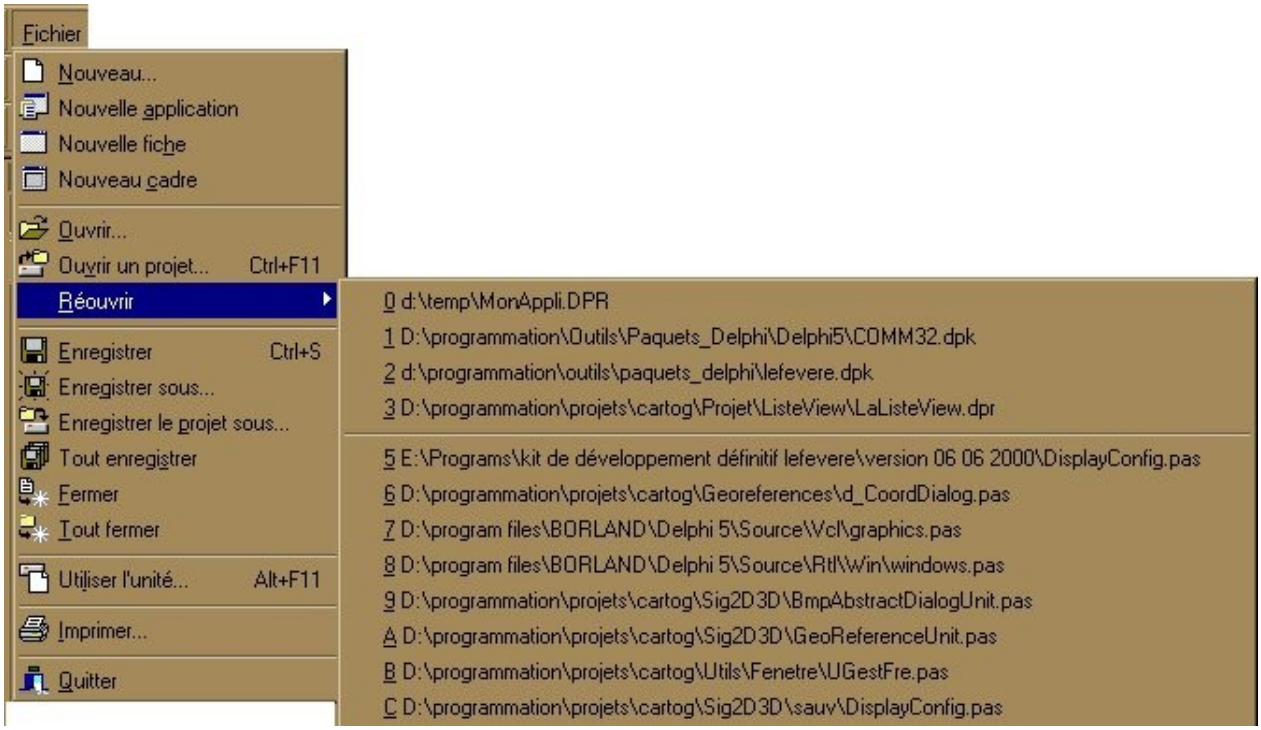


Le menu **Fichier** permet d'ouvrir, d'enregistrer, de fermer et d'imprimer des projets et des fichiers existants ou nouveaux. Il permet aussi d'ajouter de nouvelles fiches et de nouvelles unités au projet ouvert. Le menu Fichier comporte les commandes suivantes :

Commandes	**Description**
Nouveau	Ouvre la boîte de dialogue Nouveaux éléments contenant les nouveaux éléments pouvant être créés.
Nouvelle application	Crée un nouveau projet contenant une fiche, une unité et un fichier .DPR.
Nouvelle fiche	Crée et ajoute une fiche vierge au projet.
Nouveau cadre	Crée un cadre « FrameX », conteneur de composants (BD, images, …)
Ouvrir	Utilise la boîte de dialogue Ouvrir pour charger dans l'éditeur de code un projet, une fiche, une unité, ou un fichier texte existant.
Ouvrir un projet	Utilise la boîte de dialogue Ouvrir pour charger un projet existant.
Réouvrir	Affiche un menu en cascade contenant la liste des projets et modules qui viennent d'être fermés.
Enregistrer	Enregistre le fichier en cours sous son nom actuel.
Enregistrer sous	Enregistre le fichier en cours sous un nouveau nom, en incluant les modifications apportées aux fichiers du projet
Enregistrer projet sous	Enregistre le projet en cours sous un nouveau nom.
Tout enregistrer	Enregistre tous les fichiers ouverts (ceux du projet et des modules en cours).
Fermer	Ferme la fenêtre active et le fichier unité associé.

Tout fermer	Ferme tous les fichiers ouverts.
Utiliser unité	Ajoute l'unité sélectionnée à la clause uses du module actif.
Imprimer	Dirige le fichier actif vers l'imprimante.
Quitter	Ferme le projet ouvert et quitte Delphi.

Un projet est un ensemble de fichiers constituant une application. Au démarrage, Delphi ouvre un nouveau projet par défaut. On peut utiliser ce projet, ouvrir un projet existant ou lancer un projet différent en utilisant l'un des modèles de projet prédéfinis de Delphi



1.2.2.2 Menu "Edition"

Les commandes du menu Edition permettent de manipuler du texte et des composants lors de la phase de conception. Le menu Edition comporte les commandes suivantes :

Commandes	**Description**
(Défaire	Annule la dernière action effectuée ou récupère la dernière suppression
Refaire	Restitue la dernière action annulée
Couper	Supprime l'élément sélectionné et le place dans le Presse-papiers
Copier	Place une copie de l'élément sélectionné dans le Presse-papiers et laisse l'original à sa place
Coller	Copie le contenu du Presse-papiers dans l'éditeur de code, dans la fiche active ou dans le menu actif.
Supprimer	Supprime l'élément sélectionné
Tout sélectionner	Sélectionne tous les composants de la fiche
Aligner sur la grille	Aligne les composants sélectionnés sur le point de grille le plus proche
Mettre en avant-plan	Place le composant sélectionné à l'avant-plan
Mettre en arrière-plan	Place le composant sélectionné à l'arrière-plan
Aligner	Aligne les composants
Taille	Redimensionne les composants
Echelle	Redimensionne tous les composants de la fiche

Ordre de tabulation	Modifie l'ordre de tabulation des composants sur la fiche active
Ordre de création	Modifie l'ordre de création des composants non visuels
Basculer l'enfant	Inverser l'alignement Lecture gauche vers droite => droite vers gauche
Verrouiller contrôles	Fixe les composants de la fiche à leur position actuelle
Ajouter à l'interface	Définit une nouvelle méthode, un nouvel événement ou une nouvelle propriété pour un composant ActiveX.

1.2.2.3 Menu "Chercher"

Les commandes du menu Chercher permettent de localiser du texte, des erreurs, des objets, des unités, des variables et des symboles dans l'éditeur de code.

Commandes	Description
Chercher	Recherche le texte spécifié, et met en surbrillance la première occurrence trouvée dans l'éditeur de code.
Chercher dans les fichiers	Recherche le texte spécifié, affiche chaque occurrence dans une fenêtre en bas de l'éditeur de code.
Remplacer	Recherche le texte spécifié et le remplace par le nouveau texte
Occurrence suivante	Répète la recherche
Recherche incrémentale	Recherche du texte au fur et à mesure que vous le saisissez
Aller à ligne	Place le curseur sur le numéro de ligne spécifié
Erreur d'exécution	Recherche la dernière erreur d'exécution
Scruter symbole	Recherche le symbole spécifié

1.2.2.4 Menu "Voir"

Les commandes du menu Voir permettent d'afficher ou de cacher certains éléments de l'environnement Delphi et d'ouvrir des fenêtres appartenant au débogueur intégré.

Commandes	Description
Gestionnaire de projet	Affiche le gestionnaire de projet
Inspecteur d'objets	Affiche l'inspecteur d'objet
Liste à faire	Affiche la liste enregistrée des tâches à effectuer par les développeurs
Palette d'alignement	Affiche la palette d'alignement
Explorateur	Affiche le scruteur d'objet ‘globaux’, de ‘classes’ et des ‘unités’
Explorateur de code	Affiche l’éditeur de texte
Liste de composants	Affiche la boîte de dialogue Composants
Liste de fenêtres	Affiche la liste des fenêtres ouvertes
Fenêtre de déboguage	
Bureau	
Basculer Fiche/Unité	Bascule entre la fiche et la fenêtre unité
Unités	Affiche la boîte de dialogue Voir unité
Fiches	Affiche la boîte de dialogue Voir fiche
Bibliothèque de types	Affiche la fenêtre de l'éditeur de bibliothèques de types
Nouvelle fenêtre d'édition	Ouvre une nouvelle fenêtre de l'éditeur de code
Barre d'outils	Affiche ou cache la barre d'outils
Points d’arrêt	Affiche la fenêtre Points d’arrêt
Pile d'appels	Affiche la fenêtre Pile d'appels
Points de suivi	Affiche la fenêtre Liste des points de suivi
Variables locales	Affiche la fenêtres des variables locales avec leur valeur
Threads	Affiche la fenêtre Etat thread
Modules	Affiche la fenêtre Modules
Journal d’évènements	
CPU	Présente un aspect de bas niveau différent de l'exécution de l'application : Le volet de désassemblage affiche les instructions assembleur désassemblées à partir du code machine de votre application.

	Le volet d'affichage de la mémoire affiche toute portion de la mémoire accessible au module exécutable en cours de chargement. Par défaut, la mémoire est présentée sous forme d'octets hexadécimaux. Le volet de la pile machine affiche le contenu en cours de la pile du programme. Le volet des registres CPU affiche les valeurs en cours des registres du CPU. Le volet des indicateurs affiche les valeurs en cours des indicateurs du CPU.
FPU	Affiche le contenu de l'unité virgule flottante du CPU

1.2.2.5 Menu "Projet"

Le menu Projet permet de compiler ou de construire votre application. Un projet doit être ouvert.

Commandes	**Description**
Ajouter au projet	Ajoute un fichier au projet
Supprimer du projet	Supprime un fichier du projet
Importer une bibliothèque de types	Importe une bibliothèque de types dans un projet
Ajouter au référentiel	Ajoute un projet au référentiel d'objets
Voir le source	Affiche le fichier projet dans l'éditeur de code
Ajouter un nouveau projet	Affiche la fenêtre 'nouveau' pour un choix de projet
Ajouter un projet existant	Affiche fenêtrre de dialogue permettant de choisir un projet
Compiler ' …….'	Compile tout code source ayant été modifié depuis la dernière compilation
Construire ' …….'	Compile tous les éléments du projet, que le code ait ou non été modifié
Vérifier la syntaxe	Compile le projet sans le lier.
Information	Affiche les informations de construction et l'état de la construction du projet
Compiler tous les projets	Compile tout code source ayant été modifié depuis la dernière compilation
Construire tous les projets	Compile tous les éléments des projets, que le code ait ou non été modifié
Options de déploiement Web	Effectue le paramétrage nécessaire pour déployer votre contrôle ActiveX ou votre fiche ActiveForm terminé sur votre serveur web.
Déployer pour le Web	Après le paramétrage du déploiement web et la compilation de votre projet, déploie votre contrôle ActiveX ou votre fiche ActiveForm terminé.
Options	Ouvre la boîte de dialogue Options de projet.

1.2.2.6 Menu "Exécuter"

Le menu Exécuter contient des commandes permettant de déboguer un programme depuis Delphi. Les commandes suivantes constituent les fonctionnalités de base du débogueur intégré :

Commandes	**Description**
Exécuter	Compile et exécute l'application
Attacher au processus	Affiche la liste des processus exécutés sur l'ordinateur local pour effectuer un choix

Paramètres	Spécifie les paramètres de démarrage de l'application
Recenser le serveur ActiveX	Recense le projet dans la base des registres de Windows. Disponible quand le projet en cours est un projet ActiveX.
Dé-recenser le serveur ActiveX	Supprime le projet la base des registres de Windows. Disponible quand le projet en cours est un projet ActiveX.
Pas à pas	Exécute le programme ligne par ligne, évite les appels de procédures en les exécutant comme une seule unité
Pas à pas approfondi	Exécute le programme ligne par ligne, en entrant dans les procédures et en suivant l'exécution à chaque ligne
Jusqu'à la prochaine ligne	Exécute le programme en s'arrêtant à la prochaine ligne de code exécutable
Jusqu'au curseur	Exécute le programme jusqu'à ce que le débogueur atteigne la ligne où se trouve le curseur dans l'éditeur de code
Exécuter jusqu'au retour	Le processus s'arrête sur l'instruction qui suit immédiatement l'instruction qui a appelé la fonction en cours.
Montrer le point d'exécution	Positionne le curseur au point d'exécution dans une fenêtre d'édition
Suspendre le programme	Suspend temporairement l'exécution d'un programme
Réinitialiser le programme	Stoppe l'exécution du programme et le réinitialise afin de pouvoir le relancer
Inspecter	Ouvre la fenêtre Inspecteur relative au terme en surbrillance
Evaluer/Modifier	Ouvre la boîte de dialogue Evaluation/Modification qui permet d'évaluer ou de modifier la valeur d'une expression
Ajouter point de suivi	Ouvre la boîte de dialogue Propriété du point de suivi qui permet de créer ou de modifier des points de suivi
Ajouter point d'arrêt	Spécifie des emplacement du code où l'exécution du programme doit s'arrêter

Remarque :
Les commandes du débogueur intégré deviennent accessibles lorsque les informations symboliques de débogage ont été générées pour le projet en cours.

1.2.2.7 Menu "Composant"

Les options du menu Composant sont :

Commandes	**Description**
Nouveau composant	Ouvre l'expert composant
Installer un composant	Installe un composant dans un paquet nouveau ou existant
Importer un contrôle ActiveX	Ajoute des bibliothèques de types de contrôles ActiveX à votre projet Delphi
Créer un modèle de composant	Personnalise un composant et l'enregistre en tant que modèle en lui attribuant un nom, une page de la palette et une icône
Installer des paquets	Spécifie les paquets requis par votre projet
Configurer la palette	Ouvre la boîte de dialogue Palette

1.2.2.8 Menu "Base de données"

Les commandes du menu Base de données permettent de créer, de modifier et de visualiser vos bases de données suivant les versions de Delphi.

Commandes	**Description**
Explorateur	Explorer une base de données
Expert fiche	Utiliser l'assistant pour créer une fiche BD

Choisissez Base de données|Explorateur pour ouvrir l'explorateur de bases de données ou l'explorateur SQL, selon la version de Delphi que vous possédez. Les deux permettent de créer, voir et modifier des données et des alias BDE. De plus, l'explorateur permet d'interroger des bases locales et des bases distantes.

Choisissez Base de données|Expert fiche pour ouvrir l'expert fiche de Delphi qui permet de créer une fiche comportant les données d'une base de données locale ou distante.

1.2.2.9 Menu "Outils"

Les commandes du menu Outils permettent d'accéder aux outils intégrés de Delphi ou d'exécuter des applications externes. Utilisez le menu Outils pour:

- Voir et modifier les options d'environnement
- Voir et modifier les options de l'éditeur
- Voir et modifier les options du débogueur
- Modifier la liste des programmes accessibles par le menu Outils
- Modifier les modèles et les experts
- Créer et modifier des collections de paquets
- Créer et modifier des tables de bases de données.
- Créer et modifier des images

Commandes du menu Outils:

Commandes	**Description**
Options d'environnement	Spécifie les préférences de l'éditeur, du scruteur et de la configuration. Permet également de personnaliser l'aspect de la palette des composants.
Référentiel	Affiche la boîte de dialogue Référentiel d'objets.
Configurer les outils	Affiche la boîte de dialogue Options des outils. Cette boîte de dialogue permet d'ajouter, de supprimer et de modifier les programmes du menu Outils.
Module base de données	Permet de créer, voir, trier, modifier et interroger des tables au format Paradox, dBASE ou SQL.
Editeur de collection de paquets	Permet de créer et de modifier des collections de paquets. Les collections de paquets offrent un moyen simple de rassembler des paquets et leurs fichiers associés pour les distribuer aux autres développeurs.
Editeur d'image	Permet de créer et de modifier des fichiers ressource, des icônes, des bitmaps et des fichiers curseur afin de les utiliser dans les applications Delphi.

Les différentes commandes n'apparaissent que si les utilitaires correspondants ont été sélectionnés lors de la phase d'installation.

1.2.2.10 Menu "Aide"

Le menu Aide permet d'accéder au système d'aide en ligne qui s'affiche dans une fenêtre d'aide spéciale. Le système d'aide donne des informations sur quasiment tous les aspects de l'environnement Delphi, le langage Pascal Objet, les bibliothèques, etc.

Ce menu propose les commandes suivantes :

Commandes	**Description**
Aide Delphi	Ouvre la boîte de dialogue Rubriques d'aide. Dans ce dialogue, sélectionnez l'onglet Sommaire pour voir la liste hiérarchique des rubriques. L'onglet Index permet de voir les sujets classés dans l'ordre alphabétique ; l'onglet Rechercher permet de rechercher du texte dans tout le système d'aide. => sommaire, Index, Rechercher
Outils Delphi	Ouvre la boîte de dialogue Rubriques d'aide concernant les outils Delphi(Module bases de données, SQl, Winsight, …) => sommaire, Index, Rechercher
SDK Windows	Ouvre la boîte de dialogue WIN32 Developper's references
Page d'accueil Borland	Ouvre votre navigateur web et vous positionne sur le site web Inprise.fr.
Page d'accueil Delphi	Lien direct à la page d'accueil de Delphi sur le site web Inprise.fr.
Support développeurs Delphi	Lien direct Borland.com
Delphi Direct	Obtrenir les dernières rubriques de Delphi direct via Internet
Personnaliser	Configuration de l'OpenHelp
A propos…	Montre des informations sur le copyright et la version de Delphi.

1.2.3 La barre de commande

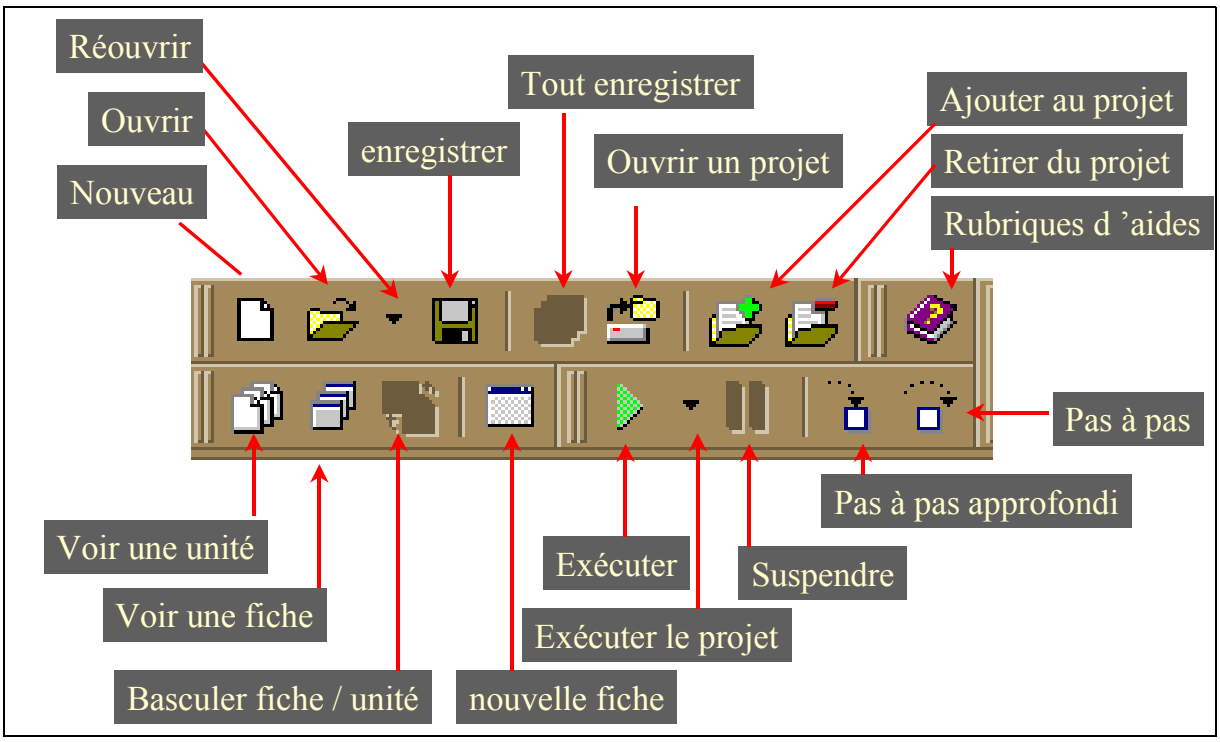
La barre d'outils de Delphi comporte des raccourcis pour les commandes de menu. Le graphique ci-dessous illustre la barre d'outils par défaut. Vous pouvez la personnaliser en choisissant Propriétés dans son menu contextuel.

Pour en savoir plus sur un bouton de la barre d'outils par défaut, cliquez sur un bouton dans le graphique ci-dessus. Vous pouvez utiliser la ligne qui sépare la barre d'outils de la palette des composants pour redimensionner horizontalement la barre d'outils.

La barre d'outils est munie de conseils que vous pouvez activer en sélectionnant Montrer conseils dans son menu contextuel.

La speed-bar est un ensemble de bouton avec icônes permettant d'exécuter directement, sans passer par les menus, les commandes les plus utilisées.

Les fonctionnalités accessibles sont, pour la configuration par défaut :



La barre de commande est configurable lorsque l'on fait appel à un menu contextuel, accessible via un click droit de la souris. La configuration se fait par "glisser-déposer".

1.2.4 La barre des composants

Cette barre, située à droite de la speed-bar et composée d'un ensemble d'onglets, permet de sélectionner les différents composants (objets prédéfinis) qui seront utilisés pour construire l'application.

Il suffit de sélectionner, à l'aide de la souris, le composant souhaité et de le déposer dans la fenêtre de conception pour que celui-ci soit intégré à l'application (le code nécessaire est généré automatiquement).

Chaque onglet contient différents composants regroupés par famille (mais il est possible de modifier cette disposition à l'aide du menu Outils/Options/Environnement /"palette").

Il est par ailleurs possible d'ajouter d'autres composants (créés ou acquis dans des bibliothèques spéciales).

Quel que soit l'onglet (ou la page) affiché, l'icône de gauche (icône curseur) permet de sélectionner le curseur.

Il suffit de cliquer sur l'onglet correspondant pour faire apparaître une page de composants spécifiques.

Page "Standard"

Cette page contient les composants utilisés les plus fréquemment pour construire une interface graphique (pour une application non orientée "base de données").

Composant	Description
TCadre	Conteneur pour d'autres composants
TMainMenu	Menu principal
TPopUpMenu	Menu déroulant ou menu pop-up
TLabel	Etiquette
TEdit	Boite de saisie ou d'affichage
TMemo	Zone "mémo" permettant l'afifchage de textes "multilignes"
TButton	Bouton
TCheckBox	Case à cocher
TRadioButton	Bouton radio
TListBox	Liste déroulante permettant une sélection parmi diverses options
TComboBox	Liste qui ne se déroule que sur l'action de l'utilisateur
TScrollBar	Ascenseur
TGroupBox	Boite regroupant des composants de même type

TRadioGroup	Boite regroupant des boutons radio
TPanel	Panneau
TAcionList	Listes d'actions utilisées avec des composants et des contrôles, tels que des éléments de menus et des boutons.

Page "Supplément"

Cette page contient des composants plus élaborés permettant de créer des applications au "look" plus actuel (barre d'icônes, onglet, images ....).

Composant	**Description**
TBitBtn	Bouton avec icône
TSpeedButton	Speed-button permettant la réalisation de barres d'icônes
TMaskEdit	Créateur de masque de saisie
TStringGrid	Grille de chaînes de caractères
TDrawGrid	Grille pouvant contenir des textes et des dessins
TImage	Insertion d'image
TShape	Objets graphiques géométriques
TBevel	Encadrements
TScrollBox	Boite avec ascenceurs
TcheckListBox	Liste défilante contenant une case à cocher à côté de chaque élément
Tsplitter	Divise la zone client d'une fiche en volets redimensionnables.
TstaticText	TStaticText est un contrôle fenêtré affichant du texte sur une fiche.
TControlBar	gère la disposition des composants de la barre d'outils.
TApplicationEvents	intercepte les événements au niveau de l'application.
Tchart	Composant standard TPanel pour la création de graphes.

Page "Win32"

Les composants de la page Win32 de la palette des composants permettent d'utiliser dans une application Delphi des contrôles d'interface utilisateur standard de Windows 32 bits (Windows 95 ou NT).

Composant	**Description**
TabControl	Contrôle onglets. Analogue aux intercalaires d'un classeur à onglets
PageControl	Contrôle pages. Ensemble de pages utilisé pour créer des boîtes de dialogue sur plusieurs pages.
ImageList	Liste d'images pouvant être référencées par leur indice.
RichEdit	Editeur de texte formaté. Contrôle mémo RTF.
TrackBar	Barre graduée.

ProgressBar	Barre de progression.
UpDown	Flèches Haut/Bas. Boutons flèches haut et bas permettant d'incrémenter et de décrémenter des valeurs.
HotKey	Touche d'accès rapide.
Animate	Animation. Un contrôle d'animation un clip AVI (Audio Video Interleaved)
DateTimePicker	Sélecteur date/heure.
TMonthCalendar	Calendrier indépendant dans lequel l'utilisateur peut sélectionner une date ou une plage de dates
TreeView	Vue arborescence.
ListView	Permet d'afficher une liste sous forme de colonnes.
HeaderControl	Contrôle en-tête. Permet d'afficher un en-tête au-dessus de colonnes de texte ou de nombres.
StatusBar	Barre d'état. Zone située au bas de l'écran et qui indique l'état des actions.
ToolBar	Barre d'outils. Permet de gérer les boutons d'outils et autres contrôles.
CoolBar	Barre multiple. Affiche une collection de contrôles fenêtrés.
TPageScroller	définit une zone d'affichage, comme une barre d'outils.

Page "Système"

Grâce aux composants de la page Système de la palette des composants, il est possible d'utiliser dans une application Delphi des contrôles spécialisés du système.

Composant	**Description**
Timer	Permet de déclencher des événements à intervalles réguliers. C'est un composant non visuel. Vous écrivez le code de ce qui doit se produire au moment spécifié, dans le gestionnaire de l'événement OnTimer du composant timer.
PaintBox	Boîte à peindre. Spécifie une zone rectangulaire de la fiche constituant la délimitation des dessins de l'application.
MediaPlayer	Multimédia. Affiche un contrôle de style télé-commande pour lire ou enregistrer des fichiers multimédias son ou vidéo.
OleContainer	Conteneur OLE. Crée dans une fiche une zone client OLE (Incorporation et Liaison d'Objet).
DdeClientConv	Conversation client DDE. Etablit une connexion client avec une application serveur DDE (Echange Dynamique de Données).
DdeClientItem	Elément client DDE. Spécifie les données du client DDE (Echange Dynamique de Données) à transférer lors d'une conversation DDE.
DdeServerConv	Conversation serveur DDE. Etablit une connexion serveur avec une application client DDE (Echange Dynamique de Données).
DdeServerItem	Elément serveur DDE. Spécifie les données du serveur DDE (Echange Dynamique de Données) à transférer lors d'une conversation DDE.

Page "Accès aux bases de données"

Cette page permet d'accéder, de manière complètement transparente, aux données contenues dans les tables (créées au préalable par l'utilitaire BDE ou par un autre SGBD) en utilisant des éléments spécialisés d'accès aux bases de données:

Composant	Description
DataSource	Source de données. Agit comme un conduit entre un composant TTable, TQuery ou TStoredProc et des composants orientés données, tels que TDBGrid.
Table	Table. Récupère les données d'une table physique, via le BDE, et les fournit à un ou plusieurs composants orientés données par le biais d'un composant DataSource. Inversement, envoie les données reçues d'un composant vers une base de données physique, via le BDE.
Query	Requête. Utilise des instructions SQL pour récupérer les données d'une table physique, via le BDE, et les fournir à un ou plusieurs composants orientés données par le biais d'un composant DataSource. Inversement, utilise des instructions SQL pour envoyer les données reçues d'un composant vers une base de données physique, via le BDE.
StoredProc	Procédure stockée. Permet à une application d'accéder à des procédures stockées sur serveur. Envoie les données reçues d'un composant vers une base de données physique via le BDE.
Database	Met en place une connexion persistante à une base de données, surtout une base de données distante nécessitant un nom de connexion utilisateur et un mot de passe.
Session	Session. Contrôle globalement un groupe de composants TDatabase associé. Un composant TSession par défaut est créé automatiquement pour chaque application base de données Delphi. Vous ne devez utiliser le composant TSession que si vous créez une application base de données multithread. Chaque thread de la base de données doit avoir son propre composant Session.
BatchMove	Action groupée. Copie une structure de table et ses données. Peut être utilisé pour transformer des tables entières d'un format de base de données à un autre format.
UpdateSQL	Mise à jour SQL. Permet d'utiliser le support de mise à jour en mémoire cache de Delphi avec des ensembles de données en lecture seule.
Provider	Fournisseur. Encapsule le mécanisme de fourniture de données depuis un serveur d'application multiliaison vers un ensemble de données client dans une application bureautique client.
ClientDataSet	Ensemble de données client. Composant ensemble de données pouvant être utilisé de façon autonome dans une application à liaison unique ou, en tant qu'ensemble de données client, dans la partie client d'une application de base de données multiliaison.
RemoteServer	Serveur distant. Etablit une connexion au serveur distant dans une application client faisant partie d'une application multiliaison.
MIDASConnection	Connexion MIDAS. Etablit une connexion DCOM, socket ou OLEnterprise à un serveur distant dans une application client faisant partie d'une application multiliaison.

Ces composants sont des composants "invisibles". Ils sont présents dans la fenêtre de conception mais n'apparaissent pas lors de l'exécution du programme.

Onglet "Contrôle des données"

Cette page permet l'affichage des données prises en compte par les composants de la page précédente. Ils correspondent, pour la plupart, aux composants de la page "Standard" mais sont spécialisés dans la construction d'interface orientées "base de données".

Composant	Description
DBGrid	Grille de base de données. Grille personnalisée orientée données permettant d'afficher les données sous forme d'un tableau comme dans une feuille de calcul.
DBNavigator	Navigateur de base de données. Boutons de navigation orientés permettant de déplacer en avant ou en arrière le pointeur d'enregistrement dans une table.
DBText	Texte de base de données. Libellé orienté données qui affiche la valeur d'un champ de l'enregistrement en cours.
DBEdit	Saisie de base de données. Boîte de saisie orientée données permettant d'afficher et de modifier un champ de l'enregistrement en cours.
DBMemo	Mémo de base de données. Mémo orienté données permettant d'afficher et de modifier du texte BLOB de l'enregistrement en cours.
DBImage	Image de base de données Image orientée données permettant d'afficher, de couper ou de coller des Blobs bitmap de et vers l'enregistrement en cours.
DBListBox	Boîte liste de base de données. Boîte liste orientée données.
DBComboBox	Boîte à options de base de données. Boîte à options orientée données qui permet d'afficher et de modifier une liste défilable des valeurs d'une colonne d'une table.
DBCheckBox	Case à cocher de base de données. Case à cocher orientée données qui permet d'afficher et de modifier un champ booléen de l'enregistrement en cours.
DBRadioGroup	Groupe de boutons radio de base de données. Groupe de boutons radio orientés données qui affichent l'ensemble des valeurs d'une colonne.
DBLookupListBox	Boîte liste de références de base de données. Boîte liste orientée données.
DBLookupComboBox	Boîte à options de références de base de données. Boîte à options orientée données.
DBRichEdit	Editeur de texte formaté de base de données. Contrôle de saisie multiligne qui permet d'afficher et de modifier un mémo formaté dans un ensemble de données.
DBCtrlGrid	Grille contrôle de base de données. Affiche des champs de plusieurs enregistrement.
DBChart	Graphe de base de données.

Page "InterBase"

Les composants de la page InterBase de la palette des composants vous permettent de vous connecter directement à une base de données InterBase sans utiliser de moteur tel que le BDE ou ADO (Active Data Objects).

Composant	Description
IBTable	Représente les données d'une seule table ou vue InterBase.
IBQuery	**Utilise les instructions SQL pour extraire des données d'une ou de plusieurs tables InterBase.**
IBStoredProc	Exécute la procédure stockée InterBase Execute.

IBDataBase	Représente la connexion à la base de données InterBase.
IBTransaction	Fournit le contrôle des transactions sur une ou plusieurs connexions à une base de données.
IBUpdateSQL	Vous permet d'utiliser le support du cache pour les mises à jour avec des requêtes en lecture seule.
IBDataSet	Répresente l'ensemble résultat d'une commande SQL SELECT.
IBEvents	Permet à une application de se recenser auprès des événements validés par un serveur InterBase et de les gérer de manière asynchrone.
IBSQL	Exécute une instruction SQL InterBase avec le minimum d'encombrement système.
IBDatabaseInfo	Renvoie des informations sur une base de données attachée, telles que la version de l'ODS (Online Disk Structure), le nombre de tampons de cache alloués, le nombre de pages lues et écrites dans la base de données, ou les informations historiques d'écriture.
IBSQLMonitor	Suit le SQl dynamique passé au serveur InterBase.

Page "Internet"

Les composants de la page Internet de la palette des composants offrent une variété de protocoles d'accès Internet pour vos applications Delphi. La version Client/Serveur offre des contrôles supplémentaires.

Composant	**Description**
ClientSocket	TClientSocket gère les connexions socket pour un client TCP/IP.
ServerSocket	TServerSocket gère les connexions de socket serveur d'un serveur TCP/IP.
WebDispatcher	TWebDispatcher transmet un message de requête HTTP aux éléments action appropriés qui assemblent une réponse.
PageProducer	TPageProducer génère une chaîne de commandes HTML à partir d'un modèle d'entrée
QueryTable Producer	TQueryTableProducer assemble une séquence de commandes HTML pour afficher les enregistrements du résultat d'une requête sous une forme tabulaire
DataSetTable Produce	TDataSetTableProducer assemble une suite de commandes HTML afin d'afficher de manière tabulaire un ensemble de données
DataSetPage Producer	TDataSetPageProducer génère une chaîne de commande HTML en se basant sur un modèle d'entrée
WebBrowser	TWebBrowser donne accès à la fonctionnalité de navigateur Web des objets Microsoft Shell Doc et Control Library

Page "Fastnet"

Les composants de la page NetMasters de la palette des composants offrent une variété de protocoles d'accès Internet pour vos applications

Composant	Description
NMDayTime	Obtient la date et l'heure d'un serveur de jour/heure internet/intranet.
NMEcho	Envoie du texte à un serveur d'écho internet, et vous renvoie l'écho.
NMFinger	Obtient des informations concernant un utilisateur à partir d'un serveur de finger internet, en utilisant le protocole Finger décrit dans RFC 1288.
NMFTP	Implémente le protocole de transfert de fichier.
NMHTTP	Ce contrôle ActiveX invisible implémente le protocole HTTP côté client à partir de la spécification HTTP. Ce contrôle vous permet de récupérer directement des documents HTTP si aucune visualisation ou traitement d'image n'est nécessaire.
NMMsg	Envoie de simples messages texte ASCII via internet ou un intranet, en utilisant le protocole TCP/IP.
NMMsgServ	Reçoit des messages envoyés avec le composant TNMMsg.
NMNNTP	Ce contrôle client ActiveX invisible permet aux applications d'accéder aux serveurs de nouvelles NNTP (Networking News Transfer Protocol).
NMPOP3	Ce contrôle ActiveX invisible récupère des messages électroniques depuis UNIX ou d'autres serveurs supportant le protocole POP3.
NMUUProcessor	Encode et décode des fichiers MIME ou UUEncodes.
NMSMTP	Ce contrôle ActiveX donne aux applications l'accès serveurs de messagerie SMTP et procure des fonctionnalités d'envoi de courrier.
NMStrm	Envoie des flux à un serveur de flux via internet ou un intranet.
NMStrmServ	Reçoit les flux envoyés par le composant TNMStrm.
NMTime	Obtient la date et l'heure à partir de serveurs d'heure Internet, comme décrit dans RFC 868.
NMUDP	Ce contrôle ActiveX Winsock invisible procure un accès facile au services réseau UDP (User Datagram Protocol).
PowerSock	Sert de base pour créer des contrôles supportant d'autres protocoles ou pour créer des protocoles personnalisés.
NMGeneralServer	Sert de classe de base pour développer des serveurs internet multithreads, tels des serveurs personnalisés ou des serveurs supportant les standards RFC.
HTML	Ce contrôle ActiveX invisible implémente un visualiseur HTML, avec ou sans récupération automatique des documents HTML sur le réseau et procure des fonctionnalités d'analyse et de mise en page des données HTML.
NMURL	Décode des données URL en une chaîne lisible et encode des chaînes normales en format de données URL.

Page "QReport"

Composant	Description
QuickRep	Etat. Le format de base sur lequel vous bâtissez tous vos états. C'est un composant visuel qui adopte la taille du format de papier sélectionné. Vous créez des états en plaçant des bandes et des composants imprimables sur le composant TQuickRep et en le connectant à un ensemble de données.
QRSubDetail	Détail. Etablit une liaison avec d'autres ensembles de données dans un état. Comme vous définissez une relation maître/détail entre des composants table ou requête, vous créez une relation similaire avec les composants TQRSubDetail.
QRStringsBand	Dépose les bandes contenant des chaînes dans un état.

QRBand	Bande. Place des bandes sur un composant TQuickRep et définit la propriété BandType de façon à indiquer comment se comportera la bande pendant la génération de l'état.
QRChildBand	Bande enfant. Si vous avez des bandes contenant des composants pouvant s'agrandir et si vous voulez que d'autres composants se déplacent avec eux, vous pouvez créer une bande enfant et y placer les composants à déplacer. Les bandes enfant sont également utiles si vous avez des bandes très longues occupant plusieurs pages.
QRGroup	Groupe.
QRLabel	Libellé. Imprime du texte statique ou tout autre texte non issu d'une base de données. Entrez le texte à afficher dans la propriété Caption. Vous pouvez découper le texte sur plusieurs lignes et même plusieurs pages.
QRDBText	Texte de base de données. Version orientée données du composant TQRLabel, pour imprimer les valeurs d'un champ d'une base de données. Les champs calculés et les champ texte peuvent être imprimés, y compris les champs de type chaîne, divers champs numériques, les champs date et les champs mémo. Le texte peut s'étendre sur plusieurs lignes ou plusieurs pages. Vous connectez le composant au champ de données en définissant les propriétés DataSource et DataField. Contrairement aux composants orientés données habituels, TQRDBText fonctionne même quand les contrôles ensemble de données sont désactivés pour améliorer la vitesse.
QRExpr	Expression. Imprime les champs, les calculs et le texte statique des bases de données. Entrez une expression QuickReport correcte dans la propriété Expression.
QRSysData	Données système. Imprime des informations système, telles le titre de l'état, le numéro de page, etc. Utilisez la propriété Data pour sélectionner les données à imprimer. Vous pouvez faire précéder ces données d'un texte quelconque, en indiquant celui-ci dans la propriété Text.
QRMemo	Mémo. Imprime une grande quantité de texte non issu d'un champ de base de données. Ce peut être un texte statique ou un texte que vous pouvez modifier pendant la génération de l'état. Vous pouvez autoriser le champ à s'étendre verticalement et à occuper plusieurs pages si nécessaire.
QRRichText	Texte formaté
QRDBRichText	Texte formaté de base de données
QRShape	Forme. Dessine dans un état des formes simples, telles des rectangles, cercles ou lignes.
QRImage	Image. Affiche une image dans un état. Accepte tous les formats d'images supportés par la classe TPicture de Delphi.
QRDBImage	Image de base de données. Imprime des images stockées dans des champs binaires (BLOB). Imprime tous les formats graphiques supportés par Delphi.
QRCompositeReport	Etat composite.
QRPreview	Prévisualisation.
QRTextFilter	Exporte le contenu d'un état au format texte.
QRCSVFilter	Exporte le contenu d'un état dans un fichier délimité virgule source de base de données.
QRHTMLFilter	Exporte le contenu d'un état au format HTML.
QRChart	Graphe.

Page "Dialogues"

Cette page contient des composants qui permettent d'accéder à des utilitaires systèmes intégrés à Windows. Ils évitent donc d'avoir à reprogrammer certaines fonctionnalités complexes.
Les composants de la page Dialogues de la palette des composants permettent d'utiliser dans une application Delphi les boîtes de dialogues communes de Windows. Grâce à ces boîtes de dialogue, il est possible de proposer une interface homogène pour des opérations relatives aux fichiers (comme l'enregistrement, l'ouverture ou l'impression).

Une boîte de dialogue commune est ouverte lors de l'exécution de sa méthode Execute. Execute renvoie l'une des valeurs booléennes suivantes :

- True, si l'utilisateur choisit OK et valide la boîte de dialogue
- False, si l'utilisateur choisit Annuler ou quitte la boîte de dialogue sans enregistrer aucune modification.

Chaque composant Boîte de dialogue commune (sauf le composant PrinterSetup) a un ensemble de propriétés regroupées sous l'intitulé Options dans l'inspecteur d'objets. Les propriétés Options interviennent sur l'aspect et le comportement des boîtes de dialogue communes. Pour afficher les propriétés Options, double-cliquez sur "Options" dans l'inspecteur d'objets.

Pour fermer par programmation une boîte de dialogue, utilisez la méthode CloseDialog.
Pour modifier la position d'une boîte de dialogue à l'exécution, utilisez les propriétés Handle, Left, Top et Position.

Composant	**Description**
OpenDialog	Boîte de dialogue d'ouverture. Affiche une boîte de dialogue d'ouverture commune de Windows. Les utilisateurs peuvent spécifier le nom du fichier à ouvrir dans cette boîte de dialogue.
SaveDialog	Boîte de dialogue d'enregistrement. Affiche une boîte de dialogue d'enregistrement commune de Windows. Les utilisateurs peuvent spécifier le nom du fichier à enregistrer dans cette boîte de dialogue.
OpenPictureDialog	Boîte de dialogue d'ouverture d'image. Affiche une boîte de dialogue modale de Windows pour sélectionner et ouvrir des fichiers graphiques. Semblable à la boîte de dialogue d'ouverture, avec en plus une zone de prévisualisation de l'image.
SavePictureDialog	Boîte de dialogue d'enregistrement d'image. Affiche une boîte de dialogue modale de Windows pour entrer des noms et enregistrer des fichiers graphiques. Semblable à la boîte de dialogue d'enregistrement, avec en plus une zone de prévisualisation de l'image.
FontDialog	Boîte de dialogue des fontes. Affiche une boîte de dialogue Police commune de Windows. Les utilisateurs peuvent spécifier la police, sa taille et son style dans cette boîte de dialogue.
ColorDialog	Boîte de dialogue des couleurs. Affiche une boîte de dialogue Couleur commune de Windows. Les utilisateurs peuvent spécifier des caractéristiques de couleur dans cette boîte de dialogue.
PrintDialog	Boîte de dialogue d'impression. Affiche une boîte de dialogue Imprimer commune de Windows. Les utilisateurs peuvent spécifier les caractéristiques d'impression (nombre de copies, intervalle de pages à imprimer) dans cette boîte de dialogue.
PrinterSetupDialog	Boîte de dialogue de configuration d'impression. Affiche une boîte de dialogue Configuration de l'impression commune de Windows. Les utilisateurs peuvent modifier ou configurer les imprimantes dans cette boîte de dialogue.
FindDialog	Boîte de dialogue de recherche. Affiche une boîte de dialogue Rechercher commune de Windows. Les utilisateurs peuvent spécifier une chaîne de caractères à rechercher dans cette boîte de dialogue.
ReplaceDialog	Boîte de dialogue de remplacement. Affiche une boîte de dialogue Remplacer commune de Windows. Les utilisateurs peuvent spécifier des chaînes de recherche et

	de remplacement dans cette boîte de dialogue.

Page "Win 3.1"

Les composants de la page Win31 de la palette des composants permettent d'utiliser dans vos applications Delphi des contrôles de Windows 3.1 pour assurer la compatibilité avec les applications construites avec des versions précédentes de Delphi. La plupart de ces anciens contrôles offrent le même comportement que les derniers contrôles 32 bits.

Composant	**Description**
TabSet	Onglets. Crée des onglets semblables à ceux d'un classeur. Le composant TabSet peut s'utiliser avec le composant Notebook pour permettre aux utilisateurs de changer de page.
Outline	Arborescence. Affiche des informations sous forme d'arborescences de différents formats.
Notebook	Classeur. Crée un composant pouvant contenir plusieurs pages. Le composant TabSet peut s'utiliser avec le composant Notebook pour permettre aux utilisateurs de changer de page.
TabbedNotebook	Classeur à onglets. Crée une zone d'affichage des données. Les utilisateurs peuvent redimensionner chaque section de cette zone pour y afficher différentes quantités de données.
Header	En-tête. Crée une zone d'affichage des données. Les utilisateurs peuvent redimensionner chaque section de cette zone pour y afficher différentes quantités de données.
FileListBox	Boîte liste de fichiers. Affiche une liste déroulante des fichiers du répertoire en cours.
DirectoryListBox	Boîte liste des répertoires. Affiche la structure des répertoires du lecteur actif. Les utilisateurs peuvent changer de répertoire dans une boîte liste des répertoires.
DriveComboBox	Boîte à options des lecteurs. Affiche une liste déroulante des lecteurs disponibles.
FilterComboBox	Boîte à options de filtrage. Spécifie un filtre ou un masque afin d'afficher un sous-ensemble des fichiers.
DBLookupList	Liste de référence de base de données. Boîte de liste orientée données pour afficher à l'exécution les valeurs trouvées dans les colonnes d'une autre table.
DBLookupCombo	Boîte à options de référence de base de données. Boîte à options orientée données pour afficher à l'exécution les valeurs trouvées dans les colonnes d'une autre table.

Page "Exemples"

Les composants de la page Exemples de la palette des composants sont des exemples de composants personnalisés que vous pouvez construire et ajouter à la palette des composants. Le code source de ces exemples est inclus dans le répertoire DELPHI\SOURCE\SAMPLES de l'installation par défaut.

Composant	Description
Gauge	Jauge
ColorGrid	Grille de couleurs
SpinButton	Incrémenteur
SpinEdit	Incrémenteur à saisie
DirectoryOutline	Arborescence de répertoires
Calendar	Calendrier
IBEVentAlerter	Alerteur d'événement

Page "ActiveX"

Les composants de la page ActiveX de la palette des composants sont des applications ActiveX, complètes et portables, créées par des développeurs tiers. Pour utiliser ces composants, vous devez d'abord ouvrir une fiche ActiveX avec un projet bibliothèque ActiveX. Après avoir placé un composant sur la fiche ActiveX, cliquez dessus avec le bouton droit pour afficher les propriétés ou d'autres commandes et boîtes de dialogues, pour définir les fonctionnalités et les valeurs de propriétés du composant. La boîte de dialogue Propriétés et d'autres contrôles contiennent des boutons Aide permettant d'accéder au système d'aide fourni par le développeur du composant.

Composant	Description
Chartfx	Permet de créer des graphes très personnels. Choisissez Propriétés pour afficher une boîte de dialogue à onglets qui permet de définir les valeurs, l'aspect et les fonctionnalités du composant Chart.
VSSpell	VisualSpeller, vérificateur d'orthographe personnalisable.
F1Book	Formula One, tableur comprenant un concepteur puissant.
VtChart	Permet de créer de vrais graphes en 3D.

Page "Servers"

Les composants de la page Serveurs de la palette des composants sont des enveloppes VCL pour les serveurs COM courants. Elles descendent toutes de TOleServer et ont été créées en important une bibliothèque de types et en installant les composants qui en résultent.
Word : WordApplication, WodDocument, WordFont, WordParagraphFormat, WordLetterContent,

Schedule : Binder,

Excel : ExcelQueryTable, ExcelApplication, ExcelChart, ExcelWorkSheet, ExcelWorkBook, ExcelOleObject,

Access : DoCmd, AccesHyperLink, AccesForm, AccesReport, AccesReferences,

PowerPoint : PowerPointApplication, PowerPointSlide, PowerPointPresentation,

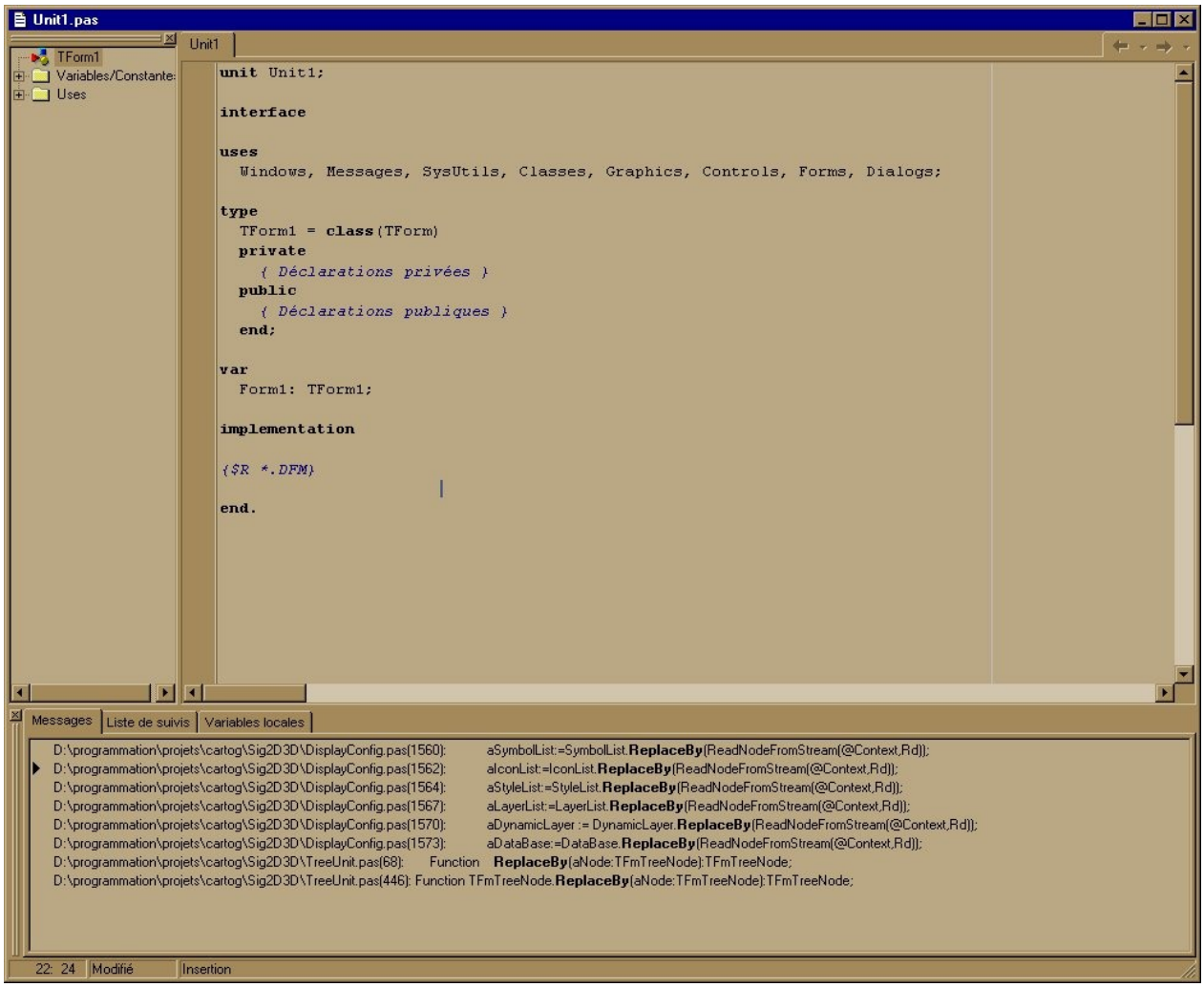
OutLook : OutlookApplication, AppointmentItem, ContactItem, JournalItem, MailItem, MeetingRequestItem, NoteItem, PostItem, RemoteItem, ReportItem, TaskItem, TaskRequestItem.

1.2.5 La forme

1.2.6 L'éditeur

L'éditeur peut comprendre :

- la fenêtre d'édition (centrale),
- des fenêtres arrimables ou pas :
  - la fenêtre explorateur comprenant :
    - les classes avec pour chaque classe, les déclarations privées, publiques et publiées,
    - les types définis,
    - les variables et constantes,
    - les unités déclarées dans le « uses »,
  - la fenêtre des messages,
  - la fenêtre « liste de suivis »,
  - la fenêtre « variables locales »,


Unit1.pas
TForm1
Variables/Constante
Uses
Unit1
unit Unit1;
interface
uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;
type
TForm1 = class(TForm)
private
{ Déclarations privées }
public
{ Déclarations publiques }
end;
var
Form1: TForm1;
implementation
{$R *.DFM}
end.
Messages
Liste de suivis
Variables locales
D:\programmation\projets\cartog\Sig2D3D\DisplayConfig.pas(1560): aSymbolList:=SymbolList.ReplaceBy(ReadNodeFromStream(@Context,Rd));
D:\programmation\projets\cartog\Sig2D3D\DisplayConfig.pas(1562): aIconList:=IconList.ReplaceBy(ReadNodeFromStream(@Context,Rd));
D:\programmation\projets\cartog\Sig2D3D\DisplayConfig.pas(1564): aStyleList:=StyleList.ReplaceBy(ReadNodeFromStream(@Context,Rd));
D:\programmation\projets\cartog\Sig2D3D\DisplayConfig.pas(1567): aLayerList:=LayerList.ReplaceBy(ReadNodeFromStream(@Context,Rd));
D:\programmation\projets\cartog\Sig2D3D\DisplayConfig.pas(1570): aDynamicLayer := DynamicLayer.ReplaceBy(ReadNodeFromStream(@Context,Rd));
D:\programmation\projets\cartog\Sig2D3D\DisplayConfig.pas(1573): aDataBase:=DataBase.ReplaceBy(ReadNodeFromStream(@Context,Rd));
D:\programmation\projets\cartog\Sig2D3D\TreeUnit.pas(68): Function ReplaceBy(aNode:TFmTreeNode):TFmTreeNode;
D:\programmation\projets\cartog\Sig2D3D\TreeUnit.pas(446): Function TFmTreeNode.ReplaceBy(aNode:TFmTreeNode):TFmTreeNode;
22: 24
Modifié
Insertion

1.2.7 L'inspecteur d'objets

L'inspecteur d'objets permet de définir les paramètres des composants à la conception. Deux onglets vous donnent accès aux propriétés et aux événements d'un composant.

2 L'environnement Windows

2.1 Description générale

Il n'est pas possible de développer une "application Windows" complexe sans connaître les caractéristiques de cet environnement ainsi que son mode de fonctionnement interne.

L'environnement graphique Windows est un produit, mis au point par la société Microsoft, destiné à mettre à la disposition des utilisateurs une interface graphique conviviale (une GUI : Graphic User Interface) permettant de travailler dans de meilleures conditions.

2.1.1 Histoire de Windows

L'origine de Windows est celle de tous les autres GUI utilisés dans le monde informatique :

- héritière des travaux de la société Xerox, dans les années 70, au laboratoire PARC (Palo Alto Research Center),
- Steve Jobs, un des fondateurs d'Apple avec la création de Mac Intosh en 1984,
- Microsoft ne commença à s'intéresser à ce type d'interface qu'en 1983, soit deux ans après l'arrivée du premier PC sur le marché (en 1982 le système d'exploitation DOS n'en était qu'à sa version 2.0…).
    - La version 1.01 de Windows, parue en novembre 1985, tenait sur 2 disquettes et se contentait de 256 Ko de RAM.
    - La version 2.0 suivie en 1987 : elle permettait la gestion des fenêtres se recouvrant et gérait la mémoire EMS.
- En 1987 d'autres environnements graphiques étaient proposés au public :
    - L'environnement du Mac Intosh,
    - Personnal Manager sur OS / 2,
    - Une interface utilisée sur DOS : l'interface GEM proposée par Digital Research.
- La version 3.0 de Windows fut livrée en mai 1990, première à rencontrer le succès malgré de nombreuses insuffisances,
- Version 3.1 apparue en 1992, l'interface pratiquement obligée des applications tournant sous DOS.
- Windows 95, sorti en 1995, utilise une nouvelle interface reprenant certains principes de fonctionnement de Windows.
- Windows NT, un véritable système d'exploitation évitant (bien que cela ne soit pas tout à fait vrai) d'avoir à recourir aux mécanismes internes du DOS.

2.1.2 Fonctionnement en mode événementiel

Windows, comme tous les autres interfaces graphiques, fonctionne selon un mode événementiel.

Le fonctionnement interne d'une application écrite selon le mode événementiel est différent de celui d'une application classique :

- Dans le cas d'une application classique, ce sont les différentes instructions du code, (et donc le programmeur), qui indiquent quand il faut interroger le système pour obtenir différentes informations ou différentes données.

Par exemple, c'est une routine du code qui interroge le système pour savoir si un caractère est disponible dans le buffer clavier, si l'utilisateur a cliqué avec sa souris, etc.... Le programme peut être bloqué tant que l'information souhaitée n'est pas disponible .

- Dans le cas d'une application événementielle, c'est le système qui alerte l'application qu'une information le concernant (un "événement") est disponible. Le programme tient alors compte de cet événement pour modifier son comportement en exécutant une fonction adaptée (fonction handler).

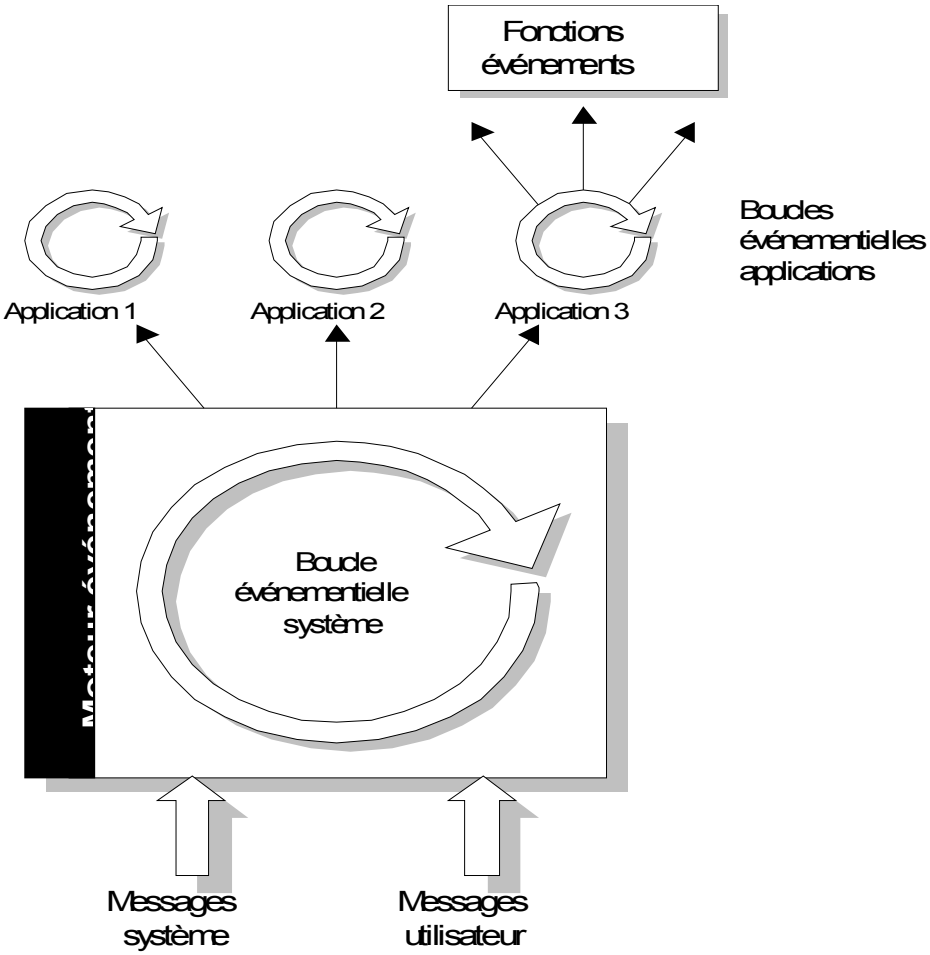
Ce qui revient à dire que le système alerte « qu'un caractère est dans le buffer clavier" , ou "il y a eu un clic de la souris à tel endroit de l'écran". Au programme de prendre en compte ou non ce message.

Dans le cas d'un mode normal de fonctionnement, le programmeur est maître de la structure du code qu'il génère et de la manière dont les différents tests, et les actions qui en découlent, s'enchaînent. Dans le cas d'une application événementielle, le programmeur doit prévoir tous les cas possibles de réactions aux différents événements mais il n'est absolument pas maître de l'ordre dans lequel ces événements apparaîtront.

On dit parfois que Windows est un environnement "piloté par événements".

- Bien prendre en compte qu'au niveau interne des milliers d'événements peuvent être générés dans des délais très brefs (pensez au déplacement d'une souris sur l'écran).

- Lorsqu'il est lancé, Windows met en place un moteur événementiel capable de récupérer les événements reçus par le système et de les placer dans une ou des files d'attente pour qu'ils soient pris en compte par les applications



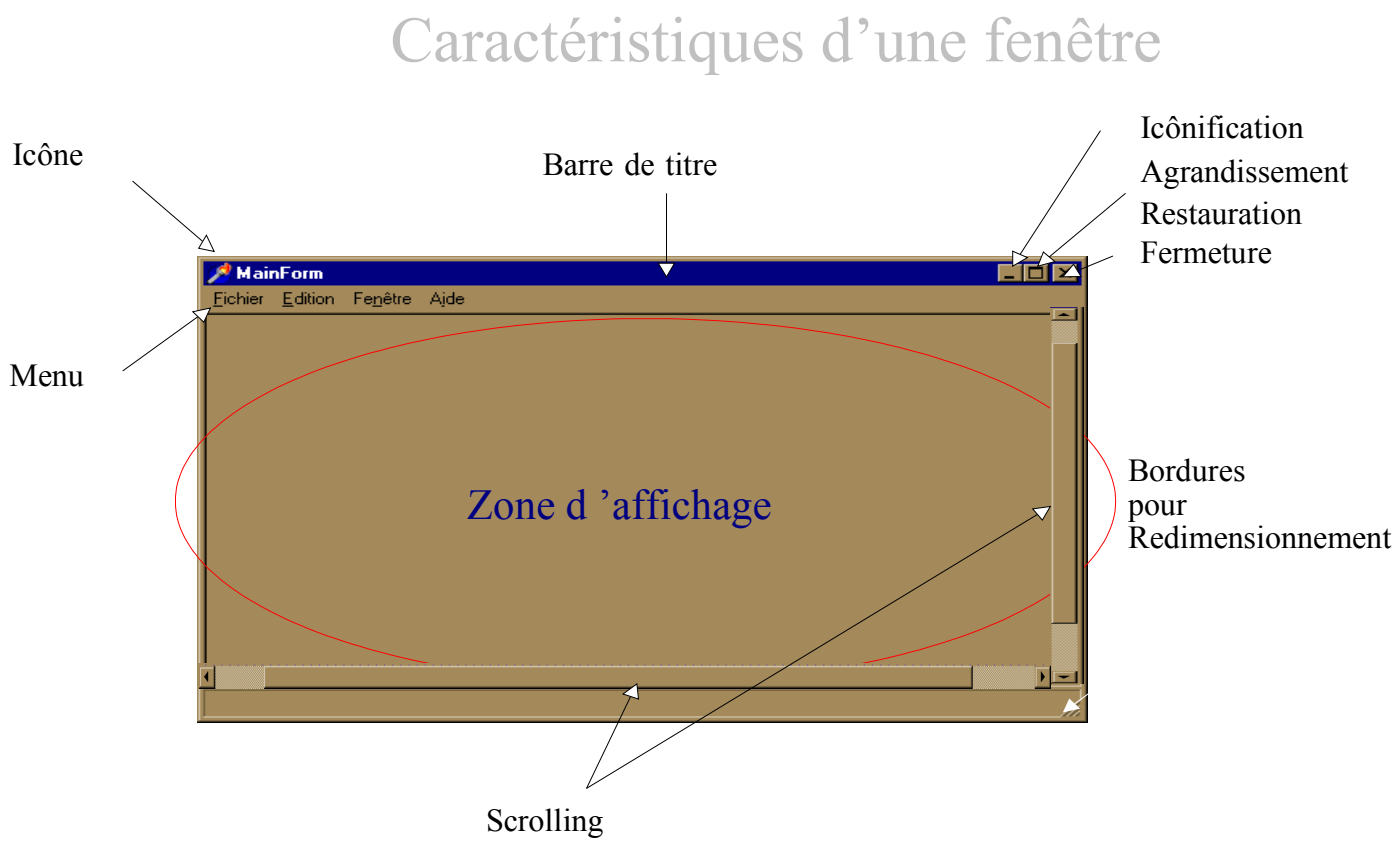
2.1.3 Caractéristiques générales d'une fenêtre

▲ **Eléments fondamentaux d'une fenêtre**

Il existe plusieurs types de fenêtres qui présentent néanmoins de nombreuses caractéristiques semblables (au niveau de leur gestion interne):

- Les fenêtres proprement dites, dans lesquelles s'exécutent les applications, sont généralement dotées d'une barre de titre, de menus et de "boutons systèmes" situés de part et d'autres de la barre de titre. Elles sont redimensionnables.

Une fenêtre se présente souvent sous la forme suivante :



Le bouton de fermeture peut être invoqué de deux manières :

- Si l'on clique simplement sur lui il y a fermeture de la fenêtre.
- Si l'on "double-clique" sur le bouton Système.

Le bouton "d'icônification" permet de réduire la fenêtre à la taille d'une icône. L'application contenue est toujours active. Cette option permet de dégager l'espace de travail.

Lorsqu'une application est icônifiée, son nom apparaît sous l'icône.

Le bouton d'agrandissement permet d'agrandir la fenêtre à la taille de l'écran. Lorsque cela est réalisé il change de forme pour indiquer qu'il est possible de réduire la fenêtre à sa taille initiale.

La barre de titre contient le nom de l'application. Elle permet aussi le déplacement de la fenêtre sur l'espace de travail (l'utilisateur clique sur la barre de titre avec la souris et maintient le bouton pressé jusqu'à ce qu'il le relâche au nouvel emplacement).

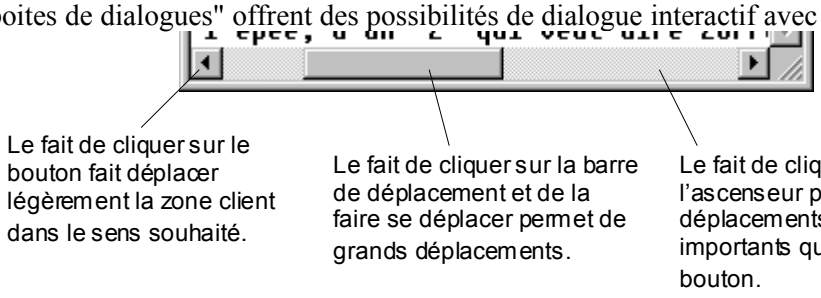
Les bordures permettent de redimensionner finement la taille de la fenêtre.

▲ Autres éléments d'une fenêtre

Dans certains cas, à la suite d'un redimensionnement de la fenêtre, la surface interne de celle-ci (appelée zone "client ") n'est plus affichée intégralement.

Windows met alors en place des ascenseurs (un vertical et un horizontal) permettant d'accéder aux zones de la fenêtre non affichées. On parle alors de scrolling.

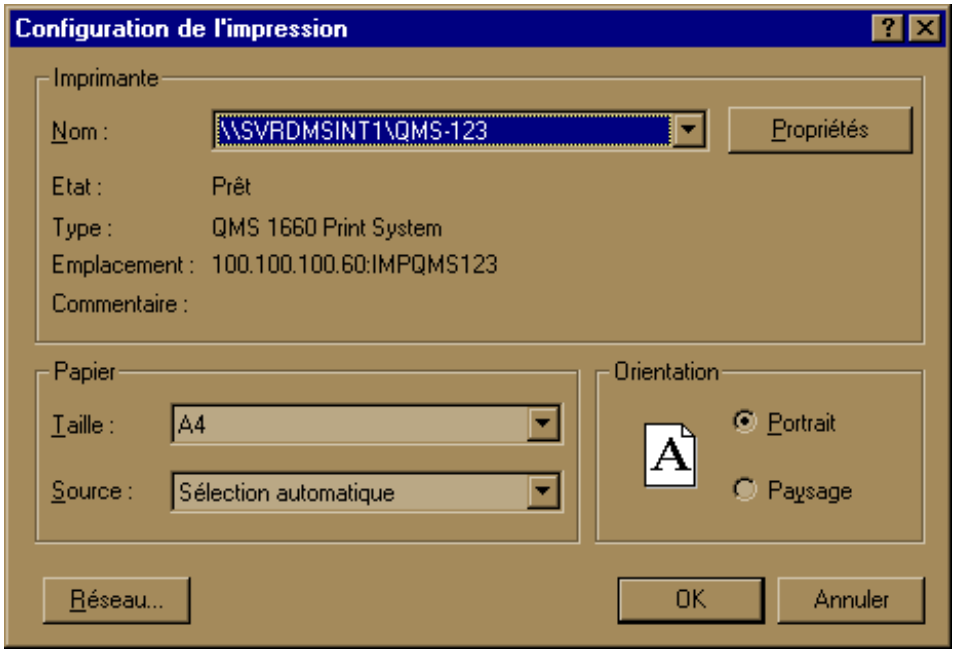
- Les "boites de dialogues" offrent des possibilités de dialogue interactif avec l'utilisateur. Elles sont



  généralement de petites tailles, ne disposent pas de menu, et ne sont pas redimensionnables. Elles sont dépourvues de boutons de redimensionnement et de bordures.

- Les "messages d'avertissement" sont des fenêtres rudimentaires, parfois dépourvues de barre de titre, et ne possédant ni "boutons systèmes" ni bordures et n'étant pas redimensionnables.

  - En interne tout objet faisant partie d'une interface est considéré comme une fenêtre et est géré comme tel (bouton, menu, ascenseur, etc…).



Une boite de dialogue ne peut généralement pas être redimensionnée, ne comporte que le bouton système de fermeture et ne peut proposer de menu.

Une boite de message ne permet que certaines actions. Elle est parfois dépourvue de barre de titre et réduite à un message et un seul bouton.

▲ L'environnement « fenêtre »

On considère parfois Windows principalement comme un "gestionnaire de fenêtres". La fenêtre ("Window") est en effet le constituant principal de cet environnement :

- Une application est lancée dans une fenêtre qui lui est propre.
- Pour terminer l'application, il faut fermer la fenêtre (et non simplement la réduire ou la cacher).

- Les événements systèmes sont gérés au niveau de la fenêtre.

Une fenêtre peut être assimilée à un terminal virtuel dans lequel une application s'exécute. Windows peut gérer simultanément plusieurs fenêtres mais une seule est active à un moment donné.

2.1.4 Fenêtre modale ou amodale

Les fenêtres, de quelque type qu'elles soient, fonctionnent selon deux modes différents : le mode modal ou le mode amodal.

En phase de conception, les boîtes de dialogue sont simplement des fiches personnalisées. A l'exécution, ce sont des boîtes de dialogue modales ou non modales. Lorsqu'une fiche est exécutée de façon modale, l'utilisateur doit obligatoirement la fermer pour pouvoir travailler dans une autre fiche. La plupart des boîtes de dialogue sont des boîtes de dialogue modales.
Les fiches exécutées d'une façon non modale peuvent rester à l'écran lorsqu'un utilisateur travaille dans une autre fiche comme la fiche principale de l'application. Les fiches non modales sont souvent utilisées pour afficher des informations d'état comme le nombre d'enregistrements trouvés dans une requête ou des informations auxquelles l'utilisateur souhaitera se référer lorsqu'il travaille.

Remarque :
En mode exécution, une boîte de dialogue non modale peut rester affichée devant les autres fenêtres si sa propriété FormStyle a été définie en mode conception à fsStayOnTop.

La modalité des fiches est déterminée par deux méthodes. Pour qu'une fiche fonctionne de façon non modale, vous devez appeler sa méthode **Show**, et pour qu'elle fonctionne de façon modale, vous devez appeler sa méthode **ShowModal**.

▲ **Mode modal :**
Dans ce mode, la fenêtre possède sa propre boucle événementielle. Lorsqu'elle est active seuls les événements qui lui sont associés peuvent être pris en compte.

Ce qui revient à dire que toute action exécutée en dehors de la superficie occupée par la fenêtre ne sera pas prise en compte : pour pouvoir accéder aux menus, par exemple, il faut au préalable fermer la fenêtre modale.

▲ **Mode non modal :**
Les fenêtres non modales (ou amodales) se partagent la même boucle événementielle. De ce fait, des événements destinés à une autre fenêtre amodale peuvent être pris en compte.

De ce fait on peut basculer d'une fenêtre à l'autre sans avoir à fermer les fenêtres.

Les règles générales sont adoptées :

- Les fenêtres des applications fonctionnent en mode amodal (ce qui permet de faire passer une fenêtre au premier plan en la rendant active). Mais ceci n'est pas une obligation.
- Les boites de dialogue fonctionnent la plupart du temps en mode modal.
- Les boîtes de messages fonctionnent impérativement en mode modal.

2.1.5 Applications MDI et SDI

On distingue les applications fonctionnant en mode normal, en mode MDI de celles fonctionnant en mode SDI :

▲ **Applications "normales" « Simple Document Interface »:**
Il s'agit d'applications qui ne sont composées que d'une fenêtre principale, constituant le cadre de l'application, et, éventuellement de boîtes de dialogue et de fenêtres de messages.

Le bloc notes de Windows fonctionne selon ce mode : lorsqu'un document est chargé il n'est pas possible d'en charger un autre sans au préalable fermer le premier document.

▲ **Applications MDI « Multiple Document Interface »:**
Ce type d'application gère plusieurs fenêtres filles (ou fenêtres enfants) qui sont des répliques d'un modèle unique : il est alors possible, en particulier, de charger plusieurs documents (chacun dans une fenêtre fille particulière).

Eventuellement les menus attachés à chaque fenêtre fille peuvent être distincts de celui de la fenêtre principale..

Il faut noter que l'ensemble des fenêtres filles ne peut pas dépasser l'espace alloué à la fenêtre principale de l'application : il y a clipping de la fenêtre fille lorsque, à la suite d'un déplacement, une partie de sa surface se retrouve à l'extérieur de la surface de la fenêtre mère (néanmoins Windows continue à gérer la partie de la fenêtre ainsi effacée).

Exemple d'application MDI : WinWord

▲ **Applications SDI :**
Ce type d'application gère plusieurs fenêtres indépendantes les unes des autres.
Dans certains cas, la fenêtre principale est réduite à la barre des menus et / ou à celle des icônes. De ce fait les fenêtres filles donnent l'impression de "flotter" au dessus d'une application étrangère sous-jacente : Delphi fonctionne selon ce mode et est grand générateur de fenêtres. Pour s'y retrouver il est conseillé de fermer ou d'icônifier toutes les autres applications actives afin que les fenêtres "flottent" sur le fond d'écran du gestionnaire de programme voire sur celui du bureau de Windows.

2.2 Modes de programmation d'une application Windows

Programmer "Windows" est un travail complexe qui nécessite beaucoup d'attention de la part des programmeurs. Pour programmer une application fonctionnant dans l'environnement Windows, il faut que celle-ci soit en mesure d'appeler les différentes fonctions faisant partie de l'API Windows. Il faut donc savoir :

- comment appeler les différentes fonctions,
- comment gérer les différents composants d'une application.

Plusieurs méthodes de programmation sont disponibles pour réaliser une application Windows :

▲ **Utilisation du SDK (Software Development Kit)**

C'est l'ensemble des objets et des fonctions (pratiquement un millier) fournis par Microsoft pour créer directement des applications Windows.

Le langage utilisé pour manipuler le SDK est le langage C traditionnel : les spécificités du C++ et de la P.O.O. n'entrent pas en ligne de compte.

Pour réaliser une application Windows avec le SDK, il faut connaître l'ensemble des fonctions et des variables prédéfinies qui le composent et maîtriser intégralement le développement de l'application (gestion des événements, création d'une fenêtre, etc…).

▲ Utilisation de bibliothèques de classes

Face à la difficulté réelle d'utiliser le SDK, les principaux éditeurs de compilateurs fournissent des bibliothèques de classes qui comportent un grand nombre d'objets permettant de réaliser plus rapidement des applications Windows importantes.

Les principales bibliothèques de classes disponibles sont écrites en langage C++ (bibliothèque OWL de Borland ou bibliothèque MFC de Microsoft). La bibliothèque VCL de Delphi est basée sur le langage Object Pascal.

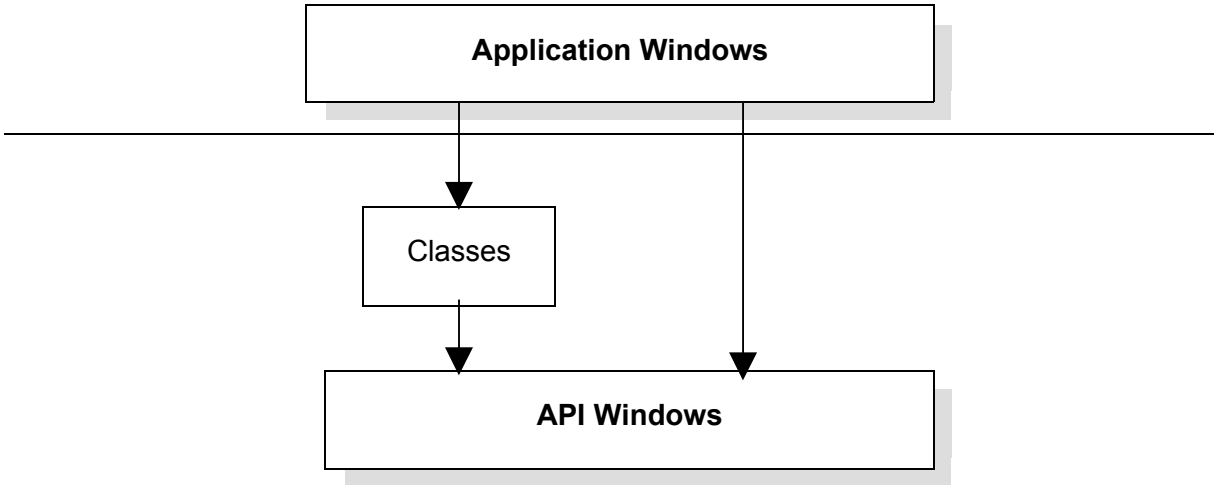
Ces bibliothèques prennent en charge plus ou moins complètement les principales difficultés de la programmation Windows (gestion des événements, gestion des interfaces graphiques, etc ...) en "encapsulant" les appels aux fonctions du SDK dans des classes spécifiques.

Elles fournissent des classes contenant des fonctions membres dont l'exécution met en œuvre des mécanismes complexes qui auraient été délicats à maîtriser avec le SDK.

Elles proposent des classes qui permettent de construire des objets qu'il n'est pas possible de créer simplement avec les outils fournis par le SDK.

Le programmeur aura alors le choix entre :

- Utiliser les possibilités offertes par la bibliothèque utilisée. Ce choix est préférable lorsqu'il s'agit d'utiliser des objets de haut niveau (fenêtres) ou des mécanismes complexes fortement encapsulés (gestion des messages et diffusion de ceux-ci).
- Utiliser directement des fonctions de l'API. Ce choix peut être judicieux lorsque l'encapsulation est faible (par exemple lorsque la fonction de la bibliothèque de classes se contente d'appeler la fonction correspondante de l'API). Dans ce cas, les performances de l'application sont améliorées car en appelant directement la fonction du SDK, les appels inutiles de fonctions sont éliminés.



▲ Les différentes API

Pour accéder à Windows il est possible d'utiliser plusieurs API :

L'API Win16 est principalement utilisée pour créer des applications standards tournant sur Windows jusqu'à la version 3.11. Cette API utilise strictement les possibilités "16 bits" des processeurs (et ne prend donc pas en compte les améliorations apportées par les processeurs 32 bits). La grande majorité des applications développées utilisent encore cette API.

L'API Win32 utilise pleinement les améliorations apportées par les nouveaux processeurs Intel 32 bits (les différentes données sont codées sur 32 bits). En particulier, toutes les fonctions d'allocation mémoire bénéficient du passage en mode 32 bits.

L'API Win32 permet de créer des applications tournant dans les environnements Windows NT et Windows 95.

A quelques fonctions près, les deux API sont compatibles. Pour assurer un meilleur portage des applications développées pour Win16 dans un environnement 32 bits, il suffit d'éviter l'utilisation des quelques fonctions n'ayant pas été intégrées à l'API Win32.
Il existe aussi une API appelée Win32s qui est un sous ensemble de l'API permettant de faire fonctionner des applications "32 bits" dans l'environnement Windows 3.1.

▲ Les choix de programmation

Par delà les distinctions évoquées précédemment, il est important de noter que les applications réalisées peuvent être classées en deux catégories très distinctes (certaines applications font partie des deux catégories) dont il est important de connaître les différences fondamentales :

- Les applications utilisant massivement des boites de dialogue et des objets prédéfinis. Ces applications sont relativement aisées à réaliser surtout si l'on utilise des bibliothèques d'objets prédéfinis. Tout le dialogue entre l'utilisateur et l'application est réalisé à travers des actions sur des boutons, des listes ou des zones de saisie.

- Les applications telles les gestionnaires de fichiers, les tableurs, et plusieurs utilitaires font partie de cette catégorie.

- Les applications créant des objets à l'écran et modifiant leur interface graphique : Ces applications doivent gérer directement l'affichage graphique via des fonctions appelant GDI.

L'utilisateur peut accéder à des outils spécifiques lui permettant de créer et modifier des objets graphiques pendant l'exécution de l'application.

Les différents logiciels graphiques et les modules graphiques des tableurs font partie de cette catégorie. La partie "gestion du texte" des traitements de texte peut être assimilée à cette catégorie.

Il faut comprendre que cette gestion "bas niveau" de l'interface graphique est plus complexe qu'il n'y paraît. GDI étant le composant système le plus complexe, les fonctions d'accès au GDI ne sont pas complètement encapsulées dans les bibliothèques de classes proposées. Dès qu'un programmeur souhaite personnaliser finement l'apparence graphique de son application il doit se résoudre, souvent, à utiliser directement les fonctions du SDK, avec toute la complexité de manipulation que cela entraîne.

2.3 Normes de programmation

L'un des principaux attraits de l'interface Windows est que la plupart des applications mettent en œuvre les mêmes mécanismes de fonctionnement :

Les menus sont souvent classés dans le même ordre afin que l'utilisateur n'ait pas à chercher longtemps la fonctionnalité désirée.

Les boites de dialogues se manipulent de la même manière et présentent les mêmes séquences de boutons de contrôle.

- Même si au début, certaines applications ont proposé des interfaces "divergentes", aujourd'hui une application, qui proposerait des menus et des boites de dialogues trop différents, serait condamnée commercialement.

  Imaginez une boite de dialogue ou le bouton "Fermer" soit placé à la place habituelle du bouton "OK"…

Quand on réalise une application Windows, il y a donc lieu de connaître les principales normes retenues.

2.3.1 Ordre des menus

Toutes les applications Windows doivent positionner les menus selon un ordre précis (sauf cas d'applications vraiment particulières). Seuls, certains menus doivent être positionnés selon les normes, les autres peuvent être spécifiques à l'application :

▲ **Le menu Fichier :**
Ce doit être le premier menu à partir de la gauche. Il comporte les rubriques permettant de manipuler les fichiers utilisés par l'application. Les rubriques imposées sont :

Nouveau :	création d'un fichier
Ouvrir :	ouverture d'un fichier existant.
Fermer :	fermeture d'un fichier sans sortir de l'application.
Enregistrer :	sauvegarde des données sur le disque.
Enregistrer sous :	sauvegarde sous un autre nom.
Imprimer :	impression du document et configuration de l'imprimante.
Quitter :	quitter l'application.

Dans ce menu, on peut insérer d'autres rubriques (selon l'application) et éventuellement, en bas du menu, l'historique des derniers fichiers ouverts.

▲ **Le menu Edition :**
Ce menu, le deuxième sur la barre de menu, donne accès aux fonctions de gestion du presse-papier (Copier | Couper | Coller) ainsi que, le cas échéant, des fonctionnalités de recherche et de remplacement de texte.

Il propose souvent une fonction permettant d'annuler la dernière action. Si cette fonctionnalité est proposée, elle doit se situer en tête de la liste des éléments du menu.

▲ **Le menu Fenêtre :**
Ce menu, systématiquement l'avant dernier dans la barre des menus, permet de gérer les différentes fenêtres ouvertes (dans le cas d'une application MDI).

Il permet de positionner les fenêtres en cascade ou en mosaïque.

▲ **Le menu "?" :**
Ce menu, s'il existe, est obligatoirement le dernier. Dans certains cas, il est déplacé à l'extrême droite de la fenêtre.

Il permet d'accéder à l'aide en ligne de l'application (obligatoirement réalisée avec en utilisant le format spécifique de texte enrichi RTF et en utilisant un compilateur d'aide permettant de générer un fichier '.HLP').

Les autres menus, placés systématiquement entre les quatre menus cités précédemment, sont spécifiques à l'application.

Les rubriques les plus utilisées des différents menus sont de plus en plus doublées par des boutons à icônes placés dans une barre située normalement juste au dessous de la barre de menus. Là aussi, il y a lieu d'utiliser des icônes standardisées facilement interprétables.

Ces mêmes rubriques sont souvent doublées de combinaisons de touches permettant les "raccourcis clavier". Les combinaisons les plus connues sont aussi standardisées :

Fichier | Ouvrir :	Ctrl + 0
Edition | Copier :	Ctrl + C
Etc…	

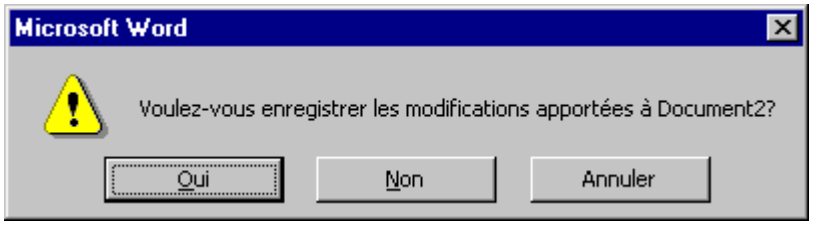
2.3.2 Les boutons d'une boîte de dialogues

Les boîtes de dialogues proposent généralement un jeu plus ou moins complet de boutons permettant de réagir aux souhaits de l'utilisateur.

Les boutons utilisés sont légendés : les légendes utilisées répondent à une standardisation de fait :

- 'OK' ou 'Oui' : L'utilisateur valide les choix réalisés dans la boîte de dialogue. Sauf erreur détectée par l'application, l'effet et immédiat et la boîte est refermée.
- 'Non' : l'utilisateur ne valide pas la proposition de la boîte de dialogue. La boite est fermée et l'application tient compte du choix de l'utilisateur .
- 'Annuler' ou 'Cancel' : Les choix réalisés ne sont pas pris en compte. La boîte de dialogue est fermée et l'application revient à la situation antérieure sans aucune modification.

Dans le cas où les trois boutons apparaissent, ils doivent respecter l'ordre de la figure ci-après :



2.3.3 Contraintes systèmes

▲ **Niveaux de fenêtres :**
Afin de ne pas gaspiller inutilement les ressources systèmes, et aussi dans le but de simplifier l'utilisation des applications, Windows impose qu'il n'y ait pas plus de deux niveaux de fenêtres filles dans une application (à l'exclusion des boîtes de dialogue et des messages).

▲ **Utilisation de composants complexes :**
Il faut aussi faire attention aux objets que l'on utilise pour construire une interface graphique moderne. Les tendances actuelles (multi-fenêtrages, icônes, multiplication des boutons, etc …) font que les applications actuelles sont plus gourmandes en ressources qu'auparavant.

Par exemple une mode actuelle est d'utiliser un objet de type "onglet" (Word 6 en fait une grande consommation). Si les objets déposés dans les différents feuillets de l'onglet ne sont pas trop complexes, cela peut aller. Mais si toute l'application est gérée au sein d'une seule fenêtre contenant un onglet, chaque feuillet comprenant plusieurs objets (dont des objets d'accès aux bases de données, les plus gourmands), la consommation en ressources systèmes peut devenir très importante. En effet l'onglet est considéré comme un seul objet et est chargé d'un seul bloc en mémoire avec tous les composants qu'il contient.

▲ **Chargement en mémoire des différentes fenêtres :**
Il est possible de charger en mémoire toutes les fenêtres constituant une application lors du démarrage de l'application. Cette option peut être gourmande en ressources. Certaines fenêtres peuvent ne jamais être utilisées pendant une session de travail.
Il est donc généralement préférable de ne charger les fenêtres que lorsque le besoin s'en fait sentir, après une action particulière de l'utilisateur.

▲ **Découpage du programme en modules dynamiques :**
Les programmes graphiques ont souvent une taille très importante. Il est souhaitable de découper un programme un peu ambitieux en plusieurs modules chargeables dynamiquement (DLL).

▲ **Etre efficace :**
Il ne suffit pas de créer une interface graphique utilisant les dernières nouveautés rendues possibles par les bibliothèques de classes ni même de respecter toutes les normes et standard pour qu'une application ait du succès. Encore faut-il que l'application soit réellement utilisable. Pour ce faire, il faut que chaque fenêtre et chaque boite de dialogue ne contiennent que les composants utiles à un moment donné, et surtout qu'il n'y en ait pas trop. Une fenêtre contenant trop de boutons, trop de zones de saisie, trop d'options devient inutilisable.

La création d'une interface peut être très longue. Surtout que l'esprit est habitué à ce qui est ordonné : il faut donc aligner les boutons et les différents cadres, et cela finit par prendre du temps.

3 Eléments de syntaxe Pascal

Le langage implémenté par Delphi est dérivé directement du langage de programmation Pascal. Il s'en distingue par sa capacité à accéder aux fonctions de l'API Windows et implémente les mécanismes permettant de réaliser des programmes respectant les principes de la programmation orientée objet (P.O.O.). Pour cela quelques évolutions ont été apportées au langage de base et à l'Object Pascal.

De fait, pour pouvoir programmer avec Delphi il faut, dans un premier temps assimiler la syntaxe générale du langage Pascal. Ce langage est connu de la plupart des programmeurs pour la clarté et la précision des règles syntaxiques et lexicales qu'il met en œuvre.

- On se rendra compte assez rapidement que beaucoup d'éléments de la grammaire du Pascal ne sont pas, ou très peu, utilisés dans un programme Delphi. De plus certaines fonctions d'entrée / sortie deviennent inopérantes (du fait de l'environnement graphique). C'est donc un ensemble restreint du Pascal qu'il faut assimiler, mais il faut y rajouter les extensions utilisées dans le cadre de la programmation événementielle et de la P.O.O.

Les chapitres qui suivent présentent donc brièvement certaines caractéristiques du langage Pascal. Plusieurs spécificités ont été volontairement omises. Un programmeur qui en aurait besoin, pour une application particulière, trouvera toujours le renseignement souhaité dans l'imposante aide en ligne livrée avec Delphi.

3.1 Identifiant, opérateurs et types

3.1.1 Identifiants

Les identifiants du langage Pascal, permettant de déclarer des variables, des fonctions ou des procédures présentent les caractéristiques suivantes :

- la taille est quelconque mais seuls les 255 premiers caractères seront pris en compte.
- ils ne peuvent être constitués que de lettres (A - Z , a - Z), de chiffres (0 - 9) et du caractère ' _ ' (à l'exclusion de tout caractère accentué).
- ils doivent commencer obligatoirement par une lettre ;

Contrairement à d'autres langages Pascal ne distingue pas les majuscules des minuscules (la variable "Toto" est la même que la variable "TOTO" ou la variable "toto").

- Cependant l'habitude est prise, en suivant en cela l'exemple des fonctions de l'API Windows, de déclarer des variables mélangeant majuscules et minuscules (quitte à ce que cela soit un peu plus compliqué à taper):

  CompteurDeTour, ValeurMaximale, etc…

Mots réservés :

Un certain nombre de mots ne doivent pas être utilisés par les programmeurs pour déclarer leurs propres variables car ils sont utilisés par le Pascal, ce sont les mots réservés :

and	exports	mod	shr
array	file	nil	string
as	finalization	not	stringresource
asm	finally	object	then
begin	for	of	threadvar
case	function	or	to
class	goto	out	try
const	if	packed	type
constructor	implementation	procedure	unit
destructor	in	program	until
dispinterface	inherited	property	uses
div	initialization	raise	var
do	inline	record	while
downto	interface	repeat	with
else	is	resourcestring	xor
end	label	set	
except	library	shl	

Les identificateurs suivants constituent les directives standard intégrées de Pascal Objet. Les directives ne sont utilisées que dans des contextes où il n'est pas possible de rencontrer un identificateur défini par l'utilisateur. A la différence des mots réservés, les directives peuvent être redéfinies par l'utilisateur, bien que cela ne soit pas conseillé.

absolute	external	pascal	safecall
abstract	far	private	stdcall
assembler	forward	protected	stored
automated	index	public	virtual
cdecl	message	published	write
default	name	read	writeonly
dispid	near	readonly	
dynamic	nodefault	register	
export	override	resident	

3.1.2 Opérateurs de base

Les opérateurs les plus fréquemment utilisés sont :

Nom	**Symbole**	**Exemple ou observation**
Opérateur d'affectation	:=	i := a + b ;
Délimiteur d'instruction	;	Une seule instruction par ligne de code
Commentaire	{ }	ou la séquence (* puis *) moins utilisée ou // pour un commentaire jusqu'à la fin de la ligne
Déclaration de type	:	Uncar : CHAR ;
Egalité	=	VAR = i ;
Différence	<>	VAR <> 1 ;
Opérateurs relationnels	< <= => >	
Opérateurs arithmétique	+ - * / div	La division ' / ' est une division réelle. Pour réaliser une division entière il faut utiliser l'opérateur div
Constante chaîne de caractère	' … '	'bonjour'
Pointeur	^	
Valeur numérique d'un code	#	#97 est équivalent au caractère 'a' : code ASCII 97

Il existe par ailleurs des opérateurs logiques (ou opérateurs booléens) permettant de réaliser des tests conditionnels complexes : and, or, xor

and		
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

or		
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

xor		
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

L'opérateur booléen not permet d'inverser la valeur logique de l'identificateur ou de l'expression qui le suit (négation).

Les opérateurs sont évalués automatiquement avec différents types de priorité. Si l'on souhaite que cet ordre d'évaluation soit modifié il faut forcer l'évaluation avec des parenthèses. Les ordres de priorité sont, du plus prioritaire au moins prioritaire :

Priorité	**Opérateur**
1	not
2	* , / , div , mod , and
3	+, - , or
4	= , <> , < , > , <= , >= , in

3.1.3 Les types

Delphi utilise un grand nombre de types pour définir les différentes variables. Ces types sont ceux du langage Pascal d'origine auxquels ont été rajouté certains types permettant l'accès à l'API Windows. Ces nouveaux types ont tous une origine "langage C" évidente.

▲ **Types simples :**

Type	Portée	Format	Observations
TYPES ENTIERS :			
shortint	-128 .. +127	1 octet	signé
smallint	-32768..32767	2 octets	signé
integer	-32768 ... +32767 -2147486648 ... +2147483647	2 octets 4 octets	suivant le système
longint	-2147486648 ... +2147483647	4 octets	signé
byte	0 .. 255	1 octet	non signé
word	0 .. 65535	2 octets	non signé
cardinal	0..65535 ou 0..2147483647	2 octets 4 octets	suivant le système

Les types entiers génériques sont Integer et Cardinal.

Types réels :

real	$2{,}9\ 10^{-39}$ ..$1{,}7\ 10^{+38}$	6 octets	11 ou 12 chiffres significatifs A n'utiliser que pour assurer la compatibilité avec les versions antérieures
single	$1{,}5\ 10^{-45}$ ..$3{,}4^{+38}$	4 octets	7 ou 8 chiffres significatifs
double	$5.0\ 10^{-324}$ .. $1.7\ 10^{+308}$	8 octets	15 ou 16 chiffres significatifs
extended	3.4×10^{-4932} .. 1.1×10^{4932}	10 octets	19 ou 20 chiffres significatifs
comp	$-2^{63}+1$ .. $2^{63}-1$	8 octets	19 ou 20 chiffres significatifs
currency	-922337203685477.5808 ..922337203685477.5807	8 octets	19 ou 20 chiffres significatifs

Autres types :

boolean		1 octet	false = 0 , true = 1 Il y a en fait 4 types booléens (*)
char	0 .. 255	1 octet	non signé
string			Chaîne de caractères d'une longueur maximale de 255 caractères ou plus suivant option $H.

Les types de caractères fondamentaux sont AnsiChar et WideChar.
Les valeurs du type AnsiChar sont les caractères (en octets), rangés dans l'ordre du jeu de caractères ANSI étendu.
Les valeurs du type WideChar sont les caractères (en mots), rangés dans l'ordre du jeu Unicode. Les 256 premiers caractères du jeu Unicode correspondent aux caractères ANSI.

Char est le type de caractère générique. Dans l'implémentation actuelle de Pascal Objet, Char correspond au type AnsiChar, mais les implémentations pour d'autres CPU ou systèmes d'exploitation peuvent définir Char en WideChar. Lors de la saisie de code qui doit gérer des caractères de n'importe quelle taille, utilisez la fonction standard SizeOf plutôt qu'une constante hard-codée pour la taille de caractère.

(*) Les différents types booléens sont :

- Boolean et ByteBool occupent 1 octet ;
- WordBool occupe 2 octets ;
- LongBool occupe 4 octets ;

Les valeurs FALSE = 0 et TRUE = 1 sont prédéfinies.

Pour assurer la compatibilité avec Windows, une proposition peut renvoyer une valeur quelconque différente de 0 qui sera considérée comme 'vraie'.

Les valeurs sont décimales. Toutefois on peut affecter une valeur hexadécimale en faisant précéder la valeur par le caractère spécial ' $' :

```
Varhex := $1FC4 ;
```

On peut représenter un caractère selon les manières suivantes :

- En utilisant des apostrophes ;
- En déclarant la valeur décimale avec la fonction Chr () ;
- En déclarant la valeur ASCII précédée du symbole # ;
- Pour les code ASCII inférieur à 31 (non imprimables), avec le caractère '^' .

```
'A' = Chr (65)= #65
Escape = ^27
```

Dans les précédentes versions (Delphi 1) deux modèles de génération de code étaient supportés pour effectuer des opérations de type réel :

Virgule flottante logicielle, qui était le mode par défaut ({$N-}): le code généré effectuait tous calculs de type réel dans le logiciel en faisant appel à des routines de la bibliothèque d'exécution. Seul le type real était autorisé, les autres types réels entraînent une erreur de compilation.

Virgule flottante 80x87: dans le mode (utiliser la directive {$N+}), le code généré effectuait tous les calculs de type réel à l'aide des instructions 80x87 et il pouvait utiliser les cinq types réels.
Cette option n'existe plus dans DELPHI 3.

▲ Compatibilité de type

Le Pascal est très rigoureux sur la compatibilité de type entre les différents opérandes intervenant dans une instruction. De ce fait :

- L'évaluation d'une expression (à droite de l'opérateur d'affectation) ne peut être réalisée qu'entre opérandes de même type.

- Une affectation ne peut être réalisée qu'entre une expression évaluée (à droite de l'opérateur d'affectation) et une variable (à gauche) de même type.

Par rapport à d'autres langages, tels le langage C, cette contrainte de compatibilité stricte - si elle assure une meilleure fiabilité - peut être source de lourdeur.

Une extension de la syntaxe Pascal apporte certaines adaptations allégeant ces contraintes : il est aujourd'hui possible de réaliser une affectation entre types différents si le type récepteur est "plus grand" (en taille mémoire) que le type résultant de l'évaluation.

▲ Transtypage

Le transtypage de valeurs consiste à changer un type d'expression en un autre. C'est un moyen mis à la disposition du programmeur pour assurer la compatibilité des opérandes lors de l'évaluation d'une expression.

La valeur résultante s'obtient en convertissant l'expression, ce qui peut impliquer la nécessité de tronquer ou d'étendre la valeur originale si la taille du type spécifié est différente de celle de l'expression. Dans les cas où la valeur est étendue, son signe est toujours préservé.

Le transtypage est réalisé selon la syntaxe suivante :

```
Identificateur_de_type (var_a_transtyper)
```

Exemples: Les instructions suivantes sont un exemple de transtypage de valeurs .

```
Integer ('A') ;          { valeur entière du code ASCII 'A' }
Char (48);                       { valeur ASCII du nombre 48 }
Boolean (0) ;            { = faux }
```

En langage C on appelle cette opération "casting".

▲ Le type variant

Le type Variant est capable de représenter des valeurs qui changent dynamiquement de type. Alors qu'une variable de tout autre type est limitée à ce type de manière statique, une variable de type Variant peut revêtir des valeurs de types différents lors de l'exécution. Le type Variant est surtout utilisé dans des situations où le type actuel (qui doit fonctionner) varie ou est inconnu au moment de la compilation.
L'identificateur prédéfini Variant est utilisé pour indiquer le type variant. Les variants ont les caractéristiques suivantes :

Les variants peuvent contenir des valeurs entières, des valeurs réelles, des valeurs chaîne, des valeurs booléennes, des valeurs date et heure ainsi que des objets OLE Automation. De plus, les variants peuvent contenir des tableaux de dimensions variables avec des éléments de n'importe lequel de ces types.
Exemple:

```
var
V1, V2, V3, V4, V5: Variant;
begin
    V1 := 1;  { Valeur entière }
    V2 := 1234.5678;   { Valeur réelle }
    V3 := 'Salut la Planète'; { Valeur chaîne }
    V4 := '1000';   { Valeur chaîne }
    V5 := V1 + V2 + V4;    { Valeur réelle 2235.5678 }
```

3.1.4 Déclarations de variables

Pascal impose de déclarer les variables, les constantes et les types utilisées dans une fonction ou une procédure avant la première instruction.

Des mots réservés sont utilisés pour déclarer les divers opérandes :

type sert à définir la liste des types définis par l'utilisateur (enregistrements, tableaux, classes et en-têtes de fonctions et procédures en particulier).
const sert à déclarer les constantes.
var sert à déclarer les différentes variables.

Le mot réservé const permet de définir des identificateurs dont on ne peut pas modifier la valeur. Il existe des constantes non typées et des constantes typées. Lorsqu'on utilise une constante non typée, le compilateur choisi "le type le plus probable" en fonction de la valeur servant à l'initialisation.

Exemples :

```
type
    TForm1 = class(TForm) ;
    procedure Memo1DragDrop (Sender, Source: TObject; X, Y:Integer);
const
    MaxData = 1024 * 64 - 16;          { constantes non typées}
    drapeau = TRUE ;
    Message = 'Bonjour tout le monde...';

    Max : integer = 10 ;               { constantes typées }
    Message2 : string [ 30 ] = ' Bonjour tout le monde ' ;

 var
    compteur : integer ;
    a , b : char ;
```

3.2 Tableaux, chaînes de caractères et enregistrements

3.2.1 Les tableaux

Un type tableau est un container regroupant des éléments du même type. Il ne peut dépasser la taille de la pile dans une fonction ou procédure (par défaut 1 Mo modifiable par l'option $M). Il est référencé par le mot réservé array et peut être multidimensionnel.

Bien prendre en considération qu'un tableau est un "type" particulier définit par l'utilisateur (à la différence du langage C).

On déclare un type tableau de la manière suivante :

array [ind_dep .. ind_arr] of type_var

array est un mot réservé
ind_dep : indice de départ , ind_arr : indice d'arrivée
type_var : type des éléments constituant le tableau

L'indice de départ est, la plupart du temps, quelconque. Une fois le type déclaré, il faut déclarer une variable à ce type afin qu'il y ait une réservation mémoire.

Exemples :

```
TabChiffre = array [ 1 .. 100 ] of Integer ;
Cube = array [ 0..1, 0..1, 0..1] of Integer;
MotsCroises = array [ 1 .. 10 , 1 .. 10 ] of String ;
```

On accède alors à un élément du tableau en spécifiant son indice :

```
TabChiffre[3] := 10;          { accès en écriture }
var := Cube[3];               { accès en lecture }
```

3.2.2 Les chaînes de caractères

Pascal propose un type "chaîne de caractères" en utilisant les mots réservés string, ShortString ou AnsiString.

Une chaîne de caractères peut varier de 1 à 255 éléments. On peut l'assimiler à un tableau de caractères de 256 éléments dont le premier (d'indice 0, accessible au programmeur) contient la longueur réelle de la chaîne.

On peut déclarer les chaînes selon les syntaxes suivantes :

```
Chaine1 : String;    { Soit maximum 255 caractères si $H-}
      { Soit maximum 2Go si $H+}
Chaine2 : String[20];    { La chaîne ne peut contenir que 20 caractères }
Chaine3 : ShortString;   { Maximum 255 caractères }
Chaine4 : AnsiString;    { Maximum 2 Go caractères }
```

On peut initialiser les chaînes comme suit :

```
Chaine := 'Bonjour tout le monde';
Chaine := 'Aujourd''hui';       { attention : doublement de
l'apostrophe obligatoire }
MaChaine := '';      { Chaîne nulle }
```

La déclaration d'un type chaîne courte spécifie une longueur maximum comprise entre 1 et 255 caractères. Les variables d'un type chaîne courte peuvent contenir des chaînes de longueur dynamique comprise entre 0 et la longueur maximum déclarée.

Le type prédéfini ShortString indique un type chaîne courte ayant une longueur maximum de 255 caractères.
Exemples:

```
Var
      ch1 : String[80];          { chaîne courte de 80 caractères }
      ch2 : ShortString;         { chaîne courte de taille maximale 255 caractères }
```

Le type chaîne longue est indiqué par le mot réservé string ou par l'identificateur prédéfini AnsiString.

Remarque Si vous changez l'état du commutateur de compilation (chaîne vaste dans Projet | Options | Compilateur) $H en {$H-}, le mot réservé string indique une chaîne courte d'une longueur maximum de 255 caractères. Cependant, l'identificateur prédéfini AnsiString indique toujours le type chaîne longue.

Les chaînes longues sont allouées de manière dynamique et n'ont pas de longueur maximum déclarée. La longueur maximum théorique d'une valeur chaîne longue est de 2 Go (deux giga-octets). En pratique, ceci signifie que la longueur maximum d'une valeur chaîne longue n'est limitée que par la mémoire disponible pour une application.

Récapitulatif sur les chaînes de caractères:

	$H-		$H+		
Déclaration	**Taille Max**	**Mémoire**	**Taille Max**	**Mémoire**	**accès ch[0]**
ch:String	255	4	2Go	4	NON
ch:String[20]	20	21	20	21	OUI
ch:ShortString	255	256	255	256	OUI
ch:AnsiString	2Go	4	2Go	4	NON

3.2.3 Les enregistrements

Un enregistrement regroupe des variables de types différents. C'est un type défini par l'utilisateur déclaré à l'aide du mot réservé record.

Comme pour un tableau, l'utilisation d'un enregistrement se fait en deux phases :

- Dans une première phase on définit le type de l'enregistrement en spécifiant sa constitution interne (ses membres)
- Ensuite on déclare des variables au type de l'enregistrement ainsi défini

On peut initialiser rapidement les membres d'un enregistrement avec l'instruction **with** :

```
with < var_enregistrement > do
begin
    < instructions >
end ;
```

Exemple :

```
type    { mot clé indiquant que l'on va définir un nouveau type }
    Date = record               {mot clé indiquant le nom du type d'enregistrement }
            Jour : Integer;     { déclarations des différents champs }
            Mois : Integer;
            Annee : Integer;
    end;

var DateCourante : Date;    {Déclaration d'une variable du type Date }


with DateCourante do        { 'with' permet d'accéder aux champs}
begin                       { sans avoir à indiquer le nom de l'enregistrement }
    if Mois = 12 then
        begin
            Mois := 1;
            Annee := Annee + 1;
        end
    else
        Mois := Mois + 1;
end;
```

3.3 Autres types complexes

Le langage Pascal permet la définition d'autres types que l'on utilise dans certains cas, lorsque l'on souhaite mieux contrôler les valeurs prises par les variables déclarées à partir de ces types.

3.3.1 Type énuméré

On déclare un type énuméré lorsque l'on souhaite qu'une variable ne puisse prendre qu'un certain nombre de valeurs prédéfinies.

La déclaration du type, d'une variable associée et son utilisation se fait selon la syntaxe :

```
type
    Jour =(Lundi,Mardi,Mercredi,Jeudi,Vendredi,Samedi,Dimanche);
var
    JourSemaine : Jour ;
begin
    JourSemaine := Lundi ;
```

En interne, la première constante de l'énumération vaut 0, la suivante 1, etc…

3.3.2 Type intervalle

On déclare un type intervalle en définissant des bornes de valeurs. Les bornes sont généralement des entiers ou des caractères.

On déclare un tel type, et une variable associée, comme suit :

```
type
    Jour = 1 .. 31 ;
var
    JourDuMois : Jour ;
```

3.3.3 Type ensemble

Un type ensemble d'un type scalaire donné (dénommé type de base) peut contenir tous les sous-ensembles possibles de valeurs du type de base, y compris l'ensemble vide. Ainsi, chaque valeur possible d'un type ensemble est un sous-ensemble de toutes les valeurs possibles du type de base.
Exemple:

```
Var
    Ens1 : Set Of byte;
```

Le type de base ne doit pas contenir plus de 256 valeurs possibles, et les rangs des bornes supérieure et inférieure du type de base doivent être dans l'intervalle 0..255.
Tout type ensemble peut contenir la valeur [ ], qui est dénommée ensemble vide.
Le constructeur d'ensemble est **[]**. Les opérations possibles sur les ensembles sont l'union (+), l'intersection (*), les comparaisons (=,<=,>=), la différence (-).

3.4 Structure d'un programme

Le langage Pascal est très procédurier: les différentes parties contenant des instructions sont précédées d'une partie déclarative qui peut être assez importante et qui est elle-même subdivisée en sections.

Certaines spécificités de la structuration d'un programme seront revues dans la partie consacrée à la "Programmation Delphi" afin de tenir compte des aménagements que cet environnement impose.

3.4.1 Section "en-tête"

L'en-tête permet de donner un nom à un programme. Il commence par le mot réservé program suivi du nom attribué.

```
program Mon_Programme;
```

3.4.2 Section "déclaration"

C'est dans cette partie que les différents types définis par l'utilisateur ainsi que les variables et les constantes qui doivent être visibles par l'ensemble du programme sont déclarés et, le cas échéant, initialisés.

```
type
    Personne = record
                Nom : string[20];
                Prenom : string[20];
                Age : integer;
    end;
procedure afficher(message : string [30]);

var
    Individu : Personne;    { Déclaration d'une variable du type
                              Personne }
    Compteur : integer;
const
    maximum : integer = 100;
```

L'ordre des différents types de déclaration peut être modifié.

A la suite de la section déclaration on place le corps des différentes fonctions ou procédures du programme.
Les en-têtes des fonctions et des procédures doivent être terminés par ';'.

Tous les blocs d'instruction sont délimités par les mots réservés ' begin ' et ' end ; ' (le programme principal commence lui aussi par un begin mais se termine lui par ' end.', avec un point au lieu d'un point virgule).

La tradition du Pascal veut que le corps des procédures et fonctions soient définis avant celui du programme principal. Mais, si les en-têtes des fonctions et procédures sont indiqués dans la partie déclaration, ce n'est pas réellement nécessaire.

Structure générale d'un programme :

```
program demonstration ;{ en-tête }

type { partie déclaration }
      ...
const
      ...
var
      ...

procedure Afficher(...);
Var ...     { déclarations locales à la procédure }

begin             { début du code de la procédure Afficher }
      ...
end ;

begin             { début du programme principal }
      Afficher (...);
      ...
end.
```

3.5 Les unités

Delphi généralise une possibilité fort intéressante de la programmation en Pascal : la programmation multi-fichiers (un programme résulte alors de la compilation et de l'édition des liens de plusieurs fichiers sources). Il est ainsi possible de décomposer le programme en modules distincts que l'on appelle unités, chaque unité correspondant à un fichier source.

On verra plus loin que chaque écran d'une application Delphi constitue une unité particulière.

3.5.1 Définition

Les unités sont à la base de la programmation modulaire. Elles permettent de créer des bibliothèques et de diviser des programmes de grandes tailles en modules reliés logiquement.

Une unité regroupe par affinités des fonctions et des procédures : on peut ainsi concevoir une unité "gestion d'écran" contenant toutes les fonctions nécessaires à cette tâche, une unité "calcul mathématique", etc .... On peut aussi utiliser des unités déjà réalisées et compilées : celles livrées avec l'environnement de développement, qui contiennent l'ensemble des fonctions et procédures standards, et celles que l'on peut acquérir dans le commerce pour réaliser des fonctionnalités spécifiques.

Pour utiliser une unité à partir du programme principal ou d'une autre unité il faut inclure son nom , dans la partie déclaration, dans la clause 'uses' qui indique la liste des unités devant être incluses à la compilation. La syntaxe est la suivante :

uses < nom_unite >

3.5.2 Partie déclarative d'une unité

La partie déclarative d'une unité est constituée, comme pour un programme, de différentes sections aux rôles bien définis.

▲ **En-tête :**
Dans le fichier source des unités, l'en-tête est remplacé par la ligne :

unit Mon_Unité;

▲ **Section interface :**
La section interface d'une unité détermine l'ensemble des identificateurs visibles et accessibles à partir de n'importe quel programme (ou autre unité) utilisant celle-ci. On appelle ces déclarations "déclarations publiques". Cette section commence avec le mot réservé interface, qui apparaît après l'en-tête d'unité, et se termine avant le mot réservé implementation.

L'interface indique en particulier, en utilisant les mots réservés type, var et const selon les cas :

- Les noms des unités utilisées par le programme ou l'unité en cours par la clause 'uses'.
- Les déclarations des types, constantes et variables. Le programme considère ces déclarations comme globales et "exportables".

- Les en-têtes de fonctions et procédures qui constituent l'unité.

Le corps des fonctions et des 'procédures' n'apparaît pas dans la partie interface, mais dans la partie 'implementation' . Cependant comme les en-tête font partie de la section 'interface', toutes les fonctions d'une unité peuvent être invoquées à partir d'une autre unité.

Il est possible de placer une clause 'uses' dans la section 'interface' (si c'est le cas 'uses' doit suivre immédiatement le mot réservé 'interface'). On verra plus loin la nécessité d'une telle déclaration.

▲ **Section implementation :**
La section implementation d'une unité contient un ensemble de déclarations d'identificateurs qui, bien que "globaux", ne seront visibles que de l'intérieur de l'unité (déclarations privées): les utilisateurs "externes" de l'unité ne peuvent donc pas en disposer.

Cette section contient aussi le corps des procédures et des fonctions déclarées dans la partie interface.

Si une fonction ou une procédure n'est utilisée qu'au sein de l'unité, il n'est pas nécessaire que son en-tête figure dans la section 'interface '.

3.6 Les structures de contrôle

Le Pascal, comme tous les langages structurés, offre un certain nombre de possibilités permettant de contrôler le déroulement d'un programme. Il y a lieu de distinguer les structures de sélection (alternative) des structures d'itération (boucles).

3.6.1 La structure de sélection simple " si … alors … sinon "

```
if <condition> then
begin
      ...
      <Instructions cas VRAI>
end
else
begin
      ...
      <Instructions cas FAUX>
end;
```

Si l'expression booléenne après if est True, l'instruction qui suit then est exécutée.
Sinon, l'expression est évaluée False et si la partie else est présente, l'instruction suivant else est exécutée. Si la partie else n'est pas présente, l'exécution continue avec la prochaine instruction après if.

Remarque: il n'est pas permis d'utiliser un point-virgule avant une clause else.

Cette structure permet de réaliser un test conditionnel booléen sur une valeur ou une expression. Elle est spécifiée par le mot réservé " if " . Si la condition est évaluée "vraie", le bloc d'instructions dépendant du ' then ' est exécuté. C'est le bloc dépendant du ' else ' qui est exécuté dans l'autre cas.

Il n'est pas nécessaire d'utiliser ' begin ' et ' end ' s'il n'y a qu'une seule instruction à exécuter .

Il n'y a pas de ' ; ' après le 'end ' du bloc d'instruction dépendant du 'then' sauf si il n'y a pas de traitement ' else ' .

- **A retenir: jamais de ; avant un else**

▲ **Evaluation booléenne d'une expression :**
Delphi permet l'évaluation implicite d'un expression. De ce fait on pourra avoir les expressions suivantes :

Avec A = 1 et C = 0 :

```
{ évaluation explicite }
if (A = 1)
...
{ évaluations implicites }
if (A) { A = 1 est considéré comme "vrai " car la valeur
         booléenne "faux" est assimilée à 0. }

if (C)       { C = 0 est considéré comme "faux " }

if (not A) { évaluation "Faux" car A = 1 est vrai donc son
             inverse est faux }
```

3.6.2 La structure de sélection multiple

Il s'agit d'une généralisation de la structure précédente permettant de proposer plusieurs possibilités de traitements en fonction de la valeur de la variable ou de l'expression testée.

La syntaxe de cette structure est la suivante :

```
case    < selecteur > of
        < valeur_1 > : < traitement 1 > ;
        < valeur_2 > : < traitement 2 > ;
        ...
        < valeur_i > : < traitement i > ;
end ;
```

En fonction de la valeur de l'opérande < selecteur > seul le traitement correspondant, et uniquement celui-ci, sera réalisé.

Pascal autorise les extensions de syntaxe suivantes :
- Utilisation d'intervalles de valeurs tests .
- Utilisation d'une possibilité " autres valeurs" matérialisée par un bloc ' else '.

Exemple :

```
case Note1 of
        0..7 : write (" Nettement insuffisant - Echec");
        8..9 : write (" Rattrapage possible ");
        10..12 : write(" Mention passable");
        13..14 : write(" Mention assez bien");
        15..16 : write(" Mention bien");
        else
                write (" Félicitations du jury ");
end;
```

Les variables testées doivent être de types scalaires. Il ne peut donc pas y avoir de test sur une variable de type chaîne.

3.6.3 Structure itérative "tant que"

Cette structure permet de réaliser un certain nombre de fois un bloc d'instructions tant que la condition testée est réalisée.

Le bloc d'instructions n'est pas exécuté si le test est faux d'emblée.

Dans le cas général, une instruction du bloc doit modifier la condition de sortie de la boucle si l'on ne veut pas que le programme "boucle indéfiniment".

Exemples:

```
While I<10 Do
Begin
    Writeln('I = ',I);
    Inc(I);
End;
```

3.6.4 Structure itérative " répéter ... jusqu'à"

Cette structure permet de réaliser un certain nombre de fois un bloc d'instructions jusqu'à ce que la condition testée soit réalisée.

Contrairement à la structure précédente :

- Le bloc d'instructions est exécuté au moins une fois (avant le test);
- Il y a itération tant que la condition est fausse.

Contrairement aux autres structures de contrôle, la structure repeat ... until ne nécessite pas de begin ... end .

Exemple:

```
Write('Voulez-vous continuer : O/N ');
Repeat
        Reponse:= ReadKey;
        Reponse:=UpCase(Reponse);
Until Reponse in ['0','N'];
```

3.6.5 Structure à itération limitée :

Cette structure permet de réaliser un certain nombre de fois - défini à l'avance - une itération .

La syntaxe est alors :

```
for < initialisation > to < valeur_max > do
begin
        ...
end ;
```

Il faut initialiser une variable de comptage dans <initialisation> (ex : i := 1). Les instructions du bloc seront exécutées tant que cette variable n'atteint pas la valeur <valeur_max> .

Il est possible de "décrémenter" les valeurs de la variable en utilisant le mot réservé ' downto' au lieu de ' to ' (la valeur d'initialisation doit alors être supérieure à la valeur <min> remplaçant <max>).

Le pas d'incrémentation ou de décrémentation de la variable ne peut être que de 1.

3.7 Les pointeurs

3.7.1 Généralités

Un pointeur est une variable spéciale qui permet d'accéder directement à l'emplacement mémoire où est stocké une variable. La valeur d'un pointeur est en effet l'adresse de la variable.

On dit qu'un pointeur "pointe" sur une variable donnée lorsque sa valeur est l'adresse de la variable.

Pour pouvoir utiliser un pointeur il faut :

1. Le déclarer en indiquant le type de la variable pointée.
2. L'initialiser en indiquant quelle est la variable sur laquelle il doit pointer : il faut pour cela fournir l'adresse de la variable.
3. On peut alors utiliser le pointeur pour accéder directement à la zone mémoire de la variable.

La syntaxe étendue du Pascal permet de référencer directement l'adresse d'une variable avec l'opérateur ' @ ' :

@var = adresse de la variable var.

Exemple :

```
var
chaine : string [ 30 ] ;    { déclaration d'une variable de type
                                    chaîne de caractères }
p : ^string ;                       { déclaration d'un pointeur p qui pointera sur une
                                    variable de
                                    type chaîne de caractères }

begin

        p := @chaine ;          { initialisation du pointeur }

end ;
```

Tant qu'un pointeur n'est pas initialisé, il contient une adresse quelconque. Il est conseillé de l'initialiser à 0 pour éviter qu'il ne pointe sur une zone mémoire dangereuse. La constante prédéfinie nil permet cette initialisation.

```
p := nil ;
```

En syntaxe étendue il existe un type de pointeur particulier, le type Pointer, qui permet de déclarer un pointeur sur n'importe quel type de données. Il faut ensuite, dans la plupart des cas, transtyper le pointeur pour l'utiliser sur un type de données particulier.

3.7.2 Utilisation d'un pointeur

Une fois un pointeur défini il est possible de l'utiliser pour accéder directement à la zone mémoire sur lequel il pointe .

On utilise pour cela la syntaxe suivante : point^ = valeur de la variable pointée.

Exemple :

```
var_1, cpt : integer;
point : ^integer;

begin
        var_1 := 10;
        point := @ var_1;
        point^ := 15;           { var_1 vaut 15 }
        cpt := point^;          { cpt vaut 15 }
        ...
```

On a donc les équivalences fondamentales suivantes :

var_1 ⇔ point^ { il s'agit dans les deux cas de valeurs }

@var_1 ⇔ point { il s'agit dans les deux cas d'adresses }

On peut utiliser indifféremment, dans les deux cas, l'une ou l'autre des écritures :

var_1 := var_1 + 10 est équivalent à point^ := point^ + 10

3.7.3 Initialisation d'une zone mémoire par un pointeur

Il est possible d'initialiser directement une zone mémoire qui ne correspond pas à une variable. Dans ce cas on utilise un pointeur pointant vers la zone mémoire et cette dernière ne sera accessible que par lui (cette zone mémoire ne dispose pas de "nom").

Exemple :

```
pchaine : ^string ;
    {pointeur de type chaîne, la chaîne peut atteindre 255 caractères ou 2 Go suivant $H}
...
pchaine^ := 'Bonjour tout le monde';
    { Le message est stocké en mémoire et n'est accessible que par le pointeur }
```

3.7.4 Allocation dynamique

Il est possible de se faire allouer dynamiquement une zone de mémoire en utilisant les pointeurs. On utilise pour cela les procédures New () et Dispose () .

```
procedure New(var P: Pointer);
procedure Dispose(var P: Pointer);
```

Exemple :

```
pointeur : ^integer ;
...
New (pointeur);          { allocation dynamique d'une zone de mémoire de la taille d'un entier }
pointeur^ := 5 ;         { initialisation de la zone mémoire }
...
Dispose (pointeur);      { libération de la zone mémoire }
```

Il peut se produire une erreur d'exécution s'il n'y a pas assez de place dans le tas pour allouer la nouvelle variable. Cependant la directive {$I+} permet de gérer les erreurs d'exécution en utilisant les exceptions (voir chapitres sur les directives et sur les exceptions).

Il est possible d'utiliser les procédures Getmem () et Freemem (), qui nécessitent les mêmes paramètres en plus de la taille réservée ou libérée.

```
procedure GetMem(var P: Pointer; Size: Integer);
procedure FreeMem(var P: Pointer[; Size: Integer]);
```

3.8 Les procédures et les fonctions

Un programme écrit en Pascal peut être découpé en plusieurs sous-programmes afin d'améliorer sa structuration et par là même sa compréhension et sa maintenance. Chaque sous-programme exécute une tâche élémentaire et peut être utilisé plusieurs fois.

Pascal distingue les procédures et les fonctions. Les deux types de sous-programmes sont constitués d'un en-tête et d'un bloc d'instructions.

La différence entre une procédure et une fonction est que cette dernière "renvoie" une valeur. Une fonction apparaît alors comme un opérande particulier au sein d'une instruction.

3.8.1 Structures d'une procédure et d'une fonction :

Les deux types de sous-programme sont identifiés par les mots réservés 'procedure ' et ' function ' au début de l'en-tête. Ils peuvent aussi nécessiter des paramètres pour leur exécution. Il s'agit de paramètres dits "paramètres formels" utilisés tels quels au sein du sous-programme.

Les déclarations sont réalisées comme suit :

```
Procedure Chercher (chaine : string ; i , y : integer );
begin
    ...         { corps de la procédure }
end ;
{ La procédure Chercher a pour paramètres une chaîne de caractères et deux entiers }

Function Addition (a , b : integer): integer;
begin
    ...         { corps de la fonction }
end;

{ La fonction Addition a pour paramètres deux entiers et renvoie un entier }
```

La valeur de retour que doit renvoyer une fonction est matérialisée, dans le corps de la fonction par une variable utilisant le nom de la fonction ou alors par le mot réservé 'result '.

Si une procédure ne demande aucun paramètre, on peut écrire l'en-tête comme suit :

```
Procedure Affichage_sans_param ;
```

Exemple :

```
Function Addition (a , b : integer): integer ;
begin
    Addition := a + b ;
    { Addition représente la valeur renvoyée }
    { ou Result := a + b ; }
end ;
```

La fonction pourra alors être appelée comme suit :

Si resultat est une variable de type Integer:

```
resultat := Addition (x , y);
```

Si une procédure ne nécessite pas de paramètre, elle peut être appelée sans parenthèse.

```
...
AfficheMenu ;
```

3.8.2 Passages de paramètres par valeurs ou par variables :

Les paramètres peuvent être passés, à une fonction ou à une procédure, de deux manières différentes :

- Par valeurs : Le paramètre passé est une copie de la variable. Celle-ci n'est donc pas modifiée après l'exécution de la procédure ou de la fonction.

- Par variables : Le paramètre passé référence directement la zone mémoire où est stockée la variable : celle-ci est donc modifiée lors de l'exécution de la procédure ou de la fonction.

Les différents types de paramètres sont distingués par les mots réservés :

- ' **const** ' (ou rien) pour les paramètres par valeurs;
- ' **var** ' pour les paramètres par variables .

Exemples :

```
Function Addition (const a , b : integer): integer ;
Function Addition (a, b : integer): integer ;

{ Deux déclarations équivalentes avec passage de paramètres par valeurs }

Procedure Addition (a, b: integer; var resultat : integer);
begin
      resultat : = a + b ;
end ;

{ La variable 'resultat' est modifiée lors de l'exécution }
```

Les paramètres qui sont passés à la procédure ou à la fonction sont appelés "paramètres réels". Ils initialisent les paramètres formels définis lors de la construction du sous-programme.

3.8.3 Durée de vie et visibilité des variables :

Pascal distingue les variables globales des variables locales :

Les variables globales sont déclarées en dehors de tout bloc d'instructions (dans la partie déclaration du programme) : elles sont accessibles (= visibles) à partir de toutes les instructions du programme, c'est à dire par l'ensemble des sous-programmes quels que soient leurs niveaux de structuration.

Les variables locales sont déclarées au sein d'une procédure ou d'une fonction. Elles ne sont accessibles qu'à partir des instructions du sous-programme où elles sont déclarées ainsi que à partir des sous-programme de niveau inférieur.

Les variables sont donc visibles dans les sous-programmes où elles ont été déclarées ainsi que dans tous les sous-programmes de niveaux inférieurs. Elles sont "invisibles" pour tous les niveaux supérieurs.

Si on utilise le même identifiant pour définir des variables à des niveaux différents de structuration, la variable de niveau inférieur "cache" la visibilité de la variable de niveau supérieur pour les instructions du sous-programme où elle est déclarée et pour les niveaux inférieurs. La variable de niveau supérieur est donc inaccessible.

Les variables globales sont créées dans le segment de données du programme alors que les variables locales, ainsi que les paramètres utilisés lors d'un appel de procédure ou de fonction, sont créées dans la pile du programme.

Dans le cadre de la programmation multi-fichiers, avec utilisation d'unités, les variables et sous-programmes définis dans la section ' interface ' de l'unité sont visibles à partir du programme ou des autres unités qui invoquent cette unité.

Les variables globales ont une durée de vie égale à celle du programme alors que les variables locales sont créées à chaque entrée dans le bloc d'instructions (c'est à dire à chaque appel du sous-programme) puis détruites à chaque sortie.

Les variables globales, ainsi que les constantes typées, sont créées dans la zone de données du programme alors que les variables locales sont créées dans sa pile.

Les variables déclarées à l'aide des possibilités d'allocation dynamique de mémoire ont une durée de vie correspondant aux instructions déclarées entre la phase d'allocation (new) et la phase de libération de la mémoire (dispose).

3.8.4 Directives de compilation

Les directives de compilation permettent de personnaliser le fonctionnement par défaut du compilateur. Il ne s'agit donc pas d'instructions en langage Pascal mais d'indications spécifiques. Elles sont assimilées à des commentaires utilisant une syntaxe particulière.

On distingue les directives locales et les directives globales :

- Les directives locales peuvent apparaître n'importe où dans une unité d'un programme, et affectent seulement une partie de la compilation.

- Les directives globales doivent apparaître avant la partie de déclaration du programme ou de l'unité. Elles affectent entièrement la compilation.

Les directives de compilation appartiennent à trois catégories :

▲ **Les directives "bascule" :**
Ces directives activent ou désactivent les caractéristiques du compilateur selon qu'il est spécifié ' + ' ou ' - 'après le nom de la directive. On peut regrouper plusieurs directives bascule en les séparant avec des virgules.

Exemple : {$F+,R+,D-}

▲ **Les directives "de paramétrage" :**
Ces directives donnent des informations au compilateur, comme un nom de fichier, du texte ou une taille mémoire. Il doit y avoir au moins un caractère espace entre le nom des directives et ses paramètres.

Exemple : {$I TYPES.INC}
{$L YOUR.DOC}

▲ **Les directives et symboles conditionnels :**
Ces directives contrôlent la compilation conditionnelle de parties du texte source, en fonction de symboles conditionnels définis par l'utilisateur. On peut définir ses propres symboles ou utiliser les symboles prédéfinis du Pascal Objet. Les directives conditionnelles doivent être spécifiées dans du code.

Les principales directives sont :

Directive	**Type**	**Description**
$A	bascule	Aligne les données
$B	bascule	Evaluation booléenne
$D	bascule	Informations de débogage
$D Text	paramètre	Description
$I	bascule	Contrôle des E/S
$I FileName	paramètre	Fichier d'inclusion
$L	bascule	Informations de symboles locaux
$L FileName	paramètre	Lien avec fichier objet
$R FileName	paramètre	Fichier de ressources
$S	bascule	Contrôle de débordement de pile
$X	bascule	Syntaxe étendue
$Y	bascule	Informations de référence aux symboles
$Z	bascule	Types énumérés de taille Word

La plupart de ces directives peuvent être paramétrées à partir de la page Compilateur de la boîte de dialogue Options de projet du menu 'Projet | Option'.

3.9 Evolutions spécifiques à Delphi

Delphi implémente diverses évolutions de la syntaxe du Pascal. On appelle ces évolutions "syntaxe étendue". Les principales évolutions concernent la manipulation des chaînes de caractères (AZT) et les appels de fonction.

Ces extensions sont dues à la nécessité d'appeler les fonctions de l'API Windows écrites en langage C. Si les appels ne posent pas de problèmes lorsqu'il s'agit d'utiliser des variables numériques, les choses sont différentes pour les chaînes de caractères qui ne sont pas gérées du tout de la même manière dans les deux langages.

Il a donc fallu créer de nouveaux types capables de reconnaître les chaînes de type "langage C", c'est à dire des chaînes de tailles indéterminées terminées par un zéro (chaînes dites "AZT" = à zéro terminal).

Pour ce faire on a donc défini les nouveaux types suivants :

▲ **Chaîne à zéro terminal (AZT) :**
Une chaîne AZT consiste en :
Une suite de caractères non nuls suivis du caractère NULL (#0).

Les chaînes AZT sont stockées sous forme de tableaux de caractères avec un indice à base zéro (obligatoirement):

```
array [ 0..X ] of Char ;
```

C'est ce que l'on appelle un tableau de caractères à base zéro (ABZ).

▲ **Le type PChar :**
Le type prédéfini PChar représente un pointeur référençant une chaîne à zéro terminal. L'unité Système déclare PChar comme :

```
type PChar = ^Char ;
```

Ce type étant prédéfini, Delphi supporte un ensemble de règles de syntaxe étendue facilitant la gestion des chaînes à zéro terminal au moyen du type PChar.

Lorsque la syntaxe étendue est active on peut affecter une chaîne à une variable de type PChar. L'effet d'une telle affectation est de faire désigner par le pointeur une zone de la mémoire contenant une copie à zéro terminal (AZT) du littéral chaîne.

Exemple :

```
var
    P: PChar ;
begin
    P := 'Bonjour tous' ;
end;
```

On peut aussi initialiser un paramètre formel de type PChar avec une chaîne :

```
Procedure AfficheMessage (P : PChar);      { Déclaration }
...
AfficheMessage ('Ceci est un message');    { Appel de la procédure }
```

Enfin on peut initialiser une variable de type PChar sur un tableau de caractères à base zéro:

```
var
     A: array [0..63] of Char;       { tableau de caractères à base 0 }
     P: PChar ;
begin
     P := A;    {P pointe sur le premier élément de A}
     Affichechaine (A);
     AfficheChaine (P); { Les deux appels sont équivalents}
end ;
```

Lorsqu'il est initialisé sur un tableau à base 0, un pointeur de type PChar peut être indicé pour permettre le déplacement dans le tableau.

Si on reprend l'exemple précédent, P[10] pointe sur le 11° caractère de la chaîne (P[0] est le 1° caractère).

▲ **Arithmétique sur les pointeurs :**
Il est possible de se déplacer au sein d'une zone mémoire en réalisant des opérations arithmétiques sur un pointeur (une fois que celui-ci a été initialisé).

```
pt := pt + 1 ; { fait déplacer le pointeur sur l'octet en mémoire suivant }
```

Il est important de remarquer que, quel que soit le type de la variable pointée, le déplacement du pointeur est réalisé par octet. Cela à la différence du langage C où le déplacement est réalisé " par unité de type pointé " (déplacement octet par octet pour un pointeur sur un caractère, de deux (ou quatre) octets pour des pointeurs sur des entiers, etc…).

Pascal permet une hiérarchisation poussée du programme et des sous-programmes: le programme est donc constitué de sous-programmes, eux même pouvant être constitués de sous-sous-programme, etc ... On parle alors de "niveaux de structuration" : chaque sous-programme est d'un niveau inférieur à celui du sous-programme appelant et est, le cas échéant, de niveau supérieur aux sous-programmes qu'il appelle.

4 Fonctions et procédures Pascal

Delphi, avec ses palettes de composants, offre un ensemble de possibilités qui permettent de créer rapidement l'essentiel de nombreuses applications. Néanmoins il est parfois nécessaire de recourir aux fonctions et procédures fournies en standard par le Pascal, en particulier pour tout ce qui est traitement interne (conversions de données, accès aux fichiers disque, allocations de ressources diverses, etc…).

La plupart de ces fonctions et procédures sont contenues dans les unités suivantes :

L'unité System implémente les routines de bas niveau pour toutes les fonctionnalités intégrées, telles que les E/S sur fichiers, la gestion des chaînes, les routines en virgule flottante et l'allocation dynamique de la mémoire.

L'unité SysUtils comprend les déclarations des classes d'exceptions, les routines de chaînes, les routines manipulant la date et l'heure et les routines utilitaires.

L'unité Windows définit les en-têtes de fonction et de procédure de l'API Windows. L'utilisation de cette unité permet d'accéder aux fonctions du SDK de Windows.

- Pour utiliser des routines de l'unité System ou SysUtils il n'est pas besoin de référencer celle-ci dans la clause 'uses' du programme car ces unités sont incorporées d'office.

Pour des raisons de compatibilité avec les versions précédentes, un alias est défini dans 'Outil | Options d'environnement | Bibliothèque' pour remplacer les unités WinProcs et WinTypes.

4.1 Routines de conversion de données

function Chr (X: Byte) : Char ;
La fonction Chr renvoie le caractère dont la valeur numérique (le code ASCII)correspond à l'expression de type Byte X.

function Int (X: Real): Real ;
La fonction Int renvoie la partie entière de son argument, sous forme d'un nombre réel.

function IntToStr (Valeur: Longint): string;
La fonction IntToStr convertit un entier en une chaîne contenant la représentation décimale de ce nombre.

procedure Str (X [: Width [: Decimals]]; var S);
La procédure Str convertit X en une représentation chaîne en utilisant les paramètres de formatage Width et Decimal.

X est une expression de type réel ou entier. Width et Decimals sont des expressions de type entier. S est une expression de type chaîne ou une expression de type PChar si la syntaxe étendue est autorisée.

function StrToInt (const S: string): Longint;
La fonction StrToInt convertit une chaîne représentant un nombre de type entier avec une notation (soit décimale, soit hexadécimale) en un nombre.
Si la chaîne ne représente pas un nombre valide, StrToInt provoque une exception EConvertError.

FUNCTION STRTOINTDEF(CONST S: STRING; DEFAULT: INTEGER): INTEGER;

Si S ne représente pas un nombre valide, StrToIntDef renvoie le nombre transmis par Default.

function IntToHex(Value: Integer; Digits: Integer): string;
La fonction IntToHex convertit un nombre en une chaîne contenant la représentation hexadécimale de ce nombre (base 16). Valeur est le nombre à convertir. Digits indique le nombre minimum de chiffres hexadécimaux à renvoyer.

function FloatToStr(Value: Extended): string;
FloatToStr convertit la valeur flottante Value en une représentation de type chaîne. La conversion utilise le format général des nombres avec 15 chiffres significatifs.

function FloatToStrF(Value: Extended; Format: TFloatFormat; Precision, Digits: Integer): string;
FloatToStrF convertit la valeur flottante Valeur en une représentation de type chaîne.
Le paramètre Format contrôle le format de la chaîne résultante (ffGeneral, ffExponent, ffFixed, ffNumber, ffCurrency :cf Aide en ligne)
Le paramètre Precision indique la précision de la valeur. Elle doit être inférieure ou égale à 7 pour les valeurs de type Single, à 15 pour les valeurs de type Double, et à 18 pour les valeurs de type Extended.
La signification du paramètre Digits dépend du format sélectionné

function FloatToText(Buffer: PChar; const Value; ValueType: TFloatValue; Format: TFloatFormat; Precision, Digits: Integer): Integer;
La fonction FloatToText convertit une valeur flottante en sa représentation décimale en utilisant le format, la précision et le nombre de chiffres indiqués. La valeur Value doit être une variable de type Extended ou Currency, comme indiqué par le paramètre ValueType. La chaîne résultante est stockée dans le tampon spécifié et la valeur renvoyée correspond au nombre de caractères qui y sont stockés. La chaîne renvoyée ne possède pas de caractère de fin nul.

procedure Val (S ; var V; var Code: Integer);
La procédure Val convertit la valeur chaîne S en sa représentation numérique. S est une expression de type chaîne qui doit former un nombre entier signé. V est une variable de type réel ou entier. Code est une variable de type Integer.

Si la chaîne est incorrecte, l'indice du caractère posant problème est placé dans Code, sinon Code est initialisé à zéro. Dans le cas d'une chaîne à zéro terminal, la position d'erreur renvoyée dans Code est supérieure de un à l'indice à base zéro réel du caractère erroné.

FUNCTION FORMATFLOAT(CONST FORMAT: STRING; VALUE: EXTENDED): STRING;

FormatFloat définit le format de la valeur flottante Value en utilisant la chaîne de format fournie par Format. Les spécificateurs de format suivants sont reconnus dans la chaîne de format :

Spécificateur	Signification
0	Emplacement d'un chiffre. Si la valeur formatée dispose d'un chiffre à la position du '0' dans la chaîne de format, ce chiffre est copié dans la chaîne en sortie. Sinon, un '0' est inclus à cette position dans la chaîne en sortie.
#	Emplacement d'un chiffre. Si la valeur formatée dispose d'un chiffre à la position du '#' dans la chaîne de format, ce chiffre est copié dans la chaîne en sortie. Sinon, rien n'est inclus à cette position dans la chaîne en sortie.
.	Séparateur décimal. Le premier caractère '.' dans la chaîne de format détermine l'emplacement du séparateur décimal dans la valeur formatée ; les caractères '.' supplémentaires sont ignorés. Le caractère effectivement utilisé comme séparateur décimal dans la chaîne en sortie est déterminé par la variable globale DecimalSeparator. La valeur par défaut de DecimalSeparator est définie dans le groupe Format des nombres de la section International du Panneau de configuration Windows.
,	Séparateur des milliers. Si la chaîne de format contient un ou plusieurs caractères ',', un séparateur des milliers sera inséré entre chaque groupe de trois chiffres situés à gauche de la virgule. L'emplacement et le nombre de caractères ',' dans la chaîne de format n'affecte en rien la sortie ; la présence de ce caractère indique simplement que les milliers doivent être séparés. Le caractère effectivement utilisé comme séparateur décimal dans la chaîne en sortie est déterminé par la variable globale ThousandSeparator. La valeur par défaut de ThousandSeparator est définie dans le groupe Format des nombres de la section International du Panneau de configuration Windows.
E+	Notation scientifique. Si l'une des quatre chaînes 'E+', 'E-', 'e+' ou 'e-' est incluse dans la chaîne de format, le nombre est formaté en utilisant la notation scientifique. Un groupe de quatre caractères '0' au plus peut immédiatement suivre la chaîne 'E+', 'E-' 'e+' ou 'e-' afin de déterminer le nombre de chiffres de l'exposant. Les spécificateurs de format 'E+' et 'e+' provoquent l'apparition d'un signe plus pour les exposants positifs et d'un signe moins pour les exposants négatifs. 'E-' et 'e-' ne provoque l'apparition du signe que pour les exposants négatifs.
'xx'/"xx"	Les caractères compris entre des guillemets simples ou doubles sont renvoyés tels quels et n'affectent pas le format.
;	Sépare les sections des nombres positifs, négatifs et nuls dans la chaîne de format.

4.2 Routines de gestion de fichiers sur disque

Pascal fait la différence entre :

Les fichiers structurés : ils sont constitués par des ensembles de données de même type (composés par des enregistrements);
Les fichiers texte : ils contiennent des lignes de texte non structurés (mais on reconnaît les caractères de délimitation de ligne);
Les fichiers binaires qui n'ont pas de structure définie.

- Quel que soit le type de fichier, les accès sont "bufferisés" c'est à dire que les transferts de données se font via des buffers.

- Pascal permet de réaliser des accès séquentiels sur les fichiers structurés

4.2.1 Manipulation des fichiers

Pascal met à la disposition des programmeurs des variables de types spéciaux , appelées "variables de fichier" permettant d'accéder d'une manière "transparente" aux fichiers sur disques.

Ces variables peuvent être considérées comme des "descripteurs de fichiers" (handles). Il s'agit en fait de pointeurs sur des structures (gérées par le système) contenant tous les renseignements nécessaires à la manipulation du fichier sur le disque.

Lorsque l'on souhaite accéder à un fichier disque, il faut dans un premier temps l'associer au nom physique du fichier (connu sous la forme d'une chaîne de caractères). On pourra ensuite réaliser toutes les opérations souhaitées sur le fichier en n'invoquant que la variable fichier.

Chaque type de fichier (structuré, texte ou binaire) dispose d'un descripteur de type particulier.

▲ **Cas d'un fichier structuré :**
Pour gérer les accès à un fichier structuré Pascal utilise une variable de fichier nommée File que l'on déclare et utilise comme suit :

```
DescF : File of < type d'enregistrement>
```

Exemple :

```
type
    Fiche = Record
        nom : string [ 20 ] ;
        age : byte ;
    end ;
...
Var
    PDesc : File of Fiche ;
{ Déclaration d'un descripteur sur un fichier structuré de type Fiche }
```

▲ **Cas d'un fichier texte :**
Selon les mêmes principes on utilise un type de "variable de fichier texte", appelé TextFile. Comme il n'y a pas d'idée de type de contenu, la déclaration est simplifiée :

```
var
    DescF : TextFile;
```

▲ **Cas d'un fichier binaire :**
Pour accéder à un fichier binaire il suffit de déclarer un descripteur selon la syntaxe :

```
var
    DescF : File ;
```

4.3 Opérations de manipulation sur les fichiers

Pour pouvoir accéder à un fichier, de quelque type qu'il soit, il faut réaliser les opérations suivantes :

L'assignation : cette opération permet d'associer une variable fichier (de type File ou TextFile) au fichier physique connu par son nom.Si l'opération réussit, les accès aux fichiers sont pris en charge par le système d'exploitation.

L'ouverture ou la création du fichier.

Les opérations de lecture ou d'écriture.

La fermeture du fichier.

Delphi propose plusieurs séries de fonctions pour réaliser les opérations sur disque. Chaque série de fonctions est adaptée à la manipulation d'un type particulier de fichier. Des fonctions spécifiques à Delphi ont été rajoutées (par leurs modes de fonctionnement elles ressemblent aux fonctions équivalentes du langage C) :

Opération	Fichier structuré	Fichier texte	Fichier binaire	Fonct. Delphi
Assignation	Assign, AssignFile	Assign, AssignFile	Assign, AssignFile	
Ouverture ou création	Rewrite, Reset	Rewrite, Reset	Rewrite, Reset	FileOpen, FileCreate
Lecture	Read	Read, Readln	BlockRead	FileRead
Ecriture	Write	Write,Writeln	BlockWrite	FileWrite
Déplacement	FilePos, FileSeek; Seek	Append	Seek	FileSeek,
Fermeture	Close, CloseFile	Close, CloseFile	Close, CloseFile	FileClose
Divers	Eof	Eof	Eof	FileExists

4.4 Routines dérivées du Pascal

L'ensemble des fonctions permettant la gestion des disques (Assign, Close, etc ...) sont utilisables avec Delphi pour assurer la compatibilité ascendante. Mais il est préférable d'utiliser les fonctions spécifiques suivantes :

La directive de compilation {$I+} permet de gérer les erreurs d'exécution en utilisant des exceptions. Si la directive {$I-} est définie, il faut utiliser IOResult pour tester les erreurs d'E/S.

procedure AssignFile (var DescF , String);
AssignFile associe une variable fichier (de n'importe quel type) DescF et un nom de fichier externe. String est une expression chaîne ou une expression de type PChar si la syntaxe étendue est autorisée. Toutes les opérations ultérieures sur DescF porteront sur le fichier externe jusqu'à la fermeture de DescF.

```
var
    F: TextFile;
    S: String;
begin
    if OpenDialog1.Execute then { boîte de dialogue d'ouverture de fichier }
    begin
        AssignFile (F, OpenDialog1.FileName );
        { Associe le fichier sélectionné dans la boîte de dialogue }
        Reset (F );      { Ouvre le fichier }
        ...;
        CloseFile (F );
    end;
end.
```

Un nom de fichier consiste en un chemin d'accès, composé ou non de noms de répertoires, séparés par des caractères barre inverse (\), suivi du nom réel du fichier :

Lecteur:\Répertoire\...\Répertoire\NomFichier

Si le chemin d'accès débute par une barre oblique inversée, il commence dans le répertoire racine . Sinon il part du répertoire en cours.

'Lecteur' est un identificateur d'unité de lecteur (A-Z). Si 'Lecteur' et le caractère deux points (:) sont omis, le lecteur par défaut est utilisé. \Répertoire\...\Répertoire est le répertoire racine et le chemin de

sous-répertoires pour accéder au fichier. 'NomFichier' du type String ou Pchar est composé du nom de fichier, suivi éventuellement d'un point et d'une extension de 1 à 3 caractères.

procedure FileClose (Descripteur : Integer);
La procédure FileClose ferme le fichier spécifié.

function Eof
Fichier typés et sans type : function Eof (var DescF): Boolean;

Fichiers Texte: function Eof [(var DescF: Text)]: Boolean;

La fonction Eof teste si le pointeur de fichier est placé sur la fin de fichier. DescF est une variable fichier texte. Eof renvoie True si le pointeur de fichier est au-delà du dernier caractère du fichier ou si le fichier ne contient pas de composants , sinon Eof renvoie False.

procedure Erase (var DescF);
La procédure Erase supprime le fichier externe associé à la variable fichier DescF.

Il faut toujours fermer un fichier avant de le supprimer.

function FilePos (var DescF): Longint;
Renvoie la position du pointeur de fichier dans un fichier. Pour utiliser FilePos, le fichier doit être ouvert. FilePos ne peut être utilisée sur un fichier texte non structuré.

DescF est une variable fichier.

Position	Résultat
Début de fichier	FilePos(F) = 0
Milieu de fichier	FilePos(F) = position en cours dans le fichier
Fin de fichier	Eof(F) = True

function FileSize (var DescF): Longint;
La fonction FileSize renvoie la taille du fichier DescF. Pour utiliser FileSize, le fichier ne doit pas être de type texte et être ouvert.

DescF est une variable fichier.

procedure Flush (var DescF: Text);
La procédure Flush vide le tampon d'un fichier texte ouvert en écriture. DescF est une variable fichier texte.

Lorsqu'un fichier texte a été ouvert en écriture avec Rewrite ou Append, Flush vide le tampon du fichier. Cela assure que tous les caractères écrits dans le fichier à ce moment sont réellement écrits dans le fichier externe.

procedure Read
fichiers typés : procedure Read (F , V1 [, V2,...,Vn]);
fichiers texte : procedure Read ([var F: Text;] V1 [,V2,...,Vn]);

La procédure Read peut être utilisée :

- Dans le cas des fichiers typés, pour lire le composant d'un fichier dans une variable.
- Dans le cas des fichiers texte, pour lire une ou plusieurs valeurs dans une ou plusieurs variables.

Avec une variable de type chaîne, Read lit tous les caractères jusqu'au prochain marqueur de fin de ligne (exclu) ou jusqu'à ce que Eof (F) renvoie True, ne passe pas à la ligne après la lecture. Si la chaîne résultante dépasse la taille maximale autorisé pour une variable chaîne, elle est tronquée.

Après le premier appel de Read, chaque appel ultérieur de Read butte sur le marqueur de fin de ligne et renvoie une chaîne de longueur nulle.

Il faut utiliser plusieurs appels de Readln pour lire des valeurs chaînes successives.

Lorsque la syntaxe étendue est autorisée, Read peut également lire des chaînes à zéro terminal dans des tableaux à base zéro.

Avec les variables de type entier ou réel :

- Read saute les espaces, tabulations ou marqueurs de fin de ligne placés avant la chaîne numérique.
- Si la chaîne numérique ne respecte pas le format attendu, il se produit une erreur d'E/S, sinon la valeur est affectée à la variable.

L'appel suivant de Read commence sur l'espace, la tabulation ou le marqueur de fin de ligne qui terminait la précédente chaîne numérique.

```
var
    F1, F2: TextFile;
    Ch: Char;
begin
    if OpenDialog1.Execute then
    begin
        AssignFile (F1, OpenDialog1.Filename);
        Reset (F1);
        if SaveDialog1.Execute then
        begin
            AssignFile (F2, OpenDialog1.Filename);
            Rewrite (F2);
            while not Eof (F1) do
            begin
                Read (F1, Ch );
                Write (F2, Ch );
            end;
            CloseFile (F2 );
        end;
        CloseFile (F1 );
    end;
end.
```

procedure Readln ([var F: Text;] V1 [, V2, ...,Vn]);
La procédure Readln lit une ligne de texte puis passe à la ligne suivante du fichier.

Readln () sans paramètre, provoque le passage du pointeur de fichier au début de la ligne suivante s'il y en a une, sinon sur la fin de fichier.

procedure Reset (var DescF [: File; Recsize: Word]);
La procédure Reset ouvre un fichier existant. DescF est une variable de tout type de fichier associée à un fichier externe par Assign. RecSize est une expression optionnelle qui peut être spécifiée uniquement si DescF est un fichier sans type. Si DescF est un fichier sans type, RecSize spécifie la taille d'enregistrement à utiliser pour les transferts de données. Si RecSize est omis, une taille d'enregistrement de 128 octets est utilisée par défaut.

Reset ouvre le fichier externe existant dont le nom a été affecté à DescF. Il se produit une erreur s'il n'existe pas de fichier externe portant ce nom. Si DescF est déjà ouvert, il est tout d'abord fermé puis ré-ouvert. Le pointeur de fichier est placé au début du fichier.

procedure Rewrite (var F: File [; Recsize: Word]);
La procédure Rewrite crée et ouvre un nouveau fichier.

F est une variable de tout type de fichier associée à un fichier externe par Assign. RecSize est une expression optionnelle qui peut être spécifiée uniquement si F est un fichier sans type. Si F est un fichier sans type, RecSize spécifie la taille d'enregistrement à utiliser pour les transferts de données. Si RecSize est omis, une taille d'enregistrement de 128 octets est utilisée par défaut. Rewrite crée un nouveau fichier externe portant le nom affecté à F.

S'il existe déjà un fichier externe de même nom, il est effacé et un nouveau fichier vide est créé à sa place.

Si F est déjà ouvert, il est tout d'abord fermé puis ré-ouvert. Le pointeur de fichier est placé au début du fichier vide. Si F est un ficher texte, F passe en lecture seule.

procedure Seek (var F; N: Longint);
La procédure Seek déplace le pointeur de fichier sur le composant spécifié. Seek ne peut être utilisée que sur des fichiers, typés ou non, ouverts.

F est une variable fichier avec ou sans type et N est une expression de type LongInt. Le pointeur de fichier de F est déplacé sur le composant numéro N. Le numéro du premier composant d'un fichier est 0.

L'instruction Seek (F, FileSize (F)) déplace le pointeur de fichier à la fin de fichier.

procedure Write
Pour des fichiers texte : procedure Write ([var F: Text;] P1 [,P2,...,Pn]);
Pour des fichiers typés :procedure Write (F, V1 [V2,...Vn]);

La procédure Write écrit des valeurs dans un fichier. Si F est omis, la variable fichier standard Output est utilisée.

Pour les fichiers texte :
F spécifie une variable fichier texte qui doit être ouvert en écriture.
Chaque P est un paramètre d'écriture comportant une expression de sortie dont la valeur doit être écrite dans le fichier. Un paramètre d'écriture peut également comporter des spécifications de largeur du champ et du nombre de décimales.

Chaque expression en sortie doit être du type Char, l'une de type entière (Byte, Shortint, Word, Longint, Cardinal) , flottante (Single, Real, Double, Extended, Currency) , chaîne (PChar, AnsiString, ShortString) , chaîne compressée (packed) ou de type booléen (Boolean, Bool).

Pour les fichiers typés :
F spécifie une variable fichier typé (FILE OF..).
Chaque V est une variable du même type que le type de composant de F.

- Pour chaque variable écrite, le pointeur de fichier est avancé sur le composant suivant.
- Si le pointeur de fichier est sur la fin de fichier, le fichier est agrandi.

procedure Writeln ([var F: Text;] P1 [, P2, ...,Pn]);
La procédure Writeln est une extension de la procédure Write telle qu'elle est définie pour les fichiers texte. Après exécution de Write, Writeln écrit un marqueur de fin de ligne (retour chariot-passage à la ligne) dans le fichier. Writeln () sans paramètre écrit un marqueur de fin de ligne dans le fichier. Writeln sans liste de paramètre correspond à Writeln (Output). Le fichier F doit être ouvert en écriture.

Exemple général d'une fonction réalisant la copie d'un fichier :

```
procedure CopieFichier (FichierDest , FichierSour : string);
var
      descS , descd : File ;
      buffer : array [ 1 .. 4096 ] of byte ;
      Lire, Ecrire : word ;
      ModeFichier : byte ;
begin
      ModeFichier := FileMode ; { sauvegarde du mode actuel }
      FileMode := 0 ;             { tous les fichiers ouverts avec reset
  sont en lecture seule }
      try
            try
                  Assign (descS , FichierSour);
                  Reset (descS , 1);
                  Assign (descD , FichierDest);
                  Rewrite (descD , 1);
                  while not Eof (descS)do
                  begin
                        BlockRead (descS , buffer , sizeof (buffer), Lire);
                        BlockWrite (descD, buffer , Lire , Ecrire);
                  end ;
                  Close (descS);
                  Close (descD);
            except
                  on EInOutError do
                  MessageDlg (' Erreur lors de la copie de ' + FichierSour + ' ! ' , mtError , [ bCancel] , 0);
            end ;
      finally
            FileMode := ModeFichier ;
      end ;
end ;
```

4.4.1 Nouvelles routines (dérivées du langage C)

function DeleteFile (const Fichier: string): Boolean;
Efface le fichier nommé Fichier du disque. Si le fichier ne peut pas être effacé ou s'il n'existe pas, la fonction renvoie False mais ne provoque pas d'exception.

function FileCreate (const Fichier: string): Integer;
Crée un nouveau fichier spécifié par Fichier. Si la fonction renvoie une valeur positive, cela indique un bon fonctionnement, et la valeur représente le descripteur du nouveau fichier. Sinon, une valeur négative indique qu'une erreur est survenue, et la valeur représente la valeur négative d'un code d'erreur DOS.

procedure FileClose (Descripteur: Integer);
Ferme le fichier spécifié.

function FileExists (const Fichier: string): Boolean;
Renvoie True si le fichier spécifié par Fichier existe. Sinon, FileExists renvoie False.

function FileOpen (const Fichier: string; Mode: Word): Integer;
Ouvre le fichier Fichier selon le mode Mode . Renvoie un descripteur avec lequel le fichier sera référencé pour toutes les opérations ultérieures.

```
MonDescript := FileOpen('EXISTS.TXT', mode);
```

Les constantes de mode d'ouverture des fichiers sont déclarés dans l'unité SysUtils

```
const fmOpenRead        $0000;
const fmOpenWrite       $0001;
const fmOpenReadWrite   $0002;
const fmShareCompat     $0000;
const fmShareExclusive  $0010;
const fmShareDenyWrite  $0020;
const fmShareDenyRead   $0030;
const fmShareDenyNone   $0040;
```

function FileRead (Descripteur: Integer; var MemTampon; Nbre:Integer): Integer;
Lit Nbre octets du Descripteur et les met dans MemTampon. Le résultat de la fonction est le nombre réel d'octets lus, qui peut être inférieur ou égal à Nbre.

function FileWrite (Descripteur: Integer; const MemTampon; Nbre: Integer): Integer;
La fonction FileWrite écrit Nbre octets à partir de MemTampon dans le fichier spécifié par Descripteur. Le nombre réel d'octets écrits est renvoyé. Si la valeur renvoyée n'est pas égale à Nbre, cela est habituellement dû au fait que le disque est plein.

Il existe aussi une méthode dont l'utilisation est particulièrement aisée : la méthode LoadFromFile dont le prototype est :

LoadFromFile (const FileName : string)

Cette méthode charge le fichier dont le nom est spécifié en paramètre sans que l'on ait à se préoccuper d'assigner au préalable celui-ci, ce qui évite l'emploi de variable fichier.

Cette méthode est contenue dans de nombreux composants de Delphi (ce qui veut dire qu'elle ne peut pas être utilisée seule dans un code). Elle adapte son mode de fonctionnement au type de composant qui l'invoque et de ce fait est en mesure de charger un fichier texte ou un fichier binaire en fonction du contexte.

4.5 Gestion des fichiers et répertoires

procedure ChDir (Chemin : String);
Définit le répertoire actif comme étant celui spécifié par le chemin d'accès Chemin . Si une unité de disque est spécifiée dans le nom du chemin , le lecteur actif est également changé.

La directive de compilation {$I+} permet de gérer les erreurs d'exécution en utilisant des exceptions. Si la directive {$I-} est définie, il faut utiliser IOResult pour tester les erreurs d'E/S.

```
begin
        {$I-}
        ChDir (Edit1.Text);
        { Passe au répertoire spécifié dans Edit1 }
        if IOResult <> 0 then
                MessageDlg (' Répertoire non trouvé ',
                mtWarning, [mbOk], 0 );
        {$I+}
end;
```

procedure GetDir (D: Byte; var S: String);
La procédure GetDir renvoie le répertoire en cours dans le lecteur spécifié.

D peut prendre l'une des valeurs suivantes :

Valeur	**Lecteur**
0	par défaut
1	A
2	B
3	C
etc.	

Cette procédure n'effectue pas de vérification d'erreur. Si le lecteur spécifié par D est incorrect, S renvoie X:\ comme si c'était le répertoire racine du lecteur incorrect.

Cette procédure ne fonctionne pas lorsque l'on est en mode "debogage". Le répertoire en cours renvoyé est celui de l'environnement Delphi. Pour que les valeurs fournies soient correctes il faut lancer l'exécutable en dehors de l'environnement de développement.

```
var
        repcour : String ;
begin
        GetDir(0, repcour);                { 0 = Lecteur courant }
        MessageDlg ('Lecteur et répertoire en cours: ' + repcour,
        mtInformation, [mbOk] , 0);
end.
```

procedure MkDir (Rep : String);
Crée un nouveau sous-répertoire avec le chemin d'accès spécifié par la chaîne Rep. Le dernier élément du chemin d'accès ne peut pas être un fichier existant.

CreateDir a le même rôle que MkDir mais utilise des chaînes à zéro terminal au lieu de chaînes Pascal.

procedure Rename (var F; Newname);
La procédure Rename modifie le nom d'un fichier externe. F est une variable de tout type de fichier. Newname est une expression de type chaîne ou une expression de type PChar si la syntaxe étendue est autorisée.

Le fichier externe associé à F est renommé en Newname. Les opérations ultérieures sur F agissent sur le fichier externe portant le nouveau nom.

```
var
        f : file;
begin
        OpenDialog1.Title := 'Choisir un fichier... ';
        if OpenDialog1.Execute then
        begin
                SaveDialog1.Title := 'Renommer en...';
                if SaveDialog1.Execute then
                begin
                        AssignFile (f, OpenDialog1.FileName);
                        Canvas.TextOut (5, 10, 'Renommer '
                                + OpenDialog1.FileName
                                + ' en ' + SaveDialog1.FileName );
                        Rename(f, SaveDialog1.FileName );
                end;
        end;
end.
```

procedure RmDir (S: String);
RmDir supprime le sous-répertoire de chemin d'accès spécifié par S. Il se produit une erreur d'E/S si le chemin d'accès n'existe pas, est non vide ou si c'est le répertoire en cours.

4.6 Gestion des chaînes de caractères

4.6.1 Chaînes de type Pascal

function CompareStr (const S1, S2: string): Integer;
CompareStr compare S1 et S2, avec une distinction majuscule/minuscule. La valeur renvoyée est inférieure à 0 si S1 est inférieure à S2, égale à 0 si S1 égal S2, ou supérieure à 0 si S1 est supérieure à S2. L'opération de comparaison est basée sur la valeur ordinale 8-bits de chaque caractère et n'est pas affectée par le pilote de langage installé.

function Concat (s1 [, s2,..., sn]: String): String;
La fonction Concat fusionne plusieurs chaînes en une grande chaîne. Chaque paramètre doit être une expression de type chaîne. Le résultat est la concaténation de tous les paramètres chaîne. Si la chaîne résultante excède 255 caractères, elle est tronquée après le 255ème caractère.

L'utilisation de l'opérateur plus (+) a le même effet sur deux chaînes que l'utilisation de la fonction Concat :

```
S := 'ABC' + 'DEF';
```

function Copy (S: String; Index, Count: Integer): String;
La fonction Copy renvoie une sous-chaîne d'une chaîne.
S est une expression de type chaîne. Index et Count sont des entiers. Copy renvoie une chaîne de Count caractères en commençant à S [Index].

Si Index est plus grand que la longueur de S, Copy renvoie une chaîne vide.

Si Count spécifie davantage de caractères qu'il y en a de disponibles, seuls les caractères de S [Index] jusqu'à la fin de S sont renvoyés.

function Length (S: String): Integer;
La fonction Length renvoie la taille réelle de la chaîne S.

function Pos (Substr : String; S: String) : Byte;
La fonction Pos recherche une sous-chaîne dans une chaîne.
Substr et S sont des expressions de type chaîne. Pos recherche Substr dans S et renvoie une valeur entière donnant l'indice du premier caractère de Substr dans S. Si Substr n'est pas trouvée, Pos renvoie zéro.

4.6.2 Chaînes à zéro terminal

function StrCat (Dest, Source: PChar): PChar;
La fonction StrCat ajoute une copie de Source à la fin de Dest et renvoie la chaîne concaténée.
StrCat n'effectue aucune vérification de taille.

function StrComp (Str1, Str2 : PChar): Integer;
La fonction StrComp compare Str1 et Str2.

Valeur renvoyée	Condition
<0	si Str1 < Str2
=0	si Str1 = Str2
>0	si Str1 > Str2

function StrCopy (Dest, Source: PChar): PChar;
La fonction StrCopy copie Source dans Dest et renvoie Dest.
StrCopy n'effectue aucune vérification de taille.

function StrLen(Str: PChar): Cardinal;
La fonction StrLen renvoie le nombre de caractères de Str, sans compter le zéro terminal.

▲ **Conversion d'une chaîne de type Pascal à une chaîne à zéro terminal :**

function StrPas (Str: PChar): String;
La fonction StrPas convertit la chaîne à zéro terminal Str en une chaîne de style Pascal.

function StrPCopy (Dest: PChar; const Source: String): PChar;
La fonction StrPCopy copie la chaîne de style Pascal Source dans la chaîne à zéro terminal Dest.
StrPCopy n'effectue aucune vérification de taille. Le tampon de destination doit disposer de suffisamment de place pour au moins Length (Source)+ 1 caractère.

4.7 Gestion de la date

function Date : TDateTime;
La fonction Date renvoie la date en cours.

function DateTimeToStr (DateTime: TDateTime): String;
La fonction DateTimeToStr convertit une variable du type TDateTime en une chaîne. Si le paramètre DateTime ne contient pas de valeur de type date, la date s'affiche sous la forme 00/00/00. Si le paramètre DateTime ne contient pas de valeur de type heure, l'heure s'affiche sous la forme 00:00:00 AM.

On peut modifier le formatage de la chaîne en modifiant certaines constantes typées de date et d'heure.

Exemple :

{ Cet exemple utilise une étiquette et un bouton sur une fiche. Lorsque l'utilisateur clique sur le bouton, la date et l'heure en cours sont affichées comme étant le libellé (propriété Caption) de l'étiquette }

```
procedure TForm1.Button1Click (Sender: TObject);
begin
        Label1.Caption := DateTimeToStr (Now );
end;
```

function DateToStr (Date: TDateTime): String;
La fonction DateToStr convertit une variable du type TDateTime en une chaîne. La conversion utilise le format spécifié par la variable globale ShortDateFormat.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
        Label1.Caption := DateToStr (Date );
end;
```

function Now : TDateTime;
La fonction Now renvoie la date et l'heure en cours, correspondant à Date + Time.

Exemple :

```
procedure TForm1.Button1Click (Sender: TObject);
begin
        Label1.Caption := DateTimeToStr (Now);
end;
{ Affiche la date et l'heure dans un composant label }
```

function StrToDate (const S: string): TDateTime;
La fonction StrToDate convertit une chaîne en un format de date. La date dans la chaîne doit être d'un format valide.

La chaîne doit consister en deux ou trois nombres, séparés par le caractère défini par la variable globale DateSeparator. L'ordre pour le mois, l'année et le jour est déterminé par la variable globale ShortDateFormat. Les combinaisons possibles sont mm/jj/aa, jj/mm/aa, et aa/mm/jj.

Si la chaîne contient seulement deux nombres, elle est interprétée comme une date (mm/jj ou jj/mm) de l'année en cours. Les années comprises entre 0 et 99 sont supposées varier de 1900 à 1999.

Si la chaîne donnée ne contient pas de date valide, une exception EConvertError est provoquée.

function Time: TDateTime;
La fonction Time renvoie l'heure en cours.

function TimeToStr (Heure: TDateTime): String;
La fonction TimeToStr convertit le paramètre Heure, une variable de type TDateTime, en une chaîne.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
        Label1.Caption := TimeToStr (Time );
end;
```

4.8 Routines diverses

function Assigned (var P): Boolean;
La fonction Assigned teste si un pointeur ou une variable procédure vaut nil (n'est pas affecté).
P doit être une référence de variable à un type pointeur ou procédure.

procedure Dec (var X [; N: Longint]);
La procédure Dec soustrait à une variable 1 ou N. Dec (X)est équivalent à X := X - 1 et Dec (X, N) à X := X - N.

procedure Inc (var X [; N: Longint]);
La procédure Inc renvoie la variable X augmenté de 1 ou de N.
Inc (X)correspond à l'instruction X := X + 1 et Inc (X, N) correspond à X := X + N.

function SizeOf (X): Integer;
Renvoie le nombre d'octets occupés par X.
X est soit une référence de variable ou un identificateur de type.

variable FileMode :
La variable FileMode (de type Byte) détermine le code d'accès à passer au DOS lorsque des fichiers typés et non typés (pas les fichiers texte) sont ouverts avec la procédure Reset.

Par défaut, FileMode est à 2. Si on assigne une autre valeur à FileMode, tous les Reset suivants utiliseront ce mode.

Les valeurs de FileMode sont les suivantes :

0	Lecture seule
1	Ecriture seule
2	Lecture/écriture

5 Principes de base de la POO

Delphi met en œuvre conjointement plusieurs concepts récemment généralisés dans le domaine de la programmation. En particulier il utilise des objets et fonctionne selon un mode événementiel.

- Dans un premier temps l'on se contentera d'utiliser des objets prédéfinis pour créer des applications, mais l'on pourra ultérieurement créer ses propres objets ou modifier les objets existants.

5.1 Le concept OBJET

5.1.1 Les principes de la modélisation

- C'est une approche de décomposition naturelle d'un problème à résoudre en se basant sur les objets physiques du monde à modéliser.
- Chaque OBJET représente des comportements et conserve ses propres informations sur ces caractéristiques intrinsèques et son état courant.
- C'est un moyen naturel de regrouper traitements et données.

Dans cette nouvelle approche, on modularise TRAITEMENTS ET DONNEES.

5.1.2 Les concepts Clés

- Classe
- Héritage
- Message
- Objet
- Méthode
- Instance
- Polymorphisme
- Encapsulation

5.1.2.1 Objet

- **Représente une partie de la réalité :**
  - Entité indépendante et autonome

- Identifiable

- **Regroupe données et traitements** (ENCAPSULATION) :
  - Traitements : METHODE : dynamique,
  - Données : PROPRIETES : statique,

- **Accessible uniquement à travers son interface** :
  - Service qu'il sait fournir,
  - Son organisation interne est cachée,
  - Les données composantes FRAGILES sont protégées des ingérences extérieures,

- **Réutilisable** (composant).

5.1.2.2 Encapsulation

- Regroupement cohérent des données et traitement au sein d'une même entité l'OBJET

- Ce concept fondamental permet une **décomposition naturelle du problème à résoudre** basé sur les objets réels du mode à modéliser**.**

- L'objet est défini à l'aide de ses trois composants :
  - **Les données,**
  - **Les traitements,**
  - **L'interface.**

Mécanisme performant de regroupement de l'information et de dissimulation des détails.

5.1.2.3 Classe

- Modèle décrivant les propriétés (données et traitement) d'une collection d'objets.

- Une classe d'objets se décompose de deux parties :
  - Les attributs (ou champs) décrivant les données de l'objet,
  - Les méthodes décrivant le comportement de l'objet :
    - Ce qu'on peut lui demander,
    - Ce qu'il sait faire.

- La classe est un moule servant à la création d'objets.

Tout objet a un modèle : **Sa classe**, regroupement de deux points de vue de la programmation traditionnelle.

5.1.2.4 Instance

C'est un objet particulier :

- Créé à partir d'une classe qui est utilisée comme modèle,
- Sa structure est celle de sa classe.
- La classe définit les caractéristiques communes à toutes les instances.
- L'instance est créée par une méthode standard du système qui initialise ses champs et lui associe un nom interne.
- L'instance contient les informations qui la rendent unique.

INSTANCE et OBJET sont considérés comme des synonymes.

5.1.2.5 Méthode

- Mode de réaction d'un objet (ce qu'il sait faire),
- Programme associé à une classe,
- L'exécution s 'applique à un objet de la classe,
- Les méthodes d'une classe sont communes à tous les objets potentiels de cette classe,
- Une méthode accepte des arguments et renvoie des valeurs,
- Des méthodes peuvent porter le même nom et correspondre à des comportements différents (SURCHARGE).

5.1.2.6 Message

- Le message est le moyen de formuler une requête à un OBJET :
  - Exécuter une méthode particulière,
- L'OBJET réagit au MESSAGE :

- En exécutant la METHODE désignée,
- En retournant les résultats éventuels,

- Le MESSAGE contient :
  - L'identifiant de l'OBJET,
  - Un sélecteur de METHODE,
  - Des arguments éventuels.

- La méthode par MESSAGE offre des niveaux importants de protection :
  - Les ATTRIBUTS ne peuvent pas être altérés,
  - Le MESSAGE est indépendant de la structure interne de l'OBJET.

5.1.2.7 Héritage

- Mécanisme fondamental de l'orientation OBJET qui simplifie la définition de nouvelles CLASSES en fonction de CLASSES existantes et facilite la réutilisation,

- La hiérarchie de CLASSE reflète la connaissance humaine qui est structurée de la même manière. Elle repose sur les concepts génériques que l'on affine pour décrire des cas de plus en plus particuliers.

- Eviter la redéfinition des METHODES et des ATTRIBUTS communs dans la classe.

- On ne précise que les différences avec la CLASSE de niveau supérieur (SUPER-CLASSE).

- Une SOUS-CLASSE hérite des ATTRIBUTS et METHODES de sa SUPER-CLASSE.

- La redéfinition des propriétés (ATTRIBUTS et METHODES) permet le traitement des cas particuliers sans impact sur l'existant (SURCHARGE).

5.1.2.8 Polymorphisme

- Le polymorphisme permet de dissimuler des particularités derrière le même interface.

- Des méthodes appartenant à des CLASSES différentes :
  - Peuvent porter le même nom et correspondre à des comportements différents.

- Le même MESSAGE peut être adressé à des OBJETS différents qui l'interpréteront à leur manière.

- La programmation par MESSAGE et le mécanisme de polymorphisme confèrent une grande autonomie à l'OBJET.

5.1.2.9 Constructeur d'objet

- Méthodes standards d'INITIALISATION/RESERVATION applicables à toutes les CLASSES par un MESSAGE prédéfini du système,
- Initialise les attributs de l'objet et lui donne un nom interne (adresse ou référence),
- La méthode standard de création peut être surchargée.

5.1.2.10 Destructeur d'objet

- Appelé quand l'objet cesse d'exister (automatique ou programmé suivant le langage),
- Restitue la place allouée au moment de l'instanciation de l'OBJET (parfois automatique).

5.1.2.11 Concepts majeurs de la programmation objet

- La modélisation à l'aide des classes
- L'encapsulation fusion des données et des traitements
- L'activation à l'aide d'envoi de messages concernant une méthode
- La construction par affinage à l'aide de la notion d'héritage
- Le polymorphisme qui consiste à attribuer le même nom à des actions (méthodes) exécutées d'une manière qui est propre à chaque objet

Ces principes simples permettent une très grande puissance d'expression

5.1.3 Plus simplement …

Un objet est la représentation informatique d'un objet dans son sens courant : il forme un tout, avec des **caractéristiques propres et un comportement**. Il est **MANIPULABLE**.

Un objet, au sens informatique, est une entité contenant à la fois **des données** modifiables et des **procédures** qui agissent sur ces données.

En programmation orientée objet, les données de l'objet - appelées propriétés- ne sont accessibles qu'au moyen des procédures contenues dans l'objet. Celles-ci constituent donc **l'interface de l'objet** avec l'extérieur. On les appelle **des méthodes**.

Il n'est pas besoin de connaître les mécanismes internes de l'objet : seule la connaissance des méthodes suffit pour le manipuler.

L'objet peut donc être considéré comme une boite noire, réalisant certains comportements prédéfinis. Ces derniers peuvent être activés à la réception d'un message, que l'on peut envoyer de l'extérieur.

5.2 Principe de fonctionnement de Delphi

5.2.1 Les composants

L'environnement Delphi est fourni avec un ensemble de composants prédéfinis qui sont des objets informatiques utilisables tels quels pour créer les applications.
Comme tout objet, un composant comprend :

- **Des données :**
  - Elles sont appelées **propriétés** et fournissent une information descriptive de l'état du composant.
  - Des propriétés ne sont accessibles qu'en lecture et ne peuvent donc pas être modifiées.
  - Des propriétés sont accessibles pendant la phase de conception du programme (on peut donc les initialiser) et permettent généralement de personnaliser l'aspect du composant (couleur, position, dimension, etc...).
  - D'autres propriétés ne sont accessibles qu'à l'exécution (on ne peut y accéder ou les modifier que par programmation).
- **Des méthodes :**
  - Chaque composant contient un nombre varié de méthodes accessibles au programmeur. Ces méthodes permettent d'agir sur le comportement du composant.
  - Comme les composants sont destinés à fonctionner dans un environnement "événementiel", ils sont conçus pour être capables de réagir à un nombre variable d'événements.
  - Ces événements peuvent être des "clicks" souris, des actions sur le clavier ou d'autres types. Un composant ne réagit qu'aux événements pour lequel il a été conçu.

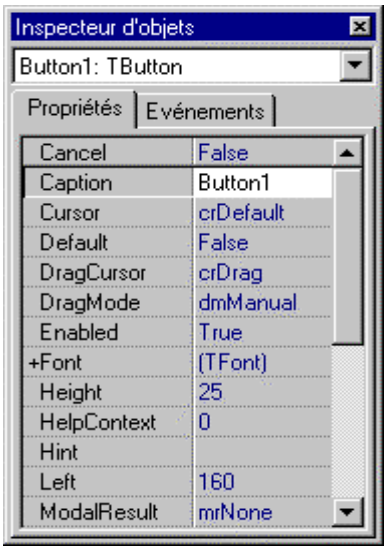
5.2.2 L'inspecteur d'objet

▲ **Présentation de l'inspecteur d'objet :**
L'inspecteur d'objet est l'utilitaire qui est le plus utilisé lorsque l'on veut créer des applications avec Delphi.

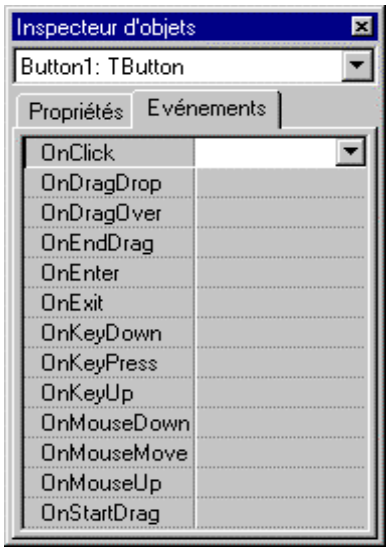
Lorsque l'on dépose un composant dans la fenêtre de conception, l'inspecteur d'objet affiche automatiquement, grâce à deux onglets :

- Une grande partie de ses propriétés (celles qui sont accessibles lors de la phase de conception ; pour connaître les autres propriétés il faut faire appel à l'aide en ligne).
- Les événements auxquels réagit le composant.

La zone de texte située en haut indique le nom de l'objet sélectionné et sa classe d'appartenance. On peut ainsi accéder à tous les objets de la feuille en faisant dérouler la zone (en appuyant sur la flèche située à droite).



Onglet "propriétés"



Onglet "événements"

L'inspecteur d'objet risque d'être caché par le nombre de fenêtres ouvertes. On peut le placer au-dessus des fenêtres en appuyant sur la touche F11 mais aussi le configurer de manière à ce qu'il soit "toujours au-dessus" en validant l'option correspondante dans le "pop-up menu" apparaissant suite à un click droit de la souris sur l'inspecteur.

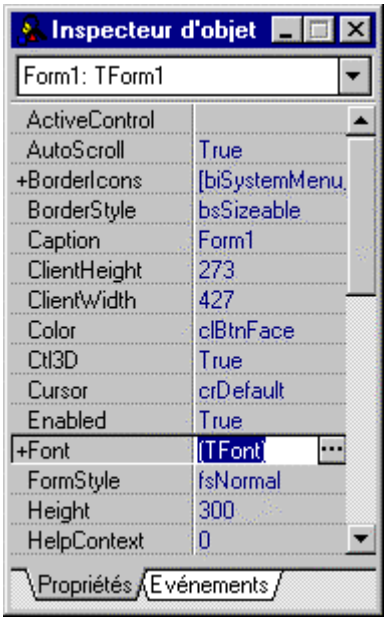
▲ **Accès aux propriétés par l'inspecteur d'objet :**

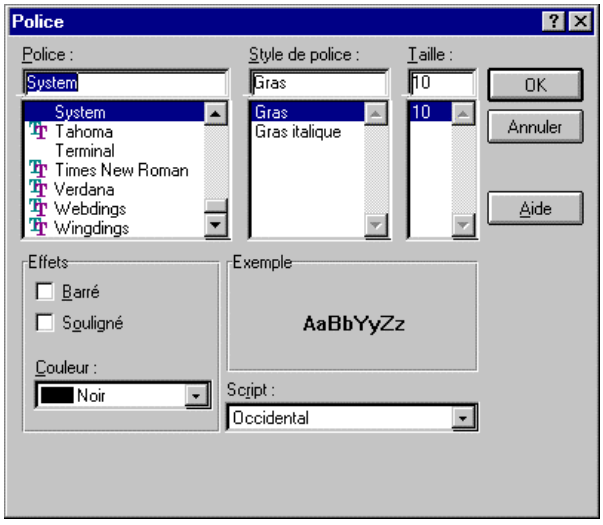
La colonne de gauche de l'inspecteur d'objet liste les identifiants des propriétés publiques (= accessibles à la conception) du composant.

La colonne de droite affiche la valeur courante des propriétés.

Le fait de modifier une propriété a un effet immédiat sur l'objet concerné (taille, positionnement, couleur, etc, ...). De même le fait de modifier à la souris certaines caractéristiques du composant modifie immédiatement les valeurs des propriétés concernées.

Si une entrée de l'onglet "propriétés" comporte un signe '+' c'est qu'on peut accéder à des sous-propriétés en cliquant sur l'entrée concernée. Quand la liste se déploie, le ' + ' est changé en ' - ' .
Dans la zone des valeurs, une valeur suivie par ' ... ' indique que l'on peut accéder à une fenêtre de dialogue en cliquant dessus.





Dans la recopie d'écran ci-dessus le fait d'avoir cliqué sur l'entrée 'Font ' a déroulé les options 'Color', 'Height', etc… Mais on peut aussi modifier directement ces options en utilisant la boite de dialogue "Polices".

Le fait de cliquer sur des valeurs de données fait apparaître une flèche descendante indiquant qu'une liste de valeurs est disponible. En déroulant la liste, on peut sélectionner la valeur souhaitée.

Certaines propriétés ne peuvent prendre que deux valeurs (en général True et False). Le seul fait de double-cliquer sur la valeur en cours la modifie en son inverse.

Lorsque l'on a cliqué dans une valeur de propriété et que l'on change ensuite de composant, c'est la propriété identique du nouveau composant qui est activée, sans que l'on ait besoin de la rechercher.

▲ **Accès aux événements par l'inspecteur d'objet :**
Lorsque l'on affiche l'onglet 'événements' de l'inspecteur d'objet, les entrées de droite sont toutes vierges.

Lorsque le programmeur choisit un événement auquel le composant réagit (en double-cliquant sur l'entrée concernée de l'inspecteur d'objet), une fenêtre d'édition apparaît proposant un nom par défaut à la fonction événement associée (on l'appelle aussi 'gestionnaire d'événement') ainsi qu'un squelette de procédure.

Le programmeur n'a plus qu'à remplir, par les instructions appropriées, le squelette pour créer le gestionnaire d'événement associé.

5.2.3 Accès aux méthodes

Certaines méthodes permettent d'accéder aux données internes du composant et ainsi d'en modifier le comportement tandis que d'autres se servent des caractéristiques du composant pour réaliser diverses fonctionnalités.

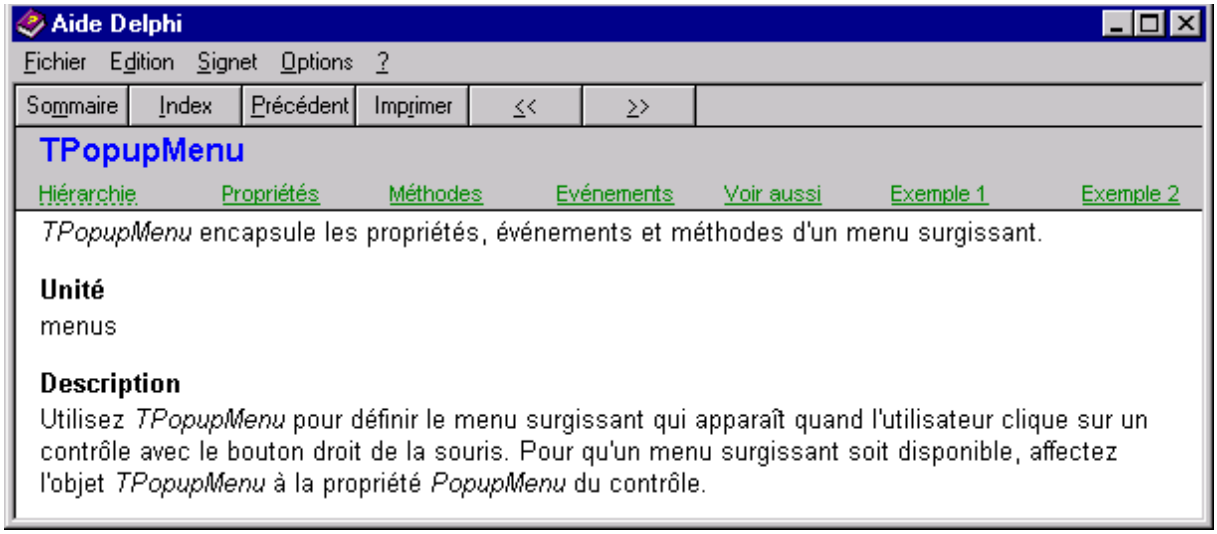
Pour savoir quelles méthodes utiliser, et comment les utiliser, il faut faire appel à l'aide en ligne de Delphi (en appuyant sur la touche F1 après avoir sélectionné le composant désiré).

5.2.4 L'aide en ligne

L'aide en ligne propose une description du composant dans le bandeau située en haut de la fenêtre. Elle permet d'accéder à trois ou quatre rubriques distinctes :

- La rubrique **"Hiérarchie"** donne l'arborescence des objets desquels le composant est dérivé.
- La rubrique **"Propriétés"** décrit le rôle de toutes les propriétés du composant.
- La rubrique **"Méthode"** décrit toutes les méthodes du composant.
- La rubrique **"Evénements"** récapitule les événements pouvant être associés au composant.
- La rubrique **"Voir aussi"** donne la liste des rubriques de l'aide connexes au sujet courant.

- Des rubriques **"Exemple"** illustre l'utilisation du composant.



Il est possible d'accéder directement à une propriété en l'ayant au préalable sélectionnée dans l'inspecteur d'objet avant d'appuyer sur F1.

Rôle de l'aide en ligne :
Il faut considérer les aides en ligne de Delphi comme des utilitaires à part entière de l'environnement de développement.

Les aides de Delphi proposent :

- La description exhaustive de la syntaxe et de la grammaire du Pascal et de ses extensions.
- La totalité de la bibliothèque de fonctions du Pascal et ses extensions Object Pascal.
- La description complète du SDK Windows.
- La description des spécificités de Delphi et de ses composants.

il n'est pas possible de connaître la totalité des fonctions, composants et autres "objets" entrant dans la réalisation d'une application Delphi. Il faut donc apprendre à utiliser l'aide en ligne en la considérant comme un outil à part entière.

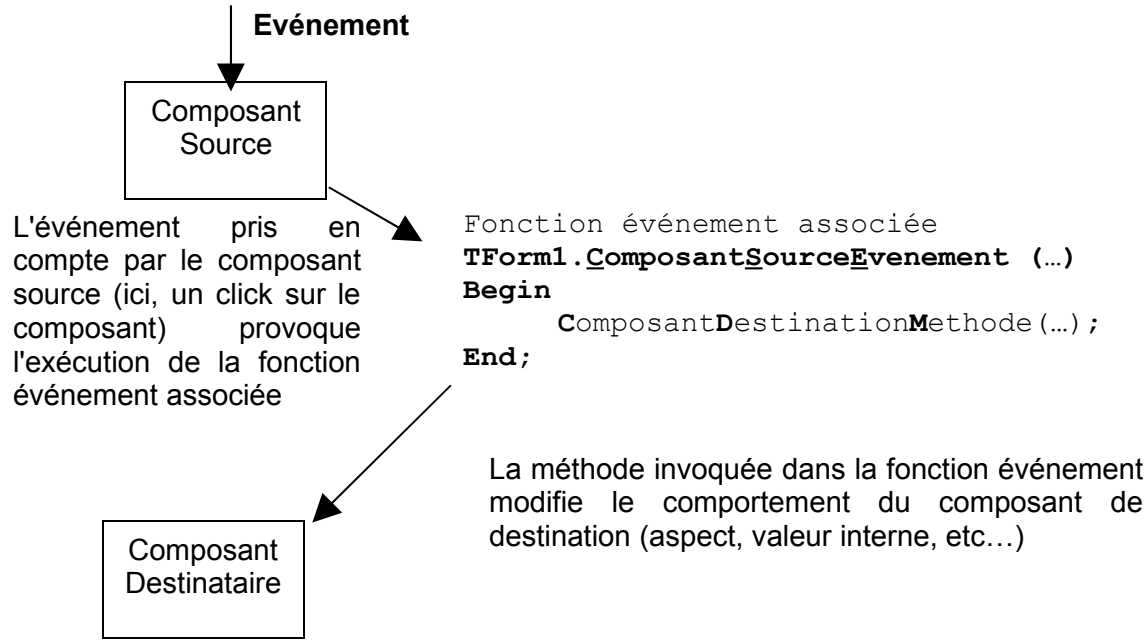
5.2.5 Programmation événementielle

Les méthodes des différents objets composent leur interface vis à vis du monde extérieur. Ainsi, les méthodes sont invoquées par d'autres composants.

L'activation d'une méthode à partir d'un autre composant correspond à l'envoi et à la réception d'un "MESSAGE" (cf. concept OBJET) :

- un objet source envoie, sous la forme de l'activation d'une méthode adéquate, un message à un objet destination pour que ce dernier modifie son comportement.

On a donc le schéma suivant :



Delphi, comme le langage C++ d'ailleurs, n'implémente pas de manière rigoureuse les différents mécanismes de base de la P.O.O.. En particulier il permet d'accéder à certaines propriétés des composants sans utiliser une méthode du composant. De ce fait un "message" pourra consister en l'activation d'une méthode du composant cible ou, tout simplement, la modification d'une de ses propriétés.

On peut comprendre ce fonctionnement sur le cas suivant :

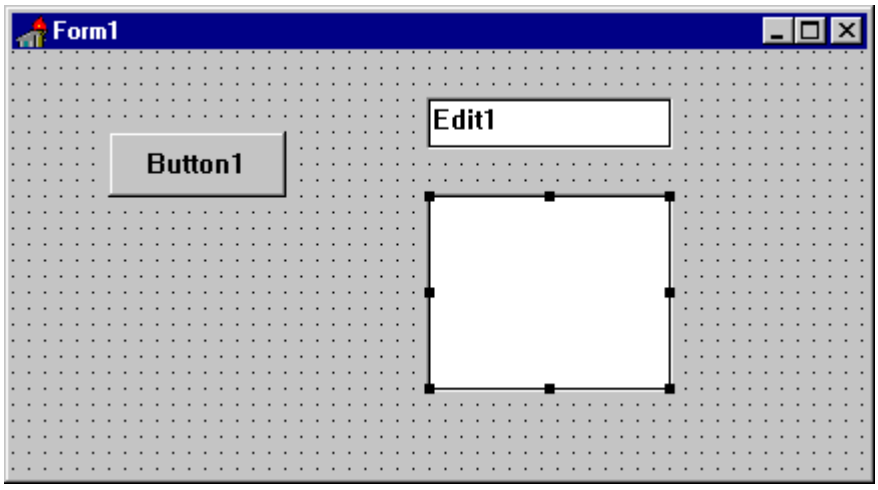
Un composant bouton (Button1) réagit à l'événement "click" souris. Lorsque cet événement survient il exécute deux actions :

- Il modifie la propriété "Text" du composant Edit1 (qui est une zone de saisie et d'affichage) en lui donnant la valeur "bonjour " ;
- Il invoque la méthode Add () du composant ListBox1 (qui est une liste de chaînes de caractères pouvant mettre en œuvre un ascenseur) afin d'ajouter un élément à la liste.

Le code du gestionnaire d'événement associé est alors le suivant :

```
procedure TForm1.ButtonClick1 (Sender : Tobject);
begin
     Edit1.Text := 'Bonjour' ;
     ListBox1.Items.Add ('Nouvel élément');
end ;
```

La fenêtre (en mode conception) se présente donc comme suit :



A l'exécution, une fois le bouton activé par un click souris, la fonction événement associée modifie les deux composants :

▲ **Notation événementielle "pointée" :**
Delphi utilise la notation événementielle "pointée. Cette notation permet de définir, pour un objet considéré, la donnée à utiliser ou la méthode à invoquer.

La notation pointée se lit de droite à gauche :

```
ListBox1.Items.Add (Edit1.Text );
```

Se lit :

Add (...) :	Ajouter. Ici il s'agit d'ajouter le contenu de la zone de saisie Edit1.
Items :	Elément de la liste
ListBox1 :	Boite de liste

Soit : "Ajouter un nouvel élément, dont la valeur est celle contenue dans la zone de saisie Edit1, dans la liste ListBox1".

5.2.6 Génération de code

Au fur et à mesure que des composants sont posés sur la feuille de conception de l'interface, les lignes de codes correspondantes sont automatiquement ajoutées par Delphi. On peut distinguer les lignes correspondant :

- A la déclaration / création du composant (avec réservation de la zone mémoire correspondante);
- Au squelette du gestionnaire d'événement associé à un composant;
- A l'en-tête du gestionnaire d'événement , dans la section 'déclaration' de l'unité.

Delphi se charge de tout :

- Si le nom d'un composant est ensuite modifié (via l'inspecteur d'objet), cette modification est répercutée dans le code ;
- Si un composant est effacé de la feuille de conception, ses références sont effacées ;
- Si un gestionnaire d'événement reste vide (il n'y a que le squelette), il est effacé lors de la première opération de sauvegarde.

▪ Cependant, Delphi ne modifie ou ne supprime que les références qu'il a lui même créé automatiquement. Si un composant est référencé "manuellement" (c'est à dire par le programmeur) dans du code cette référence n'est pas modifiée ou supprimée le cas échéant. C'est au programmeur de s'en charger.

De même il ne faut pas tenter de modifier "à la main" ce qui a été créé automatiquement par Delphi :

- Pour modifier un nom de composant il faut passer par l'inspecteur d'objet ;
- Pour supprimer un composant il faut l'effacer de la feuille de conception ;
- Pour supprimer un gestionnaire d'événement il faut simplement effacer le code tout en conservant son squelette (celui sera ensuite effacé automatiquement).

Le non-respect de ces règles peut provoquer des incohérences lors des compilations ultérieures dont il est souvent malaisé de s'en sortir.

5.3 Notion de projet

5.3.1 Généralités

Pour assurer la création d'une application, Delphi met en œuvre un gestionnaire de projet qui gère l'ensemble des éléments constitutifs de l'application.

Une application est constituée, en règle générale, d'un nombre variable de fenêtres qui sont toutes dépendantes d'une fenêtre principale.

Chaque fenêtre est un composant spécifique, appelé 'Form', qui possède un nom (par défaut 'Formx'). Contrairement aux autres composants, celui-ci ne fait pas partie de la palette de composants. Il faut passer par une rubrique du menu 'Fichier' ou par une icône de la 'speed-bar' pour disposer d'un composant Form.

5.3.2 Organisation d'un projet

Un projet est constitué des éléments suivants :
Fichiers sources :

- **BPG :** groupe de projets.
- **DPR :** Un fichier fédérateur d'extension .DPR, de format texte.
- **PAS** et **DFM :** Un ensemble de fichiers sources .PAS et les fichiers .DFM associés (un couple de fichiers par fenêtre).
- **DPK :** source de paquets.

Fichiers créés lors de la compilation :

- **DCU** ou **OBJ** : un fichier par unité,
- **EXE ou DLL** : un fichier par projet,
- **DCP** : fichier paquet comilé,
- **BPL** : paquet d'exécution.

Fichiers de configuration :

- **CFG :** Fichier de configuration du projet. Stocke les paramètres de configuration du projet. Il porte le même nom que le fichier projet, mais avec l'extension .CFG. Le compilateur cherche dcc32.cfg dans le répertoire contenant l'exécutable du compilateur, puis dans le répertoire en cours, et enfin il cherche nomduprojet.cfg dans le répertoire du projet.
- **DOF :** Fichier des options de Delphi. Contient les paramètres en cours des options de projet, comme les options du compilateur et du lieur, les répertoires, les directives conditionnelles et les paramètres de la ligne de commande.
- **DSK :** Paramètres du bureau. Enregistre l'état du bureau en cours, comme les fenêtres ouvertes et leur position.
- **RES :** Contient les ressources info de version (si nécessaire) et l'icône principale de l'application. Ce fichier peut aussi contenir d'autres ressources utilisées dans l'application mais celles-ci sont préservées.
- **DCI :** Contient les modifications de l'audit de code effectuées dans l'EDI.
- **DCT :** Contient les modifications de modèles de composants effectuées dans l'EDI.
- **DMT :** Contient les modifications de modèles de menus effectuées dans l'EDI.
- **DRO :** Contient les modifications lorsque l'on fait des ajouts au référentiel. Peut être supprimé, mais vos ajouts au référentiel sont perdus.
- **TDS :** Contient la table des symboles de débogage externes.
- **TODO :** Fichier de la liste des choses à faire. Contient la liste en cours pour le projet. Porte le même nom que le fichier projet, avec l'extension .TODO.

En plus de ces fichiers, le gestionnaire gère des fichiers binaires d'extension .DCU qui sont les modules objets résultants de la compilation des fichiers sources.

A l'ouverture de Delphi, un nouveau projet est ouvert par défaut (Project1). Il est possible d'utiliser ce projet pour construire une nouvelle application ou alors charger un nouveau projet via le menu ou l'icône adéquat.

A chaque modification d'un fichier, Delphi sauvegarde la version précédente (extension .~pa pour un fichier .pas, extension .~df pour un fichier .dfm, extension .~dp pour un fichier .dpr).

Comme tout gestionnaire de projet, la compilation des différents fichiers sources se fait de manière incrémentale: seules, les sources modifiés depuis la dernière compilation.

Le gestionnaire de projet permet de réaliser des exécutables mais aussi des DLL.

Compte tenu du nombre important de fichiers qui peuvent être générés, il est souhaitable de créer au préalable un répertoire particulier pour chaque projet important. En indiquant ce répertoire dans l'onglet ' Répertoires / Conditions' du menu 'Projet | Options ' de Delphi, les différents fichiers générés automatiquement le seront dans le répertoire souhaité et seront facilement identifiables.

5.3.3 Source d'un projet

```
program Project1; <nom prog>

uses <nom unités>
  Forms,
  Unit1 in 'Unit1.pas' {Form1},
  Unit2 in 'Unit2.pas' {Form2};

const
                <nom constante> = <valeur constante>
type
                <identificateur de type> = <définition de type>
var
                <nom variable> : <type variable>

<définition des procédures et des fonctions>

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.Run;
end.
```

Au niveau de la programmation chaque fenêtre est matérialisée par :

- Un fichier d'extension « .PAS » contenant les différentes instructions propres à chaque fenêtre (les différents gestionnaires d'événements associés aux composants et, éventuellement quelques routines internes).

- Un fichier .PAS correspond à une 'unité'.

- Un fichier d'extension .DFM (Delphi ForM) contenant, sous forme binaire, la représentation graphique de l'interface.

5.3.4 Organisation d'une unité

```
unit <nom de l'unité>

interface
        uses
                <nom des unités utilisées>
        const
                <nom constante> = <valeur constante>
        type
                <identificateur de type> = <définition de type>
                        private  { Déclarations privées }
                        public   { Déclarations publiques }
                end;

        var
                <nom variable> : <type variable>
                <déclaration des procédures et des fonctions>

implementation
        uses
                <nom des unités utilisées>
{$R *.DFM}
        const
                    <nom constante> = <valeur constante>
var
                <nom variable> : <type variable>

<définition des procédures et fonctions>
end.
```

5.3.4.1 Clause « uses »

- Elle détermine les déclarations provenant d'autres unités qui sont visibles dans l'unité en cours.
- C'est ce mécanisme qui permet à des unités d'utiliser des procédures, des fonctions, constantes, types et variables appartenant à d'autres unités.
- Si le nom d'une unité apparaît dans la clause « uses », toutes les déclarations de la section « interface » de l'unité correspondante pourront être utilisés dans l'unité ou le programme en cours.
- Les références circulaires :
    - Provoquent une erreur lors de la compilation,
    - Se produit quand deux unités se référencent mutuellement,
    - Deux solutions :

- ♦ Référencer les deux unités dans la clause « uses » de la section « implémentation » des unités respectives,
- ♦ Référencer une unité dans la section « interface » et dans l'autre unité dans la section « implementation ».

- Pour référencer une fiche dans une autre, utiliser la commande du menu **Fichier | Utiliser unité.**

5.3.4.2 Section « interface »

- Elle détermine ce qui peut être vu de cette unité par d'autres.
- On y trouve des déclarations de constantes, de types, de variables, de fonctions et de procédures (les définitions des fonctions et procédures sont dans la section « implementation »).
- Il faut limiter autant que possible les éléments placés dans cette section. Tout ce qui y figure correspond à des déclarations publiques (prendre garde de ne pas y placer des éléments qui doivent être inaccessibles).

5.3.4.3 Section « implementation »

- On peut y trouver des constantes, des types, des variables et d'autres procédures ou fonctions,
- Ces éléments de programme ne seront pas accessibles par d'autres un autre programme ou une autre unité.
- On y trouve le code de toutes les procédures et fonctions déclarées dans la section « interface ».

5.3.4.4 Section « initialization »

- Le code qu'elle contient est exécuté lors de la première utilisation de l'unité.
- C'est là que sont, en général, initialisées les structures de données.

5.3.4.5 Section « finalization »

- Rôle inverse de la section intialization.
- Le code qu'elle contient est exécuté quand le programme principal s'est achevé (exemple : libération de ressources).

5.3.5 Organisation d'une forme

Un fichier .DFM contient la description complète de l'interface graphique d'une feuille (c'est à dire la description de l'ensemble des composants de la feuille). Ce fichier a un format particulier qui le rend illisible tel quel. Mais l'éditeur de Delphi peut charger un fichier .DFM sous une forme lisible.

De même si l'on copie un composant dans le presse-papiers puis qu'on colle le résultat dans un éditeur de texte, comme le bloc-notes, c'est sa description au format texte qui apparaît.
Contenu du fichier .DFM d'un projet constitué d'une feuille et d'un bouton (rendu lisible par importation dans l'éditeur de Delphi):

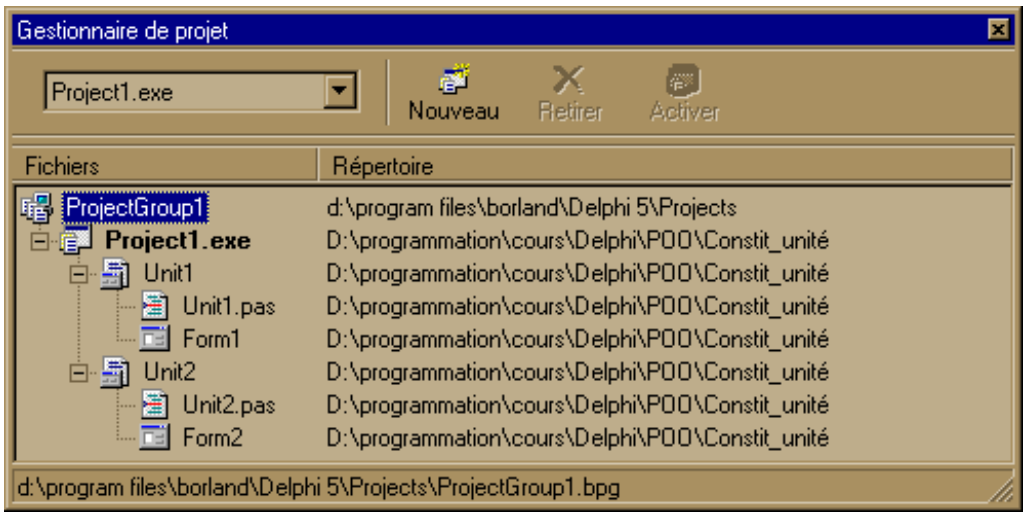
```
object Form1: TForm1
        Left = 200
        Top = 97
        Width = 435
        Height = 300
        Caption = 'Form1'
        Font.Color = clWindowText
        Font.Height = -13
```

```
                    Font.Name = 'System'
                    Font.Style = []
                    PixelsPerInch = 96
                    TextHeight = 16
                    object Button1: TButton
                        Left = 114
                        Top = 66
                        Width = 89
                        Height = 33
                        Caption = 'Button1'
                        TabOrder = 0
                    end;
            end;
```

5.3.6 Le gestionnaire de projet

Le gestionnaire de projet est accessible à partir du menu 'Voir | Gestionnaire de projet'. Il permet d'ajouter ou de retrancher des unités au projet.

La fenêtre du gestionnaire de projet



Les unités incorporées au projet peuvent provenir d'autres projets.

Lorsqu'un nouveau projet a été créé (à partir du PROJECT1 proposé par défaut), la première unité reçoit le nom de UNIT1.PAS et le nom de la fenêtre FORM1.

Lorsque l'on souhaite sauvegarder le projet, il faut donner un nom particulier à cette fenêtre ainsi qu'au fichier de projet, d'extension .DPR, et au fichier source, d'extension .PAS. Ces noms ne doivent pas être identiques (sinon ils seront refusés lors de la sauvegarde).

Il est conseillé d'utiliser des méthodes pour catégoriser les différents composants d'un projet. On peut par exemple conserver les initiales d'un composant pour pouvoir le retrouver plus facilement (FTimbres = Form appelée Timbres), Utimbres pour l'unité et préfixer le nom d'un projet par "P" pour donner "PTimbres".

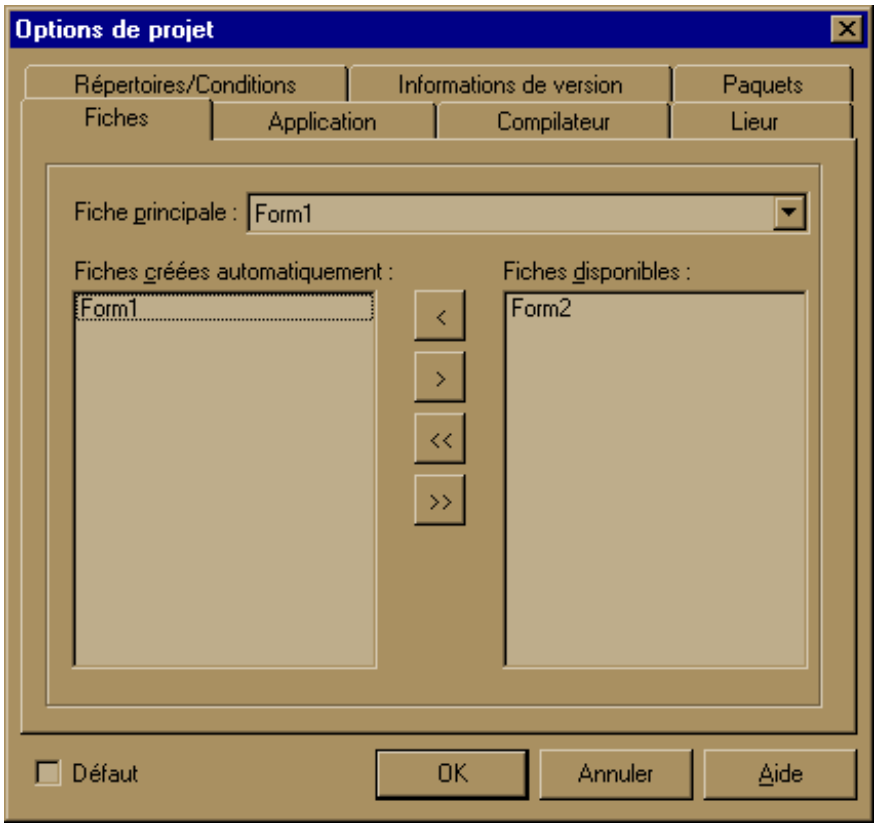
5.3.7 Options du projet

L'icône "options" du gestionnaire de projet ouvre une boîte de dialogue spécifique donnant accès à plusieurs spécificités propres au projet, grâce à un ensemble d'onglets.

L'onglet "fiches" permet de déterminer quelle est la fiche principale.

Plus important encore, il permet de déterminer si les fiches sont "auto-créées" au démarrage de l'application (c'est l'option par défaut) ou si elles ne sont créées qu'en cas de besoin.

La deuxième option est plus satisfaisante car, outre le fait qu'elle permet d'économiser la mémoire utilisée, il se peut que l'auto-création fasse "planter l'application" si une donnée nécessaire à la création d'une fenêtre n'est initialisée que pendant l'exécution de l'application. Il faut donc penser à modifier (pratiquement systématiquement) l'option par défaut.



Il est souhaitable que seule la fiche principale soit "auto-créée". Toutes les autres sont disponibles et ne seront créées qu'au besoin.

L'onglet "application" permet de donner un nom à l'application (qui peut donc être différent que celui du projet) et éventuellement indiquer quelle icône utiliser. C'est ce nom qui apparaîtra sous l'icône. Il permet aussi de désigner le fichier aide éventuellement associé à l'application.

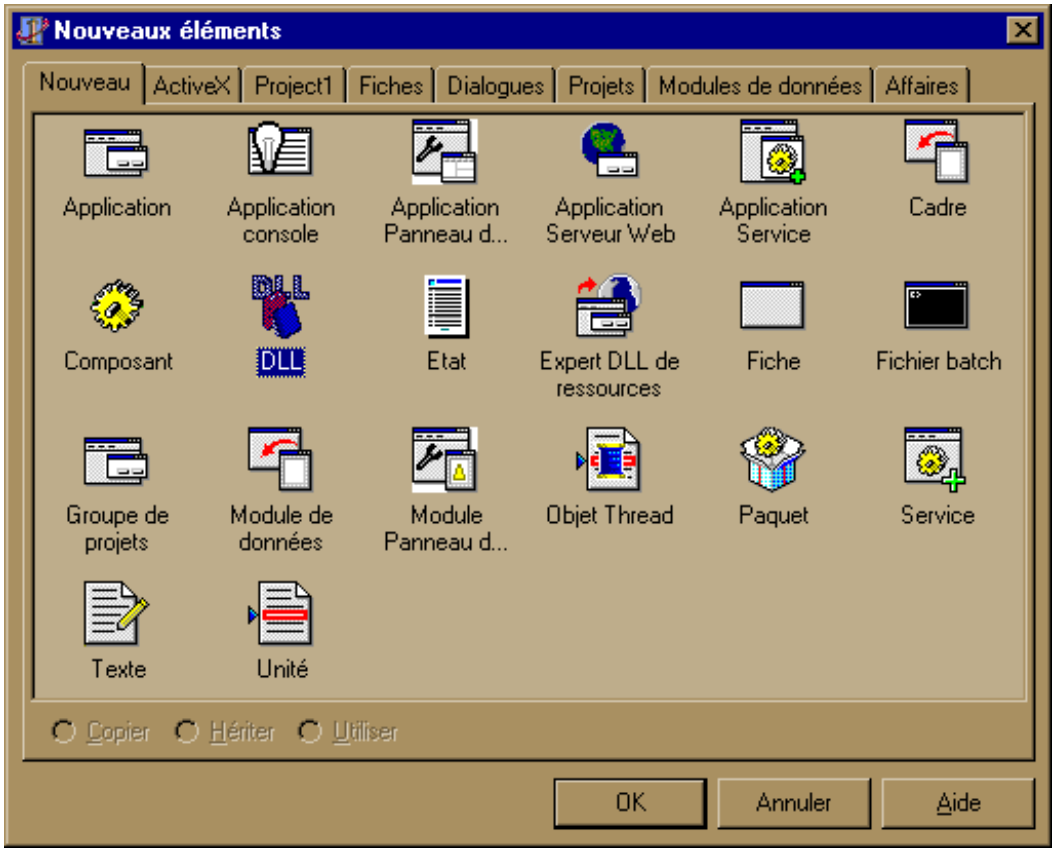
L'onglet "compilateur" permet de choisir la plupart des directives de compilations propres au Pascal. En particulier cet onglet permet de valider l'option "syntaxe étendue" (il est difficile de s'en passer en Delphi).

L'onglet "Répertoires / Conditions" permet :

- De définir le répertoire où seront générés l'exécutable ou la DLL.
- De savoir où sont stockés les différents fichiers sources et les bibliothèques ;
- De définir certaines directives de compilation conditionnelles.

5.3.8 La galerie de projets

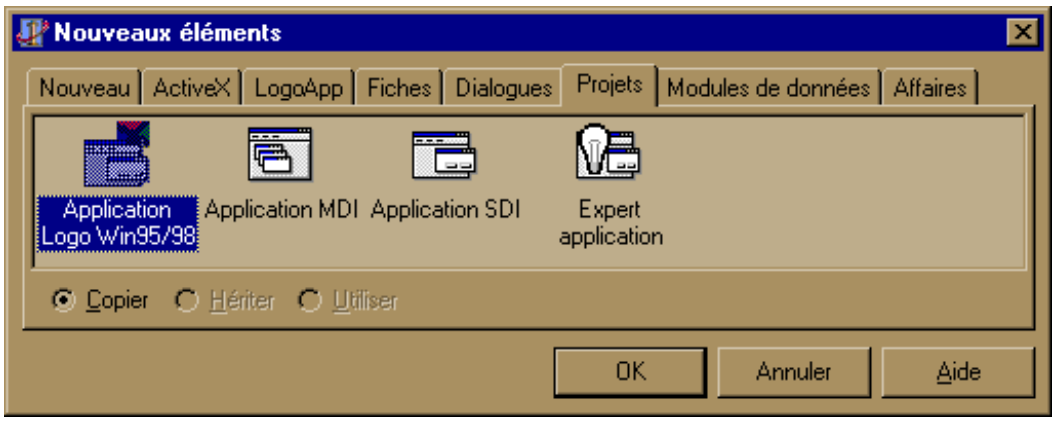
Delphi dispose d'une galerie (Gallery) contenant divers modèles de projets. Ces modèles permettent de démarrer un projet avec un "squelette" déjà réalisé.



Le projet par défaut est le modèle "vierge" mais il est possible de sélectionner les modèles suivants :

Application	Crée un nouveau projet contenant une fiche, une unité et un fichier .DPR.
Fichier batch	Crée un nouveau projet fichier batch portant l'extension .BAT et vous permettant de spécifier un lot de commandes.
Composant	Crée un nouveau composant à l'aide de l'expert composant.
Expert console	Crée un nouveau projet application console.
Application Panneau de configuration	Crée une nouvelle applet pour le Panneau de configuration de Windows.
Module Panneau de configuration	Crée un nouveau module pour une application Panneau de configuration.
Module données	Crée un nouveau module de données qui peut être utilisés comme référentiel pour les composants non visuels et les règles de

	gestion.
DLL	Crée un nouveau projet DLL.
Fiche	Crée une fiche vierge et l'ajoute au projet en cours.
Cadre	Crée un nouveau cadre.
Paquet	Crée un nouveau paquet.
Groupe de projets	Crée un nouveau groupe de projets afin de réunir des projets.
Etat	Crée un état rapide qui vous aide à concevoir visuellement des états pour vos applications de bases de données.
Expert DLL ressource	Démarre un expert qui vous aide à générer une DLL ressource contenant les versions localisées des fiches de votre application.
Service	Ajoute un nouveau service à une application service NT existante. N'ajoutez pas services à une application qui n'est pas une application service. Bien qu'un objet TService puisse être ajouté, l'application ne générerait pas les événements requis ou effectuerait les appels Windows appropriés à la place du service.
Application Service	Crée une nouvelle application service NT.
Texte	Crée un nouveau fichier texte ASCII.
Objet Thread	Crée un nouvel objet thread.
Unité	Crée une nouvelle unité et l'ajoute au projet en cours.
Application serveur web	Crée une nouvelle application serveur web (DLL ou EXE).



Application MDI (Multiple Document Interface): Ce modèle propose un squelette permettant de créer une application MDI. C'est à dire une application composée d'une feuille mère pouvant contenir des fenêtres filles.

Une application MDI permet de visualiser simultanément plusieurs documents (alors que dans une application normale il faut fermer un document pour pouvoir en ouvrir un autre).

Application SDI (Single Document Interface): Une telle application possède une fenêtre principale (comprenant des menus) et permet de créer des fenêtres associées mais "indépendantes" (ces fenêtres ne comportent pas de menus en dehors du bouton "système" et des boutons de redimensionnement).

Les différents onglets permettent de choisir le type d'application souhaité.

5.3.9 Modèles de fiches

De même qu'il est possible de choisir un modèle de projet, il est possible - une fois un projet entamé - de choisir des modèles de fiches "prêtes à l'emploi" que l'on peut donc incorporer au projet.

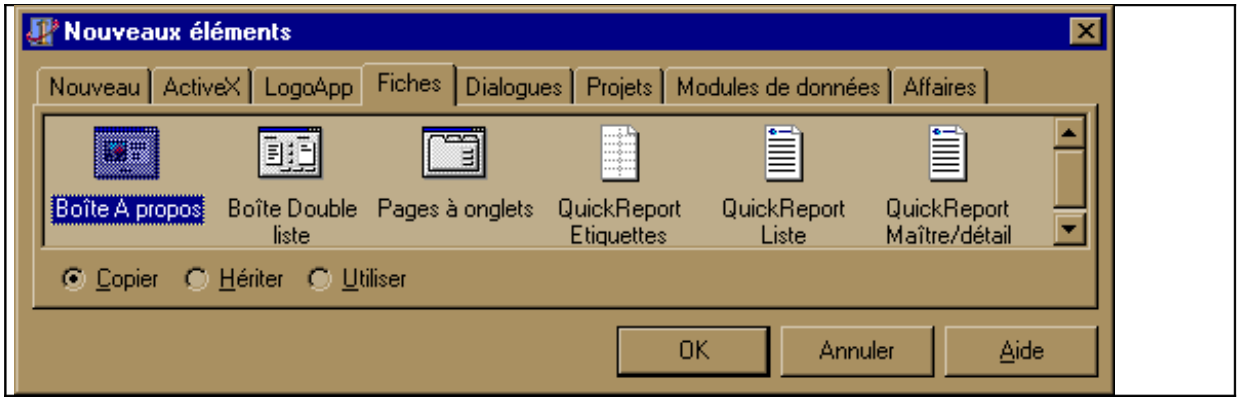
La galerie de fiches est proposée lorsque l'on choisit le menu 'Fichier | Nouveau | Fiches'.

Cette galerie offre de nombreuses possibilités. En particulier, elle donne accès à une boîte de dialogue permettant d'entrer un mot de passe et à certaines boîtes de dialogue relativement complexes.

Comme dans le cas d'un projet, un deuxième onglet permet d'accéder à des experts permettant de concevoir, pas à pas, certaines fiches spécifiques (en particulier, une fiche orientée "base de données").

Il est possible de rajouter des modèles de projets ou de fiches personnelles dans les différentes galeries en utilisant le menu Options | Galerie.

La galerie de fiches prêtes à l'emploi



5.4 Exécution d'une application

5.4.1 Taille de l'exécutable

Lorsque l'on compile une application Delphi en vue de son exécution, la taille de l'exécutable généré peut surprendre.

Il faut comprendre qu'une application Delphi, même "vide" c'est à dire ne comportant qu'une fiche sans aucun composant, met en œuvre en interne tous les mécanismes permettant :

- Le dialogue avec Windows (gestion des événements);
- Le support du mode graphique, et éventuellement du mode 3D ;
- La gestion de la souris ;
- La gestion des exceptions ;
- La gestion de la fenêtre avec son menu système, ses possibilité de redimensionnement, et autres caractéristiques.

De fait, le moindre exécutable généré par Delphi dépasse les 280 Ko.

Il faut prendre en compte le fait que l'ajout de composants dans une feuille se traduira par une augmentation de la taille de l'exécutable : lorsqu'un composant est le premier d'un type donné, l'augmentation peut atteindre quelques kilo-octets, ensuite les autres occurrences du même type de composant ne font gonfler l'exécutable que de quelques dizaines d'octets.

Il est toujours possible de réduire la taille de l'exécutable en éliminant, lors de la dernière compilation, les différentes informations de débogage.

Il est aussi possible de sélectionner l'option "optimisation" mais il faut pour cela disposer de ressources matérielles conséquentes. Néanmoins, l'optimisation est généralement très efficace.

Lorsque l'application met en œuvre un moteur de base de données, la taille de l'exécutable est encore plus importante mais il faut prendre en considération l'importance des fonctions réalisées par ce moteur.

5.4.2 Mise au point d'un programme

Delphi est livré avec un utilitaire permettant la mise au point des programmes. Comme tous les programmes de ce type, le débogueur intégré (accessible via le menu 'Exécuter ') permet :

- Le fonctionnement pas à pas (et pas à pas approfondi) en utilisant les raccourcis clavier F7 et F8,
- La mise en place de points d'arrêts (breakpoints).
- Le suivi des variables grâce à une fenêtre de définition des variables à suivre et une fenêtre d'affichage des valeurs qu'elles prennent en cours d'exécution pas à pas.
- L'évaluation de variables.

Il est possible d'afficher les variables suivies selon plusieurs formats.
L'affichage en mode "pas à pas" est parfois aléatoire (les passages incessants en mode graphique en sont la cause).

Lorsqu'un programme est en cours d'exécution, l'inspecteur d'objet disparaît de l'écran. Il ne peut être rendu visible que par la fin de l'exécution du programme.

Lorsqu'un programme plante, il est possible d'en arrêter l'exécution via le menu 'Exécuter | Suspendre l'exécution'. Cependant, il est souhaitable dans la mesure du possible, de sortir de l'application fautive par le bouton système car cela permet "de sortir proprement" et de libérer les ressources systèmes utilisées. Si l'on choisit l'option "Suspendre l'exécution", les ressources peuvent ne pas être libérées… au bout de quelques séances de débogage, Windows peut se bloquer.

5.4.3 Contraintes systèmes

▲ **Niveaux de fenêtres :**
Afin de ne pas gaspiller inutilement les ressources systèmes, et aussi dans le but de simplifier l'utilisation des applications, Windows impose qu'il n'y ait pas plus de deux niveaux de fenêtres filles dans une application (à l'exclusion des boîtes de dialogue et des messages).

▲ **Utilisation de composants complexes :**
Il faut aussi faire attention aux objets que l'on utilise pour construire une interface graphique moderne. Les tendances actuelles (multi-fenêtrages, icônes, multiplication des boutons, etc…) font que les applications actuelles sont plus gourmandes en ressources qu'autrefois (et l'on ne parle pas de l'occupation mémoire).

Par exemple une mode actuelle est d'utiliser un objet de type "onglet" (Word 6 en fait une grande consommation). Si les objets déposés dans les différents feuillets de l'onglet ne sont pas trop complexes, cela peut aller. Mais si toute l'application est gérée au sein d'une seule fenêtre contenant un onglet, chaque feuillet comprenant plusieurs objets (dont des objets d'accès aux bases de données, les plus gourmands), la consommation en ressources systèmes peut devenir très importante. En effet l'onglet est considéré comme un seul objet et est chargé d'un seul bloc en mémoire avec tous les composants qu'il contient.

▲ **Chargement en mémoire des différentes fenêtres :**
Il est possible de charger en mémoire toutes les fenêtres constituant une application lors du démarrage de l'application. Cette option peut être gourmande en ressources. Elle est en tous cas peu efficace, car certaines fenêtres peuvent ne jamais être utilisées pendant une session de travail, voire dangereuse.
Il est donc généralement préférable de ne charger les fenêtres que lorsque le besoin s'en fait sentir, après une action particulière de l'utilisateur.

▲ **Découpage du programme en modules dynamiques :**
Les programmes graphiques ont souvent une taille très importante. Il est souhaitable de découper un programme un peu ambitieux en plusieurs modules chargeables dynamiquement (DLL).

▲ **Etre efficace :**
Il ne suffit pas de créer une interface graphique utilisant les dernières nouveautés rendues possibles par les bibliothèques de classes ni même de respecter toutes les normes et standard pour qu'une application ait du succès. Encore faut-il que l'application soit réellement utilisable: pour ce faire il faut que chaque fenêtre et chaque boite de dialogue ne contienne que les composants utiles à un moment donné, et surtout qu'il n'y en ait pas trop : une fenêtre contenant trop de boutons, trop de zones de saisie, trop d'options devient inutilisable.

La création d'une interface peut être très longue. Surtout que, qu'on le veuille ou non, l'esprit est habitué à ce qui est ordonné : il faut donc aligner les boutons et les différents cadres, et cela finit par prendre du temps.

6 Généralités sur les composants

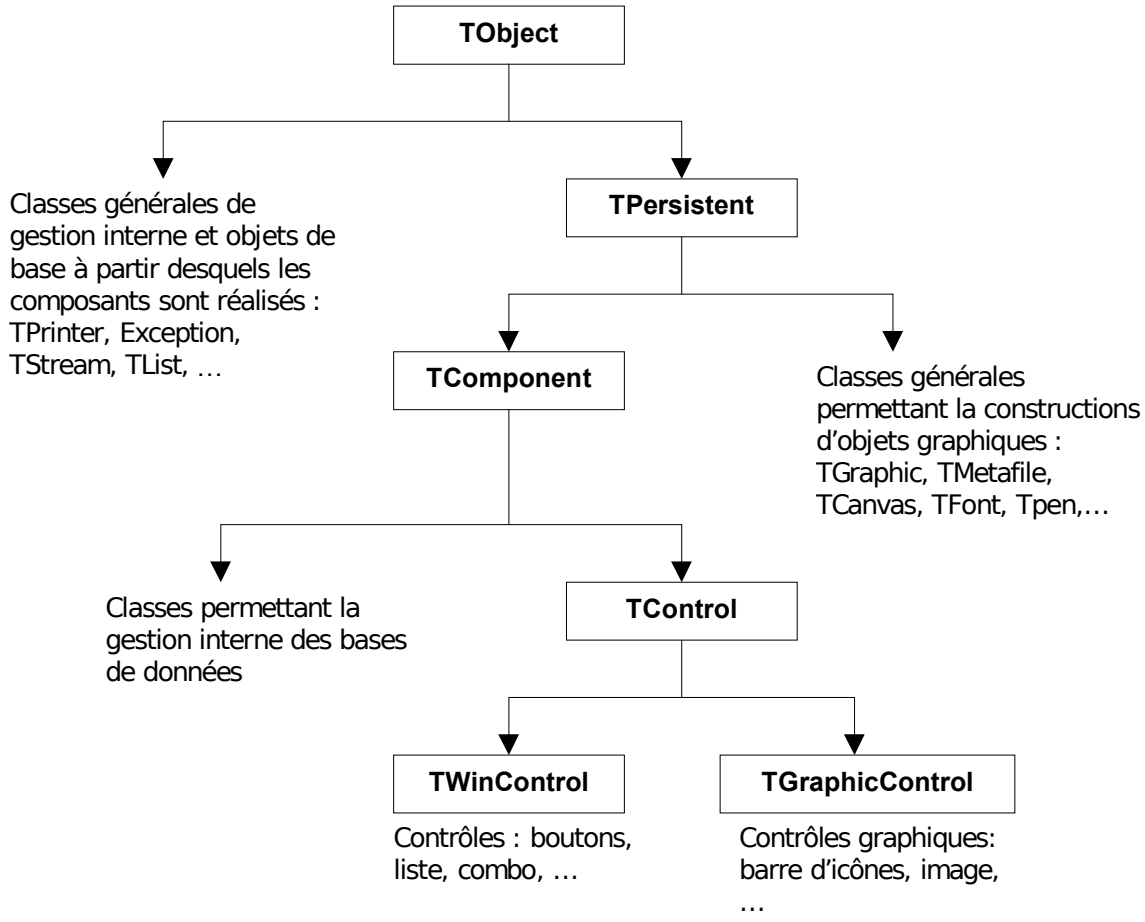
6.1 Les différentes classes

Il serait fastidieux de décrire d'emblée toutes les classes constituant la bibliothèque VCL (Visual Composant Library) de Delphi. Il est plus profitable d'apprendre à utiliser l'aide en ligne afin de savoir quelles classes sont adaptées aux besoins de l'application, quelles seront les propriétés qui seront initialisées ou modifiées à l'exécution, quelles seront les méthodes utilisées et les événements gérés.

6.1.1 Classes de base de la bibliothèque VCL

La bibliothèque VCL est composée d'un nombre important de classes d'objets dont seules certaines correspondent aux composants accessibles via la palette de composants. Toutes ces classes dérivent plus ou moins directement de la classe TObject.

En règle générale on n'utilise pas directement cette classe d'objets (car ses méthodes associées sont redéfinies dans les classes dérivées). Mais dans certains cas, il est intéressant de se rappeler que n'importe quel composant est avant tout un objet.



A partir de la classe TObject de nombreuses classes sont dérivées :

- Une première famille est composée des classes qui permettent de constituer des objets évolués (objets "liste", Objets "flux", etc…)
- L'autre famille est, à partir de la classe TPersistent, constituée principalement par les classes d'objets graphiques , qui servent eux aussi à la constitution d'objets évolués, tels les objets "fonte, "pinceau", "bitmap", etc… , et la classe TComponent

La classe TComponent est le point de départ de tous les composants visuels (ceux qui apparaissent à l'écran lors de l'exécution) ou non.

Les classes dérivées de TComponent sont toutes issues de deux branches distinctes :

- La branche dérivée de la classe TControl ;
- La branche composée des composants servant à la manipulation interne des bases de données (les composants non visuels assurant le lien entre l'application et le moteur de base de données BDE).

La classe TControl, enfin, est dérivée principalement en TWinControl et TGraphicControl et leurs classes dérivées respectives.

Il y a lieu de distinguer les composants dérivées de la classe TWinControl de ceux dérivés de TGraphicControl.

Les contrôles dérivés de TWinControl (ex : les boutons, les listes, les cases à cocher; etc ....) peuvent détenir le focus (c'est à dire qu'ils peuvent devenir le composant actif), sont gérés par Windows à l'aide de handles et peuvent être parents d'autres composants.

Les contrôles dérivés de TGraphicControl ne peuvent pas détenir le focus, ne sont pas gérés par handle et ne peuvent pas contenir d'autres composants.

Un handle est un entier qui permet à Windows d'identifier chaque objet pouvant recevoir des messages.

En fait, Windows considère les contrôles dérivés de TWinControl comme des "fenêtres" et les gère comme telles.

Dans certains cas, il y a peu de différences de comportement entre un TWinControl et un TGraphicControl. On peut alors se poser la question de savoir quel est le composant à utiliser. Sous réserve d'autres contraintes, il est intéressant de noter que la mise en place de handles est synonyme de consommation de ressources.

Par exemple, il y a peu de différence entre un BitButton, qui est un TWinControl et un SpeedButton qui est un TGraphicControl. Cependant, l'utilisation massive de SpeedButton dans une barre d'icônes est préférable quand on sait que certaines barres peuvent contenir jusqu'à 50 boutons.

6.1.2 Déclaration d'un objet

Il faut distinguer la manière dont un objet est déclaré selon qu'il s'agit d'un composant ou d'un objet créé à partir d'une classe "interne".

▲ **Cas d'un composant :**
A chaque fois qu'un composant est rajouté, en le déposant sur la fiche, le code correspondant est généré automatiquement dans l'unité liée.

Si l'on rajoute un bouton à une fiche, le code suivant est rajouté automatiquement dans la section 'interface ':

```
type
        TForm2 = class(TForm)                   { nom de la fiche }
         Button1: TButton;              { nouveau composant }
        ...
```

Par ailleurs cet objet est créé implicitement lorsque la feuille qui le contient est créée.

▲ **Cas d'un objet instancié à partir d'une classe interne :**
Dans ce cas, il faut :

- Déclarer l'objet, en fonction de la classe souhaitée (comme pour une variable quelconque);
- Créer l'objet explicitement, afin que la zone mémoire correspondante soit attribuée.

Exemple :

```
procedure Form1.Button1Click (Sender : TObject);
    var
        MaListeChainee : TSringList; { Instanciation de l'objet }
    begin
        MaListeChainee := TStringList.Create ;
            { Création de l'objet }
        MaListeChainee.Assign (ListBox.Items); { Initialisation }
        ...
        MaListeChainee.Free ; { Libération de la zone mémoire }
    end ;
```

Dans cet exemple, si l'on oublie de créer explicitement l'objet, les instructions qui suivent feront planter le système.

Il ne faut pas oublier de libérer la mémoire lorsque l'on a fini d'utiliser l'objet.

6.2 Les composants

Chaque composant possède ses propres propriétés, ses propres méthodes et réagit à divers événements. Certaines caractéristiques se retrouvent fréquemment. Il est intéressant de les connaître pour être capable, à la lecture des informations contenues dans l'inspecteur d'objet ou dans l'aide en ligne de manipuler rapidement le composant.

Les composants sont construits à partir de plusieurs classes internes : une propriété d'un composant étant alors un objet d'une de ces classes.

6.2.1 Principales classes internes

Les classes internes permettent de définir des objets de base à partir desquels les composants sont construits. Comme ils sont non visuels, ils ne sont accessibles, la plupart du temps, qu'à l'exécution.

Les principaux objets sont :

Objet	Unité	Description
TBitmap	Graphics	Image de type bitmap
TBrush	Graphics	Pinceau
TCanvas	Graphics	Zone de dessin
TClipboard	Cliprd	Presse papiers
TFont	Graphics	Fonte
TList	Classes	Liste générique d'objets
TPen	Graphics	Crayon
TPicture	Graphics	Graphique
TPrinter	Printers	Imprimante
TStringList	Classes	Gestion d'une liste de chaînes de caractères
TStrings	Classes	Manipulation de chaîne de caractères

Tous ces objets ne sont pas étudiés. Il faut connaître les caractéristiques de certains d'entre eux qui utilisent tous, plus ou moins, les propriétés et méthodes suivantes :

6.2.2 Propriétés et méthodes générales :

Propriété Items :
La propriété Items est un tableau d'éléments quelconque. On peut accéder à chaque élément d'un tableau de type Items par son indice, de type entier (Items[Indice]).

L'indice 0 permet d'accéder au premier élément du tableau.

Propriété Count :
La propriété Count permet de savoir combien il y a d'éléments dans un tableau de type Items.

Il y a un décalage de 1 entre la valeur de la propriété count et l'indice correspondant.

Exemple : Lorsque il n'y a qu'une entrée dans le tableau, Count vaut 1 mais l'indice correspondant à l'entrée est 0.

Propriété Strings :
La propriété Strings permet d'accéder à une chaîne spécifique d'un objet chaîne ou liste de chaînes. Il faut spécifier la position de la chaîne dans la liste de chaînes avec le paramètre Index (Strings [Index]). L'indice 0 référence la première chaîne.

Pour trouver l'indice d'une chaîne particulière, il faut appeler la méthode IndexOf().

Strings est la propriété par défaut des objets chaîne. On peut donc l'omettre et traiter l'objet chaîne lui-même comme un tableau indicé de chaînes.

Dans l'exemple suivant, Lines est une propriété objet chaîne d'un composant Memo (TMemo). Ces deux lignes de codes sont valides et équivalentes :

```
Memo1.Lines.Strings[0] := 'C''est la première ligne';
Memo1.Lines[0] := 'C''est la première ligne';
```

Propriété Sorted :
Si Sorted (qui est une variable de type boolean) est à True, les chaînes sont triées en ordre croissant. Si Sorted est à False, elles ne sont pas triées.

Propriété Duplicates :
Duplicates détermine si les chaînes dupliquées sont autorisées dans la liste des chaînes triées d'un objet liste de chaînes. Les valeurs possibles sont :

- **dupIgnore** Les tentatives d'ajout d'une chaîne dupliquée dans la liste des chaînes triées sont ignorées.
- **dupAccept** Les chaînes dupliquées peuvent être ajoutées dans la liste des chaînes triées.
- **dupError** L'ajout d'une chaîne dupliquée provoque une exception EListError.

Si la liste n'est pas triée, la valeur de Duplicates est sans effet.

Méthode IndexOf () :
La méthode IndexOf () renvoie l'indice de l'élément d'un tableau de type Items passé en paramètre.

Renvoie -1 si l'élément n'a pas été trouvé.

Méthodes Add (), Delete () :
La méthode Add () permet de rajouter un élément passé en paramètre dans un tableau de type Items.
La méthode Delete () permet de supprimer d'un tableau de type Items l'élément correspondant à l'indice passé en paramètre.

Méthodes Create et Free :
La méthode Create alloue la mémoire nécessaire à la création de l'objet et initialise ses données de façon appropriée.
La méthode Free détruit l'objet et libère la mémoire qui lui est associée.

Chaque objet peut disposer d'une méthode Create adaptée à la création de ce type d'objet particulier.

On utilise Free pour tout objet créé à l'aide de la méthode Create.

Méthode Assign() :
La méthode Assign () affecte un objet dans un autre.

La forme générale d'un appel à Assign() est la suivante :

```
Destination.Assign(Source );
```

où Destination indique l'objet de l'affectation et Source l'objet initial.

Par exemple, si Destination et Source sont des objets liste de chaînes (TStrings), les chaînes contenues dans Source sont copiées dans Destination.

Bien que le compilateur autorise tout objet TPersistent dans un appel à Assign(), ce dernier ne peut aboutir lors de l'exécution que si les objets concernés sont en mesure d'effectuer l'affectation.

Par exemple, si Destination est un bouton (TButton) et Source une boîte d'édition (TEdit), un appel à Assign() provoque une exception EConvertError à l'exécution.

Méthode Clear :
La méthode Clear efface toutes les chaînes d'une liste de chaînes.

Méthode Find () :
La méthode Find () recherche la chaîne spécifiée dans la liste de chaînes stockée dans un objet liste de chaînes.

Son prototype est :

function Find(const chaine: string; var Index: Integer):Boolean;

Si la chaîne spécifiée comme valeur du paramètre chaine est trouvée, Find () renvoie True (False si elle n'est pas trouvée) et la position de la chaîne dans la liste de chaîne est affectée à la valeur du paramètre Index (de base 0).

Méthode LoadFromFile () :
La méthode LoadFromFile (const Filename : string) lit le fichier spécifié par FileName et charge ses données dans l'objet ou le composant. Les objets 'graphique' chargent des graphiques, le conteneur OLE un objet OLE, et l'arborescence et les objets 'chaîne' chargent du texte.

Méthode SaveToFile() :
La méthode SaveToFile (const FileName: string) enregistre un objet dans le fichier spécifié par FileName. Les objets 'graphique' enregistrent un graphique dans un fichier, le conteneur OLE enregistre un objet OLE dans le fichier et les objets 'arborescence' et 'chaîne' enregistrent du texte dans le fichier.

Principales classes internes :
Les trois classes internes, que l'on retrouve ensuite en tant que propriété au sein de composants, sont :

Objet TList :
L'objet TList sert à la gestion des listes d'objets. Il est possible d'accéder à un élément particulier référencé dans la liste à l'aide de la propriété Items.

Pour connaître la position d'un élément dans la liste, on utilise la méthode IndexOf.

On utilise les méthodes Add() et Delete() pour ajouter ou supprimer un élément.

La propriété Count permet de déterminer le nombre d'éléments dans la liste.

Objet TStrings :

Les objets TStrings sont utilisés par différents composants pour manipuler des chaînes.

On utilise les méthodes Add () et Delete () pour ajouter ou supprimer des chaînes dans un objet TStrings.

La méthode Clear efface toutes les chaînes de l'objet TStrings.

La propriété Count contient le nombre de chaînes.
La propriété Strings permet d'accéder à une chaîne particulière à partir de sa position dans la liste des chaînes.

Pour connaître la position d'une chaîne, on utilise la méthode IndexOf () .

On peut ajouter simultanément plusieurs chaînes dans une liste de chaînes en utilisant la méthode AddStrings ().

Il est possible d'affecter un objet TStrings à un autre en utilisant la méthode Assign().

On peut stocker les chaînes dans un fichier et les recharger ultérieurement en un seul bloc à l'aide de la méthode LoadFromFile(). Pour sauvegarder les chaînes dans un fichier, utilisez la méthode SaveToFile ().

Un objet TStrings ne dispose d'aucun moyen pour stocker les chaînes, il exploite plutôt les possibilités de stockage du contrôle qui l'utilise.

Par exemple, la propriété Items d'une boîte liste contrôle est de type TStrings. Les chaînes qui apparaissent dans un contrôle boîte liste sont stockées dans un objet chaînes de boîte liste (TListBoxStrings), dérivé deTStrings.

Pour gérer une liste de chaînes en-dehors d'un contrôle, utilisez un objet liste de chaînes (TStringList).

Objet TStringList :
Un objet TStringList gère une liste de chaînes. On peut ajouter ou supprimer une chaîne de la liste à l'aide des méthodes Add () ou Delete ().

La méthode Clear efface toutes les chaînes d'une liste de chaînes.

La propriété Count contient le nombre de chaînes dans la liste.

Chaque objet liste de chaînes dispose d'une propriété Strings qui permet d'accéder à une chaîne particulière à partir de sa position dans la liste.

Pour connaître la position d'une chaîne, on utilise la méthode IndexOf ().

Pour ajouter simultanément plusieurs chaînes dans une liste de chaînes, utilisez la méthode AddStrings().

Il est possible d'affecter un objet liste de chaînes dans un autre en utilisant la méthode Assign ().

Pour déterminer si une chaîne donnée existe dans une liste de chaînes, appelez la méthode Find () .
Pour trier la liste de chaînes, utilisez la méthode Sort. Pour déterminer si la liste est triée, on peut tester la valeur de la propriété Sorted. Il est possible d'accepter ou non les chaînes en double en utilisant la propriété Duplicates.

On peut stocker les chaînes dans un fichier et les recharger ultérieurement en un seul bloc à l'aide de la méthode LoadFromFile(). Pour sauvegarder les chaînes dans un fichier, utilisez la méthode SaveToFile() .

	TList	TStrings	TStringList
Propriétés			
Items	▪		
Count	▪	▪	▪
Strings		▪	▪
Sorted			
Duplicates			▪
Méthodes			
Add	▪	▪	▪
Delete	▪	▪	▪
Clear	▪	▪	▪
Create	▪	▪	▪
Free	▪	▪	▪
IndexOf	▪	▪	▪
Assign		▪	▪
LoadFromFile		▪	▪
SaveToFile		▪	▪
AddStrings		▪	▪
Sort	▪		▪

Récapitulatif des propriétés et méthodes utilisées

6.2.3 Principales propriétés des composants

Nom et étiquette :
Tous les composants comportent la propriété Name qui permet de donner un nom logique à l'objet créé.

Grâce à ce nom logique on peut manipuler l'objet par programmation (en modifiant ses propriétés, en invoquant ses méthodes, etc....).

On ne peut modifier le nom d'un objet que pendant la phase de conception. Cette modification est reportée automatiquement dans toutes les lignes de codes gérées automatiquement par Delphi. Par contre elle n'est pas répercutée dans les lignes de code créées par le programmeur.

En conséquence, si l'on doit modifier le nom d'un composant il faut le faire dans l'inspecteur d'objet afin que cette modification soit répercutée automatiquement dans tous les codes générés par Delphi.

La valeur donnée à la propriété nom doit respecter les règles de dénomination des identifiants du Pascal (commencer par une majuscules, pas d'espaces, etc ...).

Si le nom prévu est déjà utilisé par un autre composant (du même type), une erreur est signalée par Delphi.

Plusieurs objets possèdent aussi une propriété Caption qui est une "étiquette" apparaissant à l'écran (la légende affichée sur le composant).

Ce peut être le titre d'une fenêtre, le nom apparaissant sur le haut d'un bouton, etc..... Cette propriété n'est donc qu'une propriété "apparente". Toutefois, par défaut la valeur de la propriété Caption est le nom de l'objet : si l'on modifie ce dernier la valeur de Caption est modifiée tant qu'une valeur spécifique n'est pas fournie.

Taille et dimension :
La plupart des objets possèdent différentes propriétés permettant de les dimensionner et de les positionner. Les propriétés Left, Top permettent de positionner l'objet sur la fiche. Les propriétés Height et Width permettent de modifier ses dimensions.

Si l'on modifie les dimensions et la position à la souris, cela a un effet immédiat sur les valeurs affichées dans l'inspecteur d'objet (et réciproquement).

Certains objets, dont les feuilles, possèdent des propriétés indiquant leurs dimensions internes (la dimension de l'objet moins la dimension cumulée des bordures, cadres, etc...): il s'agit des propriétés ClientHeight et ClientWidth.

Propriété Text :
Cette propriété est présente dans tous les objets qui permettent la saisie et la modification d'une chaîne de caractères. Elle correspond à la valeur du texte qui apparaît et qui peut être modifié.

Propriétés Font et Color :
La propriété Font permet de modifier l'aspect des polices de caractères utilisées avec le composant (titre, étiquette, zone de texte, etc....).

La propriété Color permet de modifier la couleur de certains composants.

Ces deux propriétés sont en fait des objets comportant plusieurs propriétés et méthodes.

Si l'on initialise à une fonte particulière (type de police, couleur, taille, etc ...) la propriété 'font' du composant TForm, cette modification est reprise automatiquement dans tous les composants utilisés dans cette fiche (sauf si une initialisation contraire intervient alors).

Propriétés Visible, Enabled et ReadOnly :
La propriété Visible est un booléen permettant d'afficher ou de rendre invisible un objet.
La propriété Enabled est un booléen qui rend actif ou inactif un composant: lorsqu'un composant est inactif, il est rendu "grisé" et les événements associés ne sont pas pris en compte. La propriété ReadOnly, lorsqu'elle est activée, permet de rendre les données associées au composant accessibles en lecture seule.

Lorsqu'un événement est inactif les textes qui lui sont associés (exemple : texte sur le haut d'un bouton) apparaissent estompés.

Ces propriétés permettent de sécuriser une application en forçant l'utilisateur à n'exécuter certaines actions qu'à certains moments bien précis.

Propriétés de listes:
Plusieurs composants (TListBox, TComboBox, TMemo, etc…) sont des containers à "lignes". On accède à ces lignes grâce aux propriétés Items ou Lines, de type TStrings.

Si un composant possède une propriété du type Items, il possède aussi la propriété ItemIndex qui lui permet de savoir quel est l'élément de la liste qui est actif.

Si le premier élément est sélectionné, ItemIndex vaut 0. Tant qu'aucun élément n'est sélectionné, ItemIndex vaut -1.

Propriété Cursor :
La propriété Cursor de chaque composant permet de modifier l'aspect du curseur lorsque celui-ci "passe" au-dessus du composant concerné (croix, sablier, etc…).

Propriété Hint :
La propriété Hint permet d'afficher une bulle d'aide concernant le composant lorsque le curseur de la souris passe dessus.

Pour que le mécanisme fonctionne il suffit d'initialiser la propriété Hint avec une chaîne de caractères correspondant au message à afficher et positionner la propriété ShowHint = True.

6.2.4 Principales méthodes utilisées

Méthode Create
Avec le paramètre 'self', cette méthode permet de créer dynamiquement (et non lors de la phase de conception) un composant.

Exemple :

```
MonEdit := TEdit.Create (self);
```

Méthodes Repaint et Refresh :
La méthode Repaint oblige le contrôle à repeindre son image à l'écran mais sans effacer ce qui est déjà affiché. Pour effacer avant de repeindre, il faut appeler la méthode Refresh.

Méthode SetFocus :
Cette méthode attribue la focalisation au contrôle (c'est lui qui devient actif).

On ne peut attribuer le focus à un composant qui n'est pas créé. En particulier on ne peut l'attribuer à un composant tant que la fenêtre qui le contient n'est pas créée.

Méthode Show et Hide :
Ces méthodes rendent visible ou non une fiche, ou un contrôle, en modifiant la propriété Visible.

Show met la propriété Visible à True, Hide la positionne à False.

6.2.5 Principaux événements utilisés

Les événements les plus souvent utilisés pour créer des gestionnaires d'événements sont :

Evénement OnClick :
Cet événement se produit lorsque l'utilisateur clique sur le composant.

Mais cet événement est aussi déclenché lorsque :

- L'utilisateur sélectionne un élément dans une grille, une arborescence, etc....en appuyant sur une touche de direction.
- L'utilisateur appuie sur Espace alors qu'un bouton ou une case à cocher détient la focalisation.
- L'utilisateur appuie sur Entrée alors que la fiche active a un bouton par défaut (spécifié par la propriété Default).
- L'utilisateur appuie sur Echap alors que la fiche active a un bouton annulation (spécifié par la propriété Cancel).
- L'utilisateur appuie sur la combinaison de raccourci clavier d'un bouton ou d'une case à cocher.
- La propriété Checked d'un bouton radio est mise à True.
- La valeur de la propriété Checked d'une case à cocher est modifiée.
- La méthode Click d'un élément de menu est appelée.

Pour une fiche, un événement OnClick se produit lorsque l'utilisateur clique dans une zone vide de la fiche ou sur un composant indisponible.

Evénements OnEnter et OnExit :
L'événement OnEnter se produit lorsqu'un composant devient actif.
L'événement OnExit se produit lorsque la focalisation passe du composant à un autre.

Les événements OnEnter et OnExit ne se produisent pas lors du passage d'une fiche à une autre ou d'une application à une autre.

Evénements OnKeyDown, OnKeyUp, OnKeyPress :
OnKeyDown se produit lorsqu'un utilisateur appuie sur une touche alors que le composant détient la focalisation.
OnKeyUp se produit lorsque l'utilisateur relâche une touche appuyée.
OnKeyPress se produit lorsqu'un utilisateur appuie sur une touche alphanumérique.

Le gestionnaire associé à un événement OnKeyDown ou OnKeyUp peut répondre à toutes les touches du clavier, y compris les touches de fonction et les combinaisons de touches avec les touches Maj, Alt et Ctr ou l'appui des boutons de la souris.

Le paramètre Key du gestionnaire d'événement OnKeyPress est de type Char ; cependant, l'événement OnKeyPress recense le caractère ASCII de la touche appuyée. Les touches ne correspondant pas à un code ASCII (Maj ou F1 par exemple) ne génèrent pas d'événement OnKeyPress. Pour répondre aux touches non ASCII ou aux combinaisons de touches, il faut utiliser les gestionnaires d'événement OnKeyDown ou OnKeyUp.

Evénements OnMouseDown, OnMouseUp, OnMouseMove :
OnMouseDown se produit lorsque l'utilisateur appuie sur un bouton de la souris alors que le pointeur de la souris est au-dessus du composant. OnMouseUp se produit lorsque l'utilisateur relâche un bouton de la souris enfoncé alors que le pointeur de la souris est au-dessus d'un composant. OnMouseMove se produit lorsque l'utilisateur déplace le pointeur de la souris alors que celui-ci est au-dessus d'un composant.

Le paramètre Button de l'événement OnMouseDown identifie le bouton de la souris qui a été appuyé. En utilisant le paramètre Shift du gestionnaire d'événement OnMouseDown, il est possible de répondre à l'état de la souris et des touches mortes (les touches mortes sont les touches Maj, Ctrl et Alt).

Le gestionnaire d'événement OnMouseUp peut répondre au relâchement des boutons gauche, droit et milieu de la souris et aux combinaisons des boutons de la souris et des touches mortes.

6.3 Accès aux composants et lancement des événements

Delphi propose un ensemble de possibilités pour faire en sorte que certains composants soient accessibles à l'aide du clavier et que certains événements, associés à des composants, soient déclenchés au clavier.

6.3.1 Accélérateurs

Si on ajoute le caractère ' & ' devant un caractère donné dans la valeur de la propriété Caption (pour les composants qui la possèdent), il est possible d'activer l'événement qui leur est associé en utilisant la combinaison de touches ' Alt + Caractère ' .

Cette possibilité est conforme à la standardisation Windows. A l'exécution le caractère qui est marqué par le ' & ' apparaît souligné et est considéré comme "l'accélérateur d'accès" au composant .
Si un bouton a la propriété Caption = &Ouvrir , le bouton sera affiché - à l'exécution - avec la légende ' Ouvrir ' et sera activé par la combinaison de touches ' Alt + O '.

Il faut faire en sorte que, pour une unité donnée, les accélérateurs soient uniques (il ne peut y avoir deux accélérateurs ayant le même caractère).

6.3.2 Composant actif

Au sein d'une fenêtre un seul composant est actif : c'est celui qui détient le "focus". Quand un composant a le focus sa propriété Default passe alors à True (elle est ou elle passe à False pour tous les autres composants).

Certains composants ont la propriété Default positionnée à True à la création (BitBtn de type 'Ok' ou 'Oui ').

Le gestionnaire d'événement associé à l'événement par défaut (en général l'événement OnClick) peut alors être lancé par action sur la touche 'Enter '.

De même la propriété Cancel = True fait que le gestionnaire d'événement associé à l'événement par défaut du composant peut être exécuté en appuyant sur la touche ' Echap' quand le composant est actif.

6.3.3 Ordre de tabulation

La plupart des composants qui sont déposés sur une fiche se voient attribuer un numéro d'ordre. Il est alors possible de faire passer le "focus" d'un composant à l'autre en utilisant la touche ' Tab ' du clavier.

Grâce aux possibilités offertes par l'utilisation de 'Tab ' et de 'Enter ' on peut exécuter une grande partie des fonctionnalités de l'application sans utiliser la souris.

Il est possible de modifier "l'ordre de tabulation" en accédant à une boîte de dialogue particulière (par le menu ' Edition | Ordre de tabulation ' ou, grâce à un clic droit sur la fiche, en faisant ouvrir un menu flottant) : on peut alors déterminer précisément l'ordre dans lequel les différents composants seront actifs.

Cette possibilité est très intéressante lorsque l'on doit créer une fenêtre comprenant un grand nombre de zones de saisie (formulaires).

Lors de la conception, on peut décider du composant actif au démarrage de deux manières différentes :
soit il est placé en première position dans la boîte de dialogue 'Ordre de tabulation'.;
soit on initialise la propriété ActiveControl de la fiche avec le nom du composant souhaité.

6.3.4 Partage d'un événement

Il est possible qu'un composant doivent réaliser les fonctionnalités déjà réalisées par un gestionnaire d'événement déjà existant. Plutôt que de réécrire le code correspondant, il est préférable de réaliser les opérations suivantes :

Sélectionner le composant à qui l'on souhaite adapter un gestionnaire d'événement déjà réalisée.
Dans l'onglet ' événement ' de l'inspecteur d'objet choisir l'événement que l'on souhaite utiliser (il faut qu'il soit d'un type compatible avec celui à associer).
Dérouler la liste d'événements gérés (flèche à droite de la zone de saisie) et choisir l'événement dont on souhaite reprendre l'action.

Il faut éviter de double-cliquer dans la zone d'édition associée à l'événement choisi avant de dérouler la liste car cela crée un squelette de gestionnaire d'événement inutile.

A partir de là, le fait d'activer le composant exécutera le gestionnaire d'événement ainsi associé.

Si le gestionnaire d'événement partagé est ultérieurement modifié, la modification sera prise en compte par tous les événements qui lui sont rattachés.

6.3.5 Exécution d'un événement par programmation

Il est enfin possible d'activer un événement par programmation.

Par exemple, si la fonction événement associée à un bouton est :

```
procedure TForm1.Button2Click (Sender: TObject );
begin
        ShowMessage (' Bonjour tous ');
end;
```

Alors il est possible d'exécuter cet événement à distance en l'appelant, comme une procédure normale :

```
procedure TForm1.Button4Click (Sender: TObject );
begin
       Button2Click (Button2 );
end;
```

La boîte de message associée au premier bouton s'affichera.

7 Les principaux composants

Chaque classe et composant de la bibliothèque VCL a une utilité particulière. Il est possible de les classer en différentes catégories de manière à mieux comprendre leurs points communs et leurs différences.

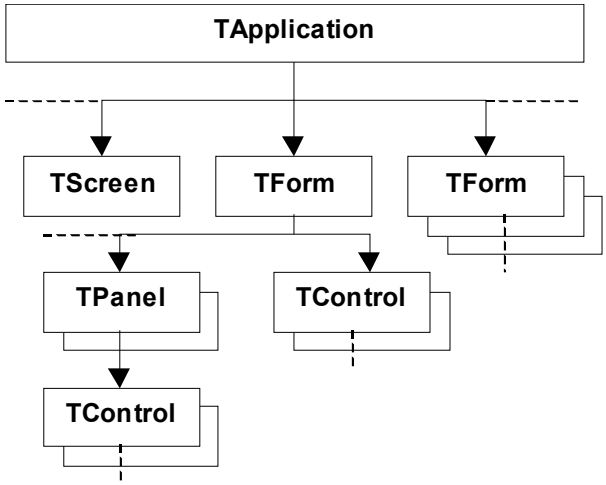
Globalement, on distingue :

- Les classes et composants de haut niveau qui représentent l'application et ses différentes fenêtres.
- Les composants qui permettent de créer une interface évoluée (panneau, labels, cadres, blocs notes, etc ...).
- Les composants plus spécifiquement chargés de réaliser des entrées et des sorties (zone de saisie et d'affichage) ou offrant des possibilités de choix à l'utilisateur (case à cocher, liste, etc, ...).

7.1 Composants du niveau "application"

Une application Delphi est composée d'un nombre variable de fenêtres dont une est la fenêtre principale. Chaque fenêtre est, du point de vue de la programmation, un composant particulier du type TForm. Mais il existe aussi un composant classe spécial, appelé TApplication qui permet de référencer directement l'application.

La structure générale d'une application est alors :



L'objet TApplication est l'objet "parent" de tous les composants. Il est en particulier parent de tous les objets TForm et d'un objet spécifique, appelé TScreen, dont on explicitera le rôle.

Chaque objet TForm est parent de tous les composants qu'il contient. Certains de ces composants (comme le composant Tpanel, TgroupBox, …) peuvent contenir eux même des composants dont ils sont le parent.

7.1.1 Le composant TForm

- Ce composant ne figure pas dans les palettes de composants. Il faut l'appeler explicitement par le menu 'Fichier | Nouvelle Fiche'.

Le fait de créer une nouvelle fiche implique la création automatique d'une unité.

▲ **Description :**
Une fiche est constituée à partir du composant TForm. Ce composant peut contenir d'autres composants. On conçoit une application en plaçant des composants sur une ou plusieurs fiches.

Les fiches peuvent être utilisées en tant que fenêtres ou en tant que boîtes de dialogue.

Une fiche représente, à l'exécution, une fenêtre distincte : en particulier, les dimensions et la position à l'écran données à la fiche seront, par défaut, celles de la fenêtre à l'exécution.

▲ **Principales propriétés :**
Un composant TForm possède un grand nombre de propriétés. En plus de celles, générales, décrites dans le chapitre précédent, il faut connaître la signification d'un certain nombre d'entre elles pour personnaliser la feuille.

Nom	Type	Description
AutoScroll	Boolean	Détermine si les barres de défilement doivent apparaître dans la fiche lorsque la taille de celle-ci est insuffisante pour afficher tous ses contrôles
BorderIcons	TborderIcons	Définissent la présence ou l'absence du menu système et des boutons d'agrandissement et de réduction de part et d'autre de la barre de titre.
BorderStyle	TFormBorderStyle	Définit l'apparence et le comportement de la bordure (retaillable ou non, style 'boite de dialogue" , etc ...)
Menu	TmainMenu	Désigne la barre des menus de la fiche.
ActiveControl	TwinControl	Désigne le composant de la fiche qui sera actif lors de l'affichage de celle-ci.
Canvas	Tcanvas	Permet de réaliser des graphismes sur la surface de la feuille.
FormStyle	TformStyle	Indique si la feuille sera une normale ou de type MDI ou si elle pourra restée toujours au dessus des autres feuilles.
KeyPreview	Boolean	Permet de gérer les événements clavier.
Position	TPosition	Permet de définir la taille et le positionnement de la fiche au démarrage.
WindowState	TwindowState	Permet de déterminer l'état de la fenêtre au démarrage.

Lors de la phase de conception on peut initialiser la plupart des valeurs de ces propriétés.

D'autres propriétés permettent de déterminer précisément la taille et la position de la fenêtre à l'écran.

Si l'on veut que la fenêtre ait la taille de l'écran il faudra que ses propriétés Height et Width correspondent à la taille de l'écran (en tenant compte de la résolution de l'écran). Il faut ensuite initialiser, à l'exécution, les propriétés Left et Top à 0 (ou alors initialiser la propriété Position := poScreenCenter).

La propriété Ctl3D spécifie si la fiche (et ses objets "enfants") sont générés avec un aspect 3D ou non.

Une fenêtre de type "boite de dialogue " (BorderStyle := bsDialog) ne peut afficher de menu (même si celui-ci a été défini lors de la phase de conception).

Par défaut, la propriété Visible est initialisée à False pour toutes autres fiches que celle de la fenêtre principale.

Remarque générale :
Comme le composant fiche est le parent de tous les composants qu'il contient, il n'est pas nécessaire de citer son nom logique pour accéder à ses différentes propriétés ou méthodes à partir des différents gestionnaires d'événements créés.

```
Form1.Caption := 'Fenêtre principale';
```

est équivalent à :

```
Caption := 'Fenêtre principale';
```

Par contre il sera nécessaire d'utiliser le nom de la fiche si on souhaite invoquer une de ses propriétés ou de ses méthodes à partir d'une autre fiche.

▲ **Principales méthodes :**
Le composant TForm possède aussi un nombre impressionnant de méthodes. Parmi celles-ci on peut citer :

Nom	**Action**
Close	Ferme la fiche. La méthode Close appelle d'abord la méthode CloseQuery pour déterminer si la fiche peut être fermée. Si CloseQuery renvoie False, l'opération de fermeture est interrompue
CloseQuery	La méthode CloseQuery est appelée pendant le processus de fermeture déclenché par la méthode Close d'une fiche afin de savoir si cette dernière peut être effectivement fermée. CloseQuery exécute le code rattaché à l'événement OnCloseQuery.
Create	Alloue la mémoire nécessaire à la création du composant et initialise les données de ce dernier de façon appropriée.
Free	Détruit l'objet et libère la mémoire qui lui est associée.
Print	Imprime une image de la fiche
Show	Rend visible une fiche en mettant sa propriété Visible à True.
ShowModal	Rend une fiche active (comme Show) mais la rend également modale.

▲ **Evénements associés :**

Les principaux événements gérés par le composants TForm sont :

Nom	Déclenchement
OnActivate	Se produit lorsque la fiche devient active.
OnClick	Se produit lorsque l'utilisateur clique sur le composant.
OnClose	Se produit lors de la fermeture d'une fiche
OnCloseQuery	Se produit lors d'une action fermant la fiche (lorsque la méthode Close est appelée ou lorsque l'utilisateur choisit Fermeture dans le menu système de la fiche). Il est possible d'utiliser un gestionnaire d'événement OnCloseQuery afin de demander à l'utilisateur une confirmation de la fermeture
OnCreate	Permet de créer une fonction associée servant à initialiser la fiche (valeurs initiales des propriétés et exécution de traitements spécifiques).
OnExit	Se produit lorsque la focalisation passe du composant à un autre.

Une fiche ne peut être fermée que si la variable globale OnClose, de type booléen, est positionnée à True. Il est donc possible de créer un gestionnaire d'événement afin d'initialiser cette variable.

Quelle que soit la manière dont la fenêtre sera fermée (par un bouton approprié, par le menu système ou par un raccourci clavier) le gestionnaire d'événement associé à OnCloseQuery est exécuté. Il est alors possible d'y inclure certains contrôles préalables à la fermeture en bloquant celle-ci le cas échéant.

```
procedure TFFille.FormCloseQuery (Sender: TObject; var CanClose: Boolean);
begin
        if MessageDlg ('Etes-vous sûr de vouloir fermer
        l''application ?', mtInformation , [mbYes, mbNo], 0) = mrYes
        then
                CanClose := True
        else
                CanClose := False ;
end;
```

Dans ce cas, la fenêtre n'est fermée que si l'utilisateur confirme son intention. Il est évident qu'il est préférable d'inclure ce code dans le gestionnaire d'événement associé à OnClosequery que de l'inclure dans le code du bouton de fermeture. Dans le premier cas toutes les causes de fermeture sont prises en compte.

Il est intéressant d'utiliser les possibilités offertes par les différents événements amenant à la création et à l'affichage de la feuille. Certains sont systématiquement appelés lors de la procédure de création. Il s'agit, dans l'ordre, des événements :

OnCreate : Evénement appelé lors de la création de la fiche. Les différents composants de la fiche existent déjà (on peut donc les invoquer et les initialiser). Par contre la variable fiche elle-même n'existe pas encore. On ne peut donc pas réaliser des initialisations en l'invoquant (mais on peut le faire en invoquant directement certaines propriétés ou méthodes).

OnShow : Evénement appelé juste avant que la fiche ne devienne visible.

OnResize : Se produit quand la fiche est redimensionnée.

OnActivate : Evénement se produisant lorsque la fiche devient active.

OnPaint : Evénement se produisant lorsque Windows demande à la fiche de se peindre quand elle reçoit la focalisation et devient visible alors qu'elle ne l'était pas avant.

Tant que la fiche n'est pas visible, on ne peut donner le focus à un de ses composants (il faut utiliser pour cela la propriété ActiveControl qui donnera le focus au composant souhaité quand cela sera possible ou alors utiliser un événement à partir de OnShow).

On ne peut réaliser des opérations d'affichage graphique que dans l'événement OnPaint et pas avant.

7.1.2 Création, appel et destruction d'une fiche

Une fiche est créée simplement par utilisation du menu ' Fichier | Nouvelle fiche '.

On peut ensuite composer l'interface graphique en déposant différents composants sur la fiche et en les initialisant. Le code correspondant est généré automatiquement par Delphi dans l'unité associée.

Une fois le projet sauvegardé, la fiche et l'unité possèdent des noms qui leurs permettent d'être manipulés par programmation.

Pour appeler une fiche à partir d'une autre fiche (ou à partir de la fiche principale) il faut:

- Créer un gestionnaire d'événement associé à un composant .La plupart du temps il s'agit d'un gestionnaire associé à l'événement OnClick d'un bouton ou d'une rubrique d'un menu, mais ce n'est pas une obligation et tous les composants peuvent permettre l'apparition d'une fenêtre fille.
- Au sein du gestionnaire, insérer le code permettant l'apparition de la fiche désirée.

A ce niveau, il y a plusieurs possibilités :

- Soit la fenêtre apparaît en mode amodal et on doit alors utiliser la méthode Show.
- Soit elle apparaît en mode modal et on doit utiliser la méthode ShowModal.

Exemple :

```
FFille.Showmodal ;      { La fenêtre FFille apparaît en mode
                                modal}
```

Le code du fichier .DPR du projet est alors le suivant :

```
uses
Forms,
Princip in 'PRINCIP.PAS' {FPrincipal},
Fille in 'FILLE.PAS' {FFille},
{$R *.RES}

begin
        Application.Initialize;
        Application.CreateForm (TFPrincipal, FPrincipal);
        Application.CreateForm (TFFille, FFille);
        Application.Run;
end.
```

Cependant ce mode de création de fiche, qui est le mode par défaut, n'est pas satisfaisant. En effet il implique que toutes les fiches constituant l'application soient créées au démarrage (et qu'elles ne sont

affichées qu'à la demande). Si l'application est composée de quelques dizaines de feuilles les ressources mémoires et systèmes nécessaires risquent d'être insuffisantes. Par ailleurs certaines fiches peuvent avoir besoin de certaines valeurs d'initialisation qui ne seront valides que par action de l'utilisateur dans d'autres fiches (se sera souvent le cas dans les applications gérant des données). L'application risque alors de "planter" au démarrage.

Il est donc préférable de loin de réaliser une création dynamique des feuilles : les zones mémoires nécessaires aux différentes fiches ne sont alors réservées qu'en cas de besoin.

Pour pouvoir réaliser cette création dynamique il faut explicitement créer la feuille dans le gestionnaire d'événement chargé de l'afficher. Il faudra aussi prendre soin de libérer la mémoire lorsque la fenêtre est fermée.

On a alors le code suivant :

```
procedure TFApplication.BOuvrirClick (Sender: TObject);
begin
        FFille := TFFille.Create (Self);
        FFille.ShowModal ; { ou FFille.Show }
        FFille.Free ;
end ;
```

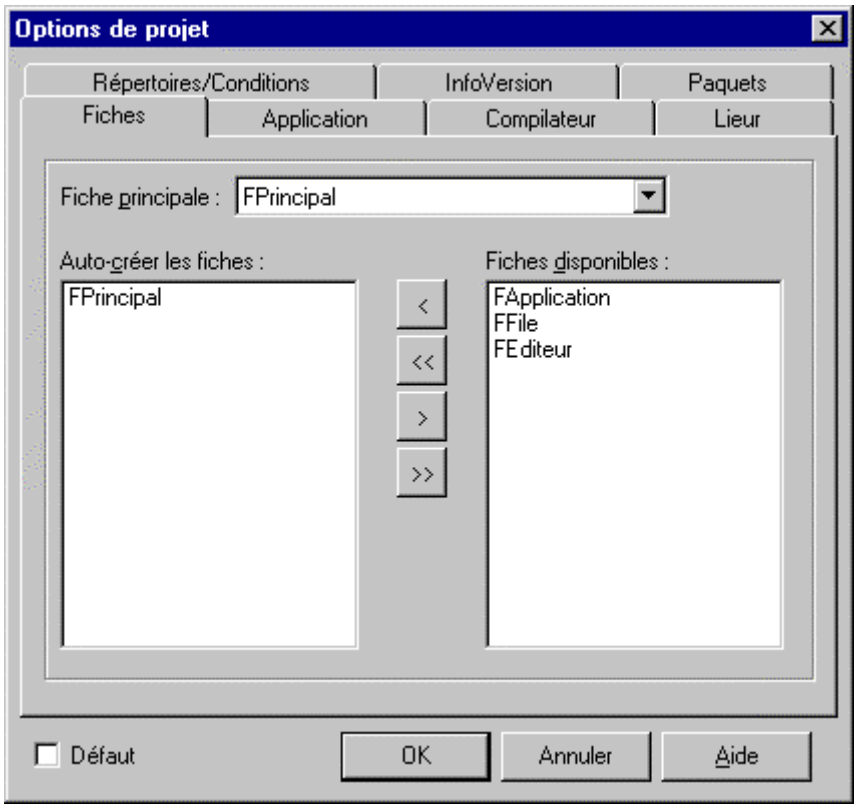
La méthode Create réalise la "construction" de la fiche (avec les composants qui y sont inclus.

La méthode ShowModal affiche la fenêtre. Comme celle-ci est modale toutes les actions de l'utilisateur se déroulent au sein de cette fenêtre. On ne sort de cette ligne de code que lorsque la fenêtre est fermée.

La méthode Free libère les ressources allouées.

Pour que cette méthode d'allocation dynamique soit réellement prise en compte, il faut que l'option "auto-création" des fiches soit annulée dans le projet. Il faut pour cela se rendre dans le menu 'Options | Projet ', onglet " Fiches" pour faire passer les différentes fiches secondaires dans la catégorie " fiches disponibles ".

Si on affiche alors le source du fichier .DPR du projet, on constate que les lignes provoquant la création des fiches secondaires (exemple : Application.CreateForm (TFFille, FFille)) ont disparues.



Projet dont les différentes feuilles secondaires sont disponibles et donc peuvent être créées dynamiquement

▲ **Appel de fiche "protégé" :**

Il est possible que Windows ne soit pas en mesure d'afficher la fenêtre (lors de l'exécution de la méthode ShowModal (ou Show) car il ne dispose plus de suffisamment de ressources.

Il faut donc protéger l'appel à cette méthode en gérant l'exception qui est susceptible d'intervenir (et qui se traduit par un message système à l'écran bien peu esthétique).

Un chapitre est consacré ultérieurement à la gestion des exceptions. Néanmoins il est intéressant de voir quel est le code complet et sécurisé de l'appel d'une fiche :

```
procedure TFApplication.BOuvrirClick (Sender: TObject);
begin
        FFille := TFFille.Create (Self );
        try
                FFille.ShowModal ; { ou FFille.Show }
        finally
                FFille.Free ;
        end ;
end ;
```

Enfin, il faut que l'identificateur de la fiche secondaire soit connu de la fiche "mère". Pour cela il faut que le nom de l'unité constituée par la fiche fille (ici "fille") soit invoqué dans la clause "uses" de la fiche mère.

▲ **Cas des fenêtres "splash" :**

Les fenêtres "splash" sont à la mode. On appelle "fenêtre splash" l'écran d'accueil, contenant en général une image, le nom de l'application et certaines références sociales, qui s'affiche lorsque le programme se charge en mémoire. Une fois le chargement effectué, la fenêtre splash s'efface et ne sert plus.

La conception d'une telle fenêtre est réalisée de la même manière qu'une fenêtre normale. Par contre son appel se fait d'une manière totalement différente. C'est en effet dans le fichier de projet qu'elle est appelée (alors que toutes les autres fenêtres sont appelées à partir des unités).

On a alors le code suivant :

```
begin
        SplashForm := TSplashForm.Create(Application);
        SplashForm.Show;
        SplashForm.Update;
        Application.Title := 'Mon Application' ;
        Application.CreateForm (TMainForm, MainForm);
        ...
        { Création des fiches "auto-créées" }
        ...
        SplashForm.Hide;
        SplashForm.Free;
        Application.Run;
end.
```

7.1.3 Le composant TApplication

Il existe un composant spécial, nommé TApplication, invisible mais automatiquement intégré au projet par Delphi et nommé Application, qui sert à définir l'application dans son ensemble. Il est principalement utilisé pour créer la fiche principale et lancer l'application.

C'est ce composant qui est invoqué dans le fichier projet .DPR et qui est permet de lancer l'application par l'instruction:

```
Application.Run ;
```

On n'utilise pas fréquemment les propriétés et méthodes de ce composant. Néanmoins quelques unes sont intéressantes et permettent d'avoir accès à des renseignements supplémentaires.

▲ **Principales propriétés :**

ExeName	Nom de l'application exécutable (chemin complet)
HelpFile	Nom du fichier d'aide associé (chemin complet)
Hint	Texte apparaissant sous forme de bulle d'aide. Lorsque la propriété 'ShowHint' du composant est à True ainsi que celle des différents composants, on peut utiliser la propriété Hint de l'application pour afficher les messages correspondants directement à chaque fois que l'on passe le curseur sur un composant.
HintColor	Couleur des bulles d'aide de l'application.
HintPause	Délai, en millisecondes, de déclenchement des bulles d'aide.
Icon	Icône du programme lorsque celui-ci est intégré au Gestionnaire de programme.
Title	Texte associé à l'icône dans le gestionnaire de programme.

On peut par exemple utiliser le renseignement fourni par la propriété Exename pour se positionner dans le répertoire de l'exécutable (en utilisant la fonction ExtractFilePath()):

```
ChDir (ExtractFilePath (Application.Exename ));
```

▲ **Principales méthodes :**

MessageBox	Appelle la fonction correspondante de l'API Windows
Run	Lance l'exécution de l'application

▲ **Principaux événements :**

OnActivate	Se produit lorsque l'application devient active (lorsqu'elle est créée ou lorsqu'elle reçoit le focus).
OnException	Se produit lorsqu'une exception non gérée survient dans l'application.
OnHint	Se produit lorsque le curseur de la souris passe au-dessus d'un composant dont la propriété Hint n'est pas vide.

L'utilisation des propriétés Hint et ShowHint, associées à l'événement OnHint, permet de détourner le message d'aide qui doit être affiché dans la bulle d'aide afin qu'il apparaisse aussi ailleurs (le message d'aide du composant passe d'abord dans la propriété Hint de l'application).

Le problème vient du fait que le composant TApplication étant invisible, ses propriétés et événements associés n'apparaissent pas dans l'inspecteur d'objet. Il faut donc créer complètement un gestionnaire d'événement pour qu'il puisse être pris en compte par l'application :

```
procedure TForm1.MontreHint (Sender: TObject);
begin
        PAide.Caption := Application.Hint ;
end;

{Ne pas oublier de déclarer l'en tête de la procédure dans la
section déclaration de l'unité}
```

L'initialisation du gestionnaire d'événement se fera alors sous la forme :

```
procedure TForm1.FormCreate (Sender : TObject);
begin
        ...
        Application.OnHint := MontreHint ;
        ...
end ;
```

▲ **Sauvegarde d'une fiche ou d'un projet :**

Il est possible de sauvegarder une fiche, avec l'ensemble de ses composants, en tant que modèle réutilisable ultérieurement.

Pour cela il faut utiliser le menu 'Options | Galerie... ' afin de prendre en compte la nouvelle fiche dans cette dernière.

7.1.4 Le composant TScreen

Comme le composant TApplication, TScreen est un composant invisible qui est automatiquement intégré au projet. son nom est Screen.

De même ses propriétés et méthodes ne sont pas très connues (il faut dire que la documentation à son sujet est très réduite et qu'il faut presque un hasard pour savoir qu'il existe). Néanmoins certaines de ses caractéristiques sont très intéressantes et, pour tout dire, incontournables.

Le composant TScreen représente l'état de l'écran au fur et à mesure de l'exécution de l'application.

On peut connaître la résolution courante du moniteur en invoquant les propriétés Height et Width. La propriété PixelsPerInch indique le nombre de pixels par pouce tel qu'il est défini dans le pilote du périphérique d'affichage courant.

▲ **Principales propriétés :**

Activecontrol	Indique le contrôle actif dans la feuille active
Activeform	Indique la fenêtre active
Cursor	Permet de gérer le curseur d'une manière globale
Cursors	Tableau des curseurs disponibles
Fonts	Liste des fontes supportées par l'écran
Height	Height indique la hauteur, exprimée en pixels, de l'écran.
Width	Width indique la largeur, exprimée en pixels, de l'écran.

La propriété 'Cursor' de chaque composant permet de modifier l'aspect du curseur lorsque celui-ci "passe" au-dessus du composant concerné (croix, sablier, etc ....).

Par contre, lorsque l'on souhaite modifier le curseur en cours d'exécution (par exemple pour qu'il prenne la forme d'un sablier indiquant qu'un traitement est en cours), il faut utiliser la propriété 'Cursor' de TScreen qui permet de modifier de manière globale son apparence.

```
procedure TForm1.Button1Click(Sender: TObject);
var
        i : LongInt ;
begin
        i := 0 ;
        Screen.Cursor := crHourGlass ;
        while i < 1000000 do
                inc (i);
        Screen.Cursor := crDefault ;
end;
{ Le curseur prend la forme d'un sablier lorsque l'on clique sur
un bouton puis reprend son aspect habituel }
```

7.2 Composants permettant la réalisation d'une interface

Quelques composants sont essentiellement destinés à la constitution d'interfaces évoluées. Il est important de connaître leurs caractéristiques pour pouvoir les utiliser de manière optimale.

7.2.1 Le composant TPanel

Le composant TPanel (en français : volet ou panneau) est un composant que l'on utilise souvent pour réaliser des interfaces complexes. Sa caractéristique principale est qu'il peut servir de "container" à d'autres composants. Ceci permet d'agir directement sur ce composant lorsque l'on souhaite agir sur l'ensemble des composants.

Un composant TPanel peut servir à créer des "cadres" dans lesquels d'autres contrôles peuvent ensuite être placés. Il peut aussi être utilisé comme base pour créer une palette d'outils, une barre d'icônes ou une barre d'état.

▲ **Effets de la parenté :**

Outre le fait que le composant sert de container aux composants qu'il contient, il y a "propagation" de certaines de ses propriétés :
Lorsque l'on déplace ce composant, tous les composants inclus sont déplacés.
La palette d'alignement agit sur les composants inclus par rapport aux dimensions du container.
Les modifications des propriétés Visible et Enabled s'appliquent d'un seul coup au container et à tous les composants inclus.
Les propriétés Color, Font, CTL3D sont elles aussi propagées.

Le contrôle TPanel doit être placé sur la fiche avant les contrôles qu'il contiendra. Cela étant réalisé :
Si le cadre est déplacé, tous les composants qu'il contient sont déplacés en même temps.
Il est possible ensuite d'aligner ces différents contrôles par rapport au cadre TPanel en utilisant la palette d'alignement.

▲ **Principales propriétés :**

Align	Positionner le composant à l'intérieur de la fenêtre	
Alignment	Positionne le texte (Caption) du composant.	
BevelInner **BevelOuter** **BevelWidth**	Permet de spécifier l'aspect des biseaux des bordures du composant. En jouant sur l'ensemble des possibilités proposées on peut créer des interfaces soignées.	
	BvNone	Pas de biseau.
	BvLowered	Le biseau semble enfoncé.
	BvRaised	Le biseau semble en relief.
	bvSpace	Le biseau est représenté par un espace si son type n'est pas bkTile. Sinon, le biseau apparaît en relief.
BorderStyle **BorderWidth**	Propriétés permettant de jouer sur l'aspect des bordures.	

La propriété Align permet de définir l'alignement d'un cadre par rapport à la fiche : il est possible d'aligner les volets pour que leur position relative dans la fiche soit toujours identique, même si l'utilisateur redimensionne la fiche.

On peut alors concevoir une fiche constituée de 3 composants TPanel :

- Un sera positionné tout en haut avec la propriété Align := alTop (ce qui aura pour effet de lui faire occuper toute la partie supérieure de la fiche, quel que soit les redimensionnements ultérieurs de la fenêtre).
- Un sera positionné en bas de la fiche avec la propriété Align := alBottom (il occupe donc tout le bas de la fiche).
- Un sera positionné entre les deux précédent avec la propriété Align := alClient. Il est alors redimensionné automatiquement pour occuper tout l'espace disponible entre les deux cadres précédents.

Il n'y a plus qu'à redimensionner éventuellement l'épaisseur des divers cadres puis de donner un aspect "3D" aux différentes bordures (avec les propriétés Bevel... et Border....) pour disposer d'une application avec barre d'icônes et barre d'état en bas.

Pour construire une barre d'outils ou une palette d'outils, il suffit d'ajouter des turbo-boutons (TSpeedButton) au cadre du haut ainsi que tout autre contrôle utile.

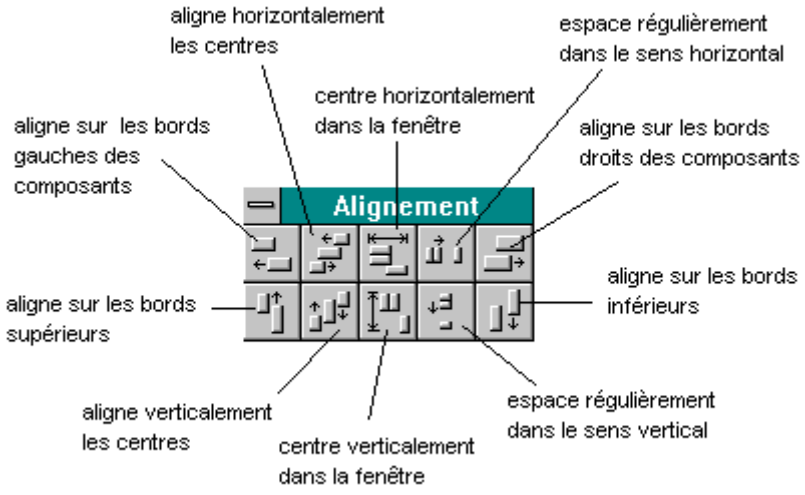
Outre le composant TForm, il existe d'autres composants qui comme TPanel peuvent servir de container à d'autres composant (TGroupBox, TRadioGroup, TPageControl). Ces composants reprennent les caractéristiques propres aux containers.

L'utilisation conjointe de plusieurs TPanel, rendus actifs et visibles à tour de rôle, peut être un moyen efficace pour éviter la multiplication excessive des fenêtres. Néanmoins dans ce cas l'utilisation du composant TNotebook est encore plus efficace.

La palette d'alignement

Lorsque plusieurs composants sont placés sur une feuille, ou à l'intérieur d'un composant Tpanel, il est possible de les aligner rapidement grâce à la palette d'alignement (ou plus facile en les sélectionnant et en cliquant sur le bouton droit -Aligner-).

Pour ce faire, il faut d'abord sélectionner les composants que l'on souhaite aligner. Il faut pour cela les sélectionner à la souris un par un en maintenant la touche ‘Shift’ enfoncée.



7.2.2 Le composant TBevel

Comme le composant TPanel, TBevel permet de réaliser des cadres au sein de l'interface. Mais ce n'est pas un container pouvant regrouper d'autres composants.

TBevel est surtout utilisé pour créer des lignes de séparations et des cadres sur l'interface. Les différents possibilités de configurations sont accessibles à l'aide des propriétés Shape et Style.

Le fait de placer un composant TBevel en tant que cadre au tour d'un ou plusieurs composants risque d'empêcher l'accès à ces derniers si on n'a pas positionné le TBevel " en-dessous" (utiliser l'option "Mettre en arrière plan" dans le menu Edition de Delphi).

Lorsque la sélection est réalisée, on peut faire apparaître la palette d'alignement à partir du menu 'Voir | Palette d'alignement'.

7.2.3 Le composant TLabel

Ce composant permet l'affichage de texte à l'écran. En règle générale, ce texte (qui correspond à la propriété Caption) sert de titre à un autre composant. Il ne peut pas être modifié par l'utilisateur lors de l'exécution (mais il peut l'être par programmation).

▲ **Principales propriétés :**

Alignment	Aligne le texte à l'intérieur de la surface du composant.
AutoSize	Réajuste automatiquement la taille du composant en fonction de celle du texte qu'il contient. Si Autosize :=False, le texte risque d'être tronqué.
FocusControl	Lie le contrôle TLabel à un autre contrôle de la fiche
Transparent	Permet de rendre le fond du contrôle transparent ou non. Intéressant pour placer TLabel au dessus d'un objet image.
WordWrap	Lorsque cette propriété est à True, permet de réaliser du texte sur plusieurs lignes.

TLabel est un composant dérivé de TGraphicControl : il ne peut donc pas avoir le focus. Par contre, il peut être associé à un composant susceptible de recevoir le focus de manière à pouvoir utiliser les raccourcis clavier pour rendre actif le composant "focusable".

Exemple :

Déposer un composant TLabel avec la propriété Caption := 'Nom: ' .
Déposer à coté un composant TEdit avec la propriété Text := ' ' (vide).
Initialiser la propriété FocusControl du TLabel avec le nom de la zone d'édition .

A l'exécution, le fait d'utiliser le raccourci clavier ' Alt +N ' rend la zone d'édition active.

7.2.4 Le composant TTabControl

TTabControl est un ensemble d'onglets ayant l'aspect d'un séparateur de classeur.

Utilisez TTabControl pour ajouter un contrôle contenant plusieurs onglets à une fiche. TTabControl n'est pas constitué de plusieurs pages contenant différents contrôles. TTabControl n'est constitué que d'un seul objet. Quand l'onglet sélectionné change, le programmeur a la charge de réactualiser l'affichage de la page onglet, dans un gestionnaire d'événement OnChange, afin de refléter la modification.

▲ **Principales propriétés :**

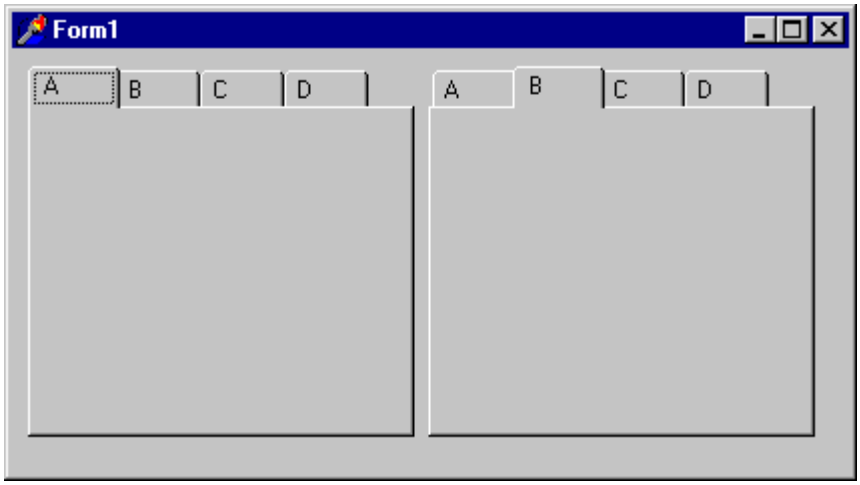
Multiline	A True, les étiquettes de pages d'onglet peuvent être arrangés sur plusieurs lignes.
Tabs	Liste des libellés des onglets. Cette propriété, de type Tstrings, permet de créer de nouveaux onglets.
TabIndex	La propriété TabIndex détermine l'onglet sélectionné par l'utilisateur. TabIndex est l'indice de l'onglet dans la liste des libellés maintenus par la propriété Tabs. Le premier onglet (celui de gauche) a l'indice 0, le suivant l'indice 1, etc. Si aucun onglet n'est sélectionné, TabIndex a la valeur -1. Initialisez TabIndex pour modifier par le programme l'onglet sélectionné du contrôle onglets.

▲ **Principaux événements :**

OnChanging	Se produit immédiatement avant qu'un nouvel onglet ne soit sélectionné.
OnChange	Se produit lors de la sélection d'un nouvel onglet.

7.2.5 Le composant TPageControl

TPageControl affiche plusieurs pages superposées qui sont des objets TTabSheet. L'utilisateur sélectionne une page en cliquant sur l'onglet de la page qui apparaît en haut du contrôle. A la conception, pour ajouter une nouvelle page à un objet TPageControl, cliquez avec le bouton droit de la souris dans l'objet TPageControl et choisissez Nouvelle page.

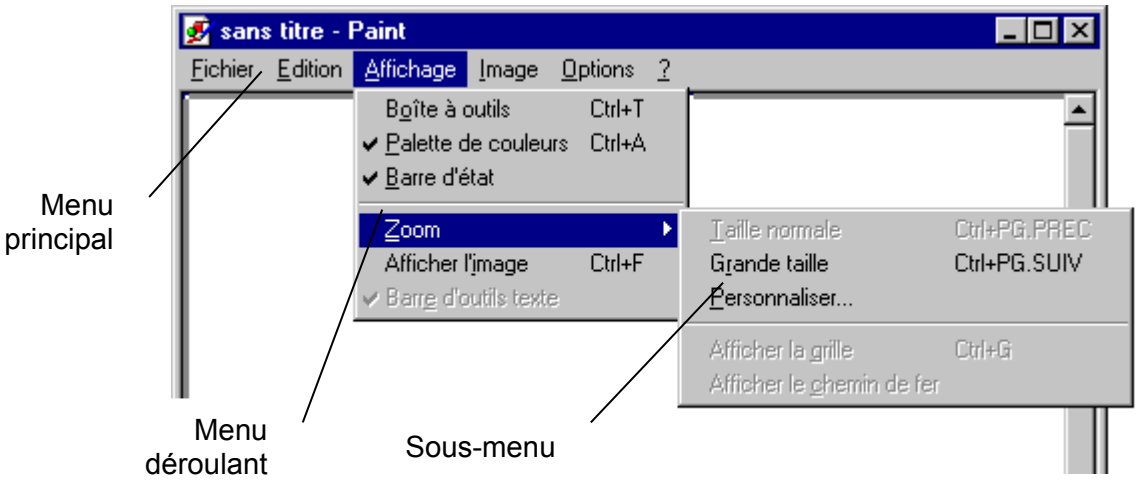


D'un aspect identique, le TTabControl et le TPageControl ont des fonctions différentes

7.3 Les menus

Les menus font partie des éléments standards de l'environnement Windows. Ils sont en fait constitués de deux parties distinctes :

- Un menu principal affiché en haut de la fenêtre de l'application (juste en dessous de la barre de titre);
- Des menus déroulants qui s'affichent lorsque l'on clique sur une des rubriques du menu principal et qui contiennent un nombre variable de rubriques.



Différents composants d'un menu

Chaque rubrique d'un menu déroulant peut ouvrir un sous-menu particulier.

Il existe aussi des menus flottants qui peuvent apparaître à des emplacements divers de la fenêtre et, souvent, être déplacés.

Delphi propose deux composants permettant la création aisée de menus complexes. Ces deux composants font partie du premier onglet de la barre de composants. Ce sont des composants "non-visuels", c'est à dire que l'on peut les poser n'importe où sur la feuille de conception, il n'apparaîtront pas à l'exécution : les menus seront alors attachés à la barre de titre.

7.3.1 Le concepteur de menu

L'utilisation du composant 'Main Menu ' permet d'accéder, en cliquant sur le bouton droit de la souris après l'avoir déposé sur la fiche (ou en passant par la propriété Items du composant), à un utilitaire très pratique permettant de réaliser la totalité du menu :: le concepteur de menu.

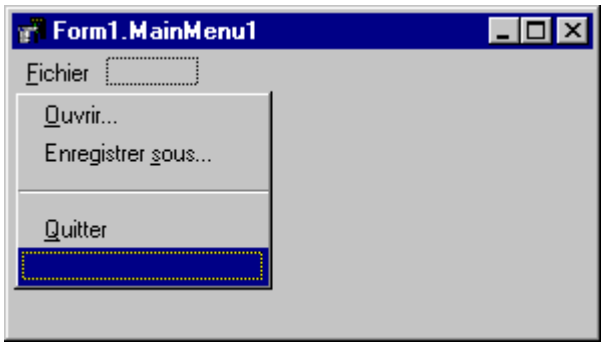
Une des premières choses à faire est de donner un nom au menu (propriété Name).

A l'ouverture, le concepteur présente un menu vierge dont la première case (la 1° rubrique du menu principal) est sélectionnée. Il suffit alors de remplir la propriété Caption (soit directement en sélectionnant la zone concernée du menu, soit dans l'inspecteur d'objet) par le nom de la rubrique, pour que la rubrique soit créée.

Le fait d'utiliser la touche ' flèche droite ' valide l'entrée et prépare une nouvelle rubrique dans le menu principal. De même la touche ' flèche bas ' valide l'entrée et prépare une nouvelle rubrique dans le menu déroulant associé.

Si l'on entre le caractère ' - ' en tant que Caption, une barre de séparation est générée directement dans le menu concerné.

Les rubriques qui ouvriront une boîte de dialogue doivent être terminées par trois points '...'.



Aspect du concepteur de menu

A la conception il y a toujours une case de vide en fin de chaque menu déroulant ainsi qu'au bout du menu principal. Ces cases vides n'apparaissent pas à l'exécution.

▲ **Modifications d'un menu existant :**
Pour insérer une rubrique il faut se positionner sur la rubrique qui suivra la rubrique à insérer et appuyer sur la touche ' Inser '. Une rubrique vierge s'insère devant cette rubrique.
De même si l'on souhaite effacer une rubrique il faut se positionner dessus et appuyer sur la touche ' Suppr '.
On peut déplacer les différentes rubriques en réalisant un "glisser- déplacer" avec la souris.

▲ **Utilisation de "raccourcis clavier" :**
Il est possible d'utiliser le caractère ' & ' pour définir les accélérateurs qui seront reconnus par le système.
Certaines rubriques peuvent être associées à des combinaisons de touches. Il faut alors renseigner la propriété 'ShortCut ' de la rubrique concernée par la combinaison de touche à prendre en compte .....ce qui rajoute automatiquement cette combinaison de touche en regard du titre de la rubrique.

▲ **Création de sous-menu :**
Pour créer un sous-menu à partir d'une rubrique d'un menu déroulant il faut :

- Sélectionner l'élément du menu concerné.
- Appuyer sur le combinaison de touches ' Ctrl + Flèche Droite ' pour créer le première marque du sous-menu à l'aplomb de la rubrique. Une flèche est créer dans le menu déroulant pour indiquer qu'il existe un sous-menu pour cette rubrique.
- Renouveler les opérations habituelles pour composer le sous-menu.

Le Turbo-menu, accessible à l'aide du bouton droit de la souris, propose l'utilisation de modèles de menus prédéfinis.

Une fois un menu constitué il est toujours possible de la sauvegarder en tant que modèle, en vue d'une utilisation ultérieure. Pour cela il faut ouvrir le menu contextuel attaché au composant TMenu en cliquant sur celui-ci avec le bouton droit de la souris.

7.3.2 Le composant TMenuItem

On constate que chaque élément du menu créé par le concepteur de menu est un objet particulier, de type TMenuItem qu'il est possible de manipuler individuellement.

▲ **Principales propriétés :**

Checked	Affiche ou non une coche à coté du libellé de la rubrique
Count	Nombre d'élément d'un menu principal (uniquement pour les TMenuItems du menu principal).
Items [i]	Permet d'accéder en programmation à un élément d'un sous menu rattaché.
Short Cut	Raccourci clavier rattaché à l'élément du menu

▲ **Evénement associé à un élément du menu :**
Les différents éléments TMenuItem d'un menu ne réagissent qu'à un seul type d'événement: l'événement OnClick qui correspond à un click sur le bouton gauche de la souris.

Pour chaque élément il n'y a donc qu'à créer un gestionnaire d'événement associé pour réaliser les fonctionnalités souhaitées.

7.3.3 Manipulation des menus par programmation

Par programmation il est possible de modifier l'aspect des différents menus en jouant principalement sur les valeurs des propriétés Enabled, Checked et Visible des différents objets TMenuItem :
La propriété Enabled rend actif ou non, selon sa valeur booléenne, l'élément de menu concerné. Lorsque l'élément est inactif, il apparaît grisé et ne réagit plus aux clicks souris.
La propriété Visible cache ou non un élément du menu.
La propriété Checked, lorsqu'elle est à True, permet d'ajouter une coche à gauche de la rubrique pour indiquer qu'elle est sélectionnée.

```
Option.Checked := NOT Option.Checked ;
{ Cette ligne insérée dans le gestionnaire d'événement permet d'afficher ou de supprimer la 'coche' à
chaque clic sur la rubrique Option }
```

La propriété Items permet d'accéder à un élément du menu.
Par exemple le code suivant rend indisponible un sous-élément du menu :

```
Fichier.Items[2].Enabled := False;
{ Rend inactif le 3° élément du menu Fichier }
```

On peut par ailleurs insérer ou supprimer des éléments de menu, à l'exécution en utilisant les méthodes Insert() et Remove() .
La méthode Add() ajoute un élément de menu en fin de menu.
Par exemple, si l'on veut ajouter un élément de menu dans le menu 'Fichier' d'une application il faut écrire le code suivant :

```
procedure TFMenu.BAjoutMenuClick(Sender: TObject);
var
 nouvItem : TMenuItem ;
 nomItem : string [ 30 ] ;

begin
 nouvItem := TMenuItem.Create (self);
 nouvItem.Caption := Edit.Text ;
 nomItem := 'NouvFich' + IntToStr (nbItem);
 nouvItem.Name := nomItem ;
 inc (nbItem);
 nouvItem.Checked := True ;
 Fichier1.Add (nouvItem);
 Edit.Clear ;
end;
```

Le code de cette procédure est intéressant à plus d'un titre :

- Il faut explicitement créer un nouveau composant de type TMenuItem. Une fois déclaré, il faut le créer par la méthode Create.
- On doit initialiser la propriété Caption du nouveau Item (c'est le texte qui apparaîtra dans le menu). par contre il faut faire attention lorsque l'on initialise la propriété Name car celle-ci doit se conformer aux spécifications des identificateurs Pascal (pas d'espace, etc...) c'est pour cela qu'il ne faut pas utiliser directement le nom proposé par l'utilisateur).

- Enfin on ajoute l'item au menu.

Il est enfin possible qu'une application présente plusieurs menus en fonction du contexte d'exécution. Dans ce cas il faut préparer les différents menus et indiquer, dans la propriété 'Menu' de la fiche, quel est le menu utilisé au démarrage. Les modifications apportées à cette propriété en cours d'exécution permettront les changements de menus.

▲ **Utilisation de la méthode 'Click ' :**
La méthode 'Click' simule un clic de souris provoquant ainsi l'exécution du code rattaché à l'événement OnClick. Cette méthode évite de dupliquer le code des gestionnaires d'événements un bouton pourra donc agir comme un élément de menu en invoquant cette méthode. De même il est possible de faire surgir un menu Pop-Up en utilisant la méthode Popup de ce type de composant.

7.3.4 Les menus "Pop-up"

Le deuxième composant de la palette permet de concevoir des menus surgissants "Pop-Up". La création de tels menus se fait selon les mêmes principes que ceux utilisés pour créer un menu principal.

Un menu "Pop-Up s'affiche lorsque l'on appuie avec le bouton droit de la souris sur un composant ou sur le fond de la fenêtre.

Il faut pour cela que le menu ait été assigné au composant ou à la fiche en initialisant la propriété Popup Menu du composant concerné avec le nom du menu surgissant.

Par défaut (propriété AutoPopup := True) le menu s'affiche lorsque l'on clique avec le bouton droit de la souris. Mais l'on peut aussi l'activer en invoquant la méthode PopUp () qui en plus permet de spécifier l'endroit où le menu va s'afficher.

7.4 Les boutons

Les boutons sont les composants privilégiés mis à la disposition des utilisateurs pour spécifier leurs choix et influencer le déroulement de l'application.

Delphi propose trois composants différents pour créer des boutons : les composants TButton, TBitBtn et TSpeedButton. Chacun, outre des caractéristiques communes, propose des spécificités.

7.4.1 Généralités sur les boutons

▲ **Principales propriétés :**

Propriété	Btn	BitBtn	SpBtn	Description
AllowAllUp		▪		Spécifie si tous les boutons d'un groupement de SpeedButton peuvent être déselectionnés.
Cancel	▪	▪		Indique si le bouton est un bouton d'annulation. Si Cancel := True, l'utilisation de la touche Echap appelle le gestionnaire d'événement associé à l'événement OnClick du bouton.
Default	▪	▪		Indique si le bouton est le bouton par défaut. Si Default := True, l'utilisation de la touche Enter appelle le gestionnaire d'événement associé à l'événement OnClick du bouton
Down			▪	Indique si le composant s'affiche enfoncé ou non
Glyph		▪	▪	Image affichée sur le bouton
GroupIndex			▪	Permet de définir des groupes de SpeedButton
Kind		▪		Indique le type de BitButton
Layout		▪	▪	Positionne l'image sur le bouton
Margin		▪	▪	Distance entre l'image et la limite du bouton
ModalResult	▪	▪		Utilisée si le choix du bouton doit fermer une fiche modale.
Spacing		▪	▪	Distance entre l'image et le texte du bouton
Style		▪	▪	Détermine l'aspect du bouton

▲ **Remarques générales :**
Sous une apparente similitude il faut noter une grosse différence entre les TButtons et les TBitBtns, d'une part, et les TSpeedButtons, d'autre part : les premiers sont des composants dérivés de la classe TWinControl (ils peuvent donc acquérir le focus et sont gérés par un handle) alors que les TSpeedButtons dérivent de TGraphicControl (pas de focus possible, pas de handle).

Lorsque l'on hésite à choisir entre un TBitBtn et un TSpeedButton il faut se rappeler qu'un TSpeedButton est plus économe en ressources. Par contre le fait qu'il ne puisse pas obtenir le focus peut être un handicap (en particulier dans la création de boites de dialogue).

Le fait d'utiliser un '&' dans la propriété Caption d'un TButton ou d'un TBitBtn fait que le gestionnaire d'événement associé à son événement OnClick peut être activé par le raccourci clavier ainsi constitué.

7.4.2 Spécificités des composants TBitBtn

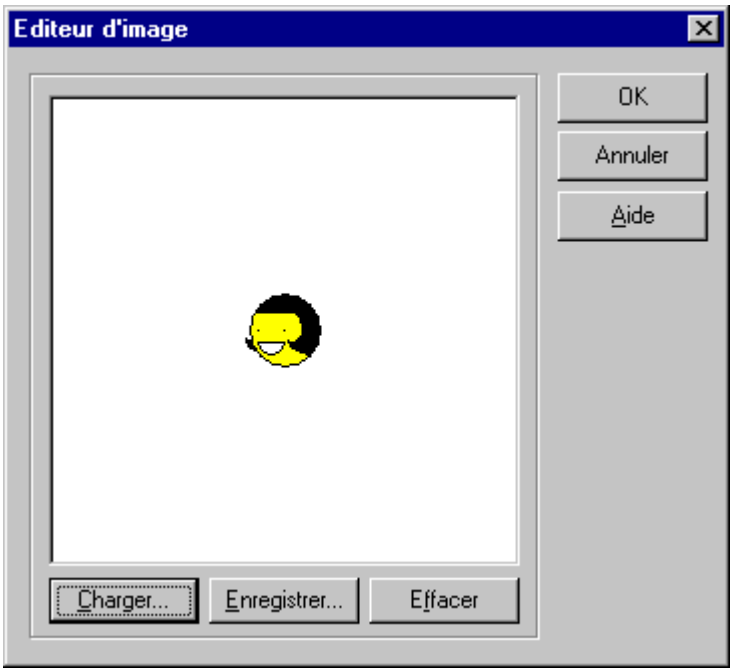
Le composant TBitBtn est un composant directement dérivé du composant TButton dont il reprend les principales caractéristiques. Il ajoute une petite image bitmap à coté de la légende du bouton. Cette image sert à repérer facilement l'action de chaque bouton.

Les propriétés spécifiques à ce type de bouton Kind et Glyph permettent de définir quelle image sera affichée. Ces deux propriétés fonctionnent ensemble : la propriété Kind permet de décider si l'image affichée sera d'un type prédéfini par le composant, ou une image quelconque, à partir d'un fichier image sur le disque.

Bien prendre en compte que le fait de choisir un type de bouton particulier fait que certaines actions sont réalisées automatiquement selon la valeur choisie (en particulier fermeture de la fenêtre ou de la boite de dialogue). Cela du fait que la valeur de la propriété ModalResult est initialisée à une autre valeur que mrNone, et que les propriété Cancel et Default sont, le cas échéant, positionnées.

Dans le cas où la propriété Kind := bkCustom (c'est à dire que l'on utilise un bouton qui n'est pas prédéfini, et qui n'a donc pas de comportement par défaut) il faut charger - grâce aux '...' qui apparaissent dans la propriété Glyph - l'image désirée.

L'éditeur d'image (qui n'est en fait qu'un chargeur / enregistreur). Il permet de charger une image (au format .BMP) de taille variée dans les composants adéquats (TBitBtn, TSpeedBtn, TImage)



Il existe aussi un autre éditeur d'image (qu'il ne faut pas confondre avec le précédent) et qui est accessible via le menu ' Outils | Editeur d'images '. Il s'agit cette fois-ci d'un puissant éditeur permettant de gérer les principales ressources d'un projet (bitmap, icônes, curseur, ressources, etc...). Il permet donc d'éditer les ressources existantes et de les modifier. Il est aussi possible d'en créer de nouvelles pour les inclure dans de nouveaux projets.

L'utilisation de cet éditeur permet en particulier de créer les petites images bitmap utilisées dans les BitBtn et les SpeedBtn. Il permet aussi de modifier l'icône par défaut proposée par Delphi afin que le programme en ait une de spécifique.

▲ **Utilisation de la propriété ModalResult :**

La propriété ModalResult (contenue dans les TButton et les TBitBtn) permet de spécifier des comportement par défaut à des boutons.

Par défaut cette propriété est initialisée à mrNone (= pas de comportement par défaut).

Si, par exemple, une fenêtre a été créée - en mode modal - avec deux boutons, 'OK' et 'Annuler', il faut initialiser la propriété ModalResult à mrOK pour le bouton 'OK' et à mrCancel pour le bouton 'Annuler'. Lorsque l'utilisateur choisit l'un ou l'autre de ces boutons, l'état modal de la boîte de dialogue s'achève car la valeur de ModalResult est supérieure à mrNone et la fenêtre disparaît. En utilisant ModalResult, il n'est pas nécessaire d'écrire un gestionnaire d'événement uniquement pour fermer la boîte de dialogue.

La valeur de ModalResult initialise la valeur de retour de la fonction ShowModal utilisée pour afficher la fenêtre. On peut donc tester cette valeur dans la fenêtre mère pour savoir quel bouton a été utilisé.

Lorsqu'on utilise un TBitbtn et que sa propriété Kind n'est pas bkCustom, la propriété ModalResult prend souvent une valeur différente de mrNone. Si l'on ne veut pas que le bouton ait le comportement prédéfini habituel, il faut repositionner la propriété à ModalResult := mrNone.

Il est toujours possible d'affecter une valeur quelconque à ModalResult (la valeur doit être assez grande de manière à ce qu'elle ne corresponde pas à une valeur prédéfinie). On peut alors tester cette valeur au retour dans la feuille mère.

7.4.3 Les TSpeedButton et les barres d'icônes

Les composants TSpeedButton sont surtout utilisés pour réaliser des barres d'icônes, permettant un accès rapide aux principales fonctionnalités proposées par un menu.

Grâce à ce composant, Delphi permet de réaliser facilement des barres d'icônes. Dans un premier temps elles peuvent être fixes mais il est possible d'imaginer des barres configurables et flottantes.

▲ **Réalisation d'une barre d'icônes :**
Pour réaliser ce type de barre il faut utiliser conjointement un composant TPanel (la plupart du temps aligné en haut de la fenêtre) et un nombre variable de composants TSpeedButton.

Les opérations à réaliser sont alors les suivantes :

- Positionnement et configuration du composant TPanel.
- Positionnement des différents composants TSpeedButton.

En règle générale, ces composants sont regroupés, de manière contiguë, par fonctionnalités. Un espace libre sépare les différents groupes.

Les images utilisées peuvent être chargées via la propriété Glyph et l'éditeur d'images.

Une fois cela réalisé il est possible de lier l'événement OnClick de chaque Speed Button à l'événement OnClick de l'élément de menu associé (réalisé auparavant).

Par exemple si l'on a un speed-button 'Ouvrir', en cliquant dans la zone de saisie de son événement OnClick on affiche la liste des événement de même nature gérée par la fiche. Il suffit alors de sélectionner l'événement correspondant à la rubrique 'Ouvrir' du menu pour qu'une action sur le bouton active le gestionnaire d'événement associé à la rubrique du menu.

▲ **Possibilités de configurations avancées :**

Groupement de Speed-boutons :
Les SpeedButtons peuvent être utilisés "groupés". De cette manière lorsque l'on clique sur un bouton du groupe, il reste enfoncé. Le fait de cliquer sur un autre bouton du groupe remet le premier bouton dans sa position initiale.

Pour cela il faut regrouper les boutons devant faire partie d'un groupe grâce à la propriété GroupIndex : il suffit de donner la même valeur (différente de 0) à tous ces boutons pour qu'ils soient "groupés".

Par exemple on peut regrouper 3 boutons avec la propriété GroupIndex := 1 puis deux autres avec la propriété à 2, etc… Il est judicieux que les boutons ainsi regroupés soient juxtaposés à l'écran et qu'un espace sépare les différents groupes.

Dans un ensemble de boutons groupés, le fait d'initialiser la propriété AllowAllUp := True fait qu'il est possible de n'avoir aucun bouton enfoncé.

Il est possible d'utiliser cette propriété pour un bouton unique (mais celui-ci doit avoir sa propriété GroupIndex différente de 0): cela permet de le laisser en position enfoncée lorsqu'il est activé. Il faut de nouveau cliquer dessus pour qu'il soit remis en position haute. Ainsi l'on peut créer une barre de boutons dont plusieurs peuvent avoir la position enfoncée.

7.5 Les composants permettant la saisie et l'affichage de texte

Delphi propose deux composants permettant la saisie de texte : le composant TEdit et le composant TMemo. D'autres composants peuvent aussi être utilisés pour cela mais ce n'est pas leur fonction principale (TComboBox).

7.5.1 Composant TEdit

Le composant TEdit présente à l'écran une zone de saisie qu'il est possible de configurer (taille, position, fonte, couleur). Il permet la saisie et l'affichage de texte (jusqu'à 255 caractères). Sa propriété principale est la propriété Text qui contient le texte affiché ou saisi.

▲ **Principales propriétés :**

Charcase	Permet de forcer les affichage en minuscules ou en majuscules
MaxLength	Indique le nombre maximal de caractères que l'on peut saisir. Si MaxLength := 0 il n'y a pas de contrôle de longueur.
Modified	Indique si le texte a été modifié ou non
PasswordChar	Permet de définir un caractère qui sera affiché en lieu et place des autres caractères lorsque l'on souhaite réaliser une saisie sécurisée.
ReadOnly	Permet de placer le composant en lecture seule.
Text	Texte affiché ou saisi.

La méthode Clear permet d'effacer le texte contenu dans le composant.

7.5.2 Le composant TMemo

Ce composant permet de réaliser des saisies ou des affichages multilignes. La taille du texte géré ne peut dépasser 32 Ko.

TMemo est surtout utilisé pour réaliser des éditeurs de texte simples.

▲ **Principales propriétés :**

Alignment	Aligne le texte au sein du composant
BorderStyle	Type de bordure
Lines	Permet de gérer les lignes du texte individuellement
MaxLenght	Longueur maximale du texte
Modified	Indique si le texte a été modifié ou non
ReadOnly	Place le composant en mode "lecture seule"
Scrollbars	Affiche ou non le ou les ascenseurs.
WantReturn	Détermine si les retour chariot sont acceptés
WordWrap	Autorise ou non le retour à la ligne automatique en fin de ligne

Chaque ligne du composant est gérée (via la propriété Lines) comme un objet particulier ayant ses propres propriétés et méthodes.

On peut, en particulier charger un fichier texte :

```
procedure TForm1.BOuvrirClick (Sender : TObject)
begin
      if OpenDialog.Execute then
       Memo.Lines.LoadfromFile (OpenDialog.Filename);
end ;
```

{ Le fichier sélectionné grâce à la boite de dialogue OpenDialog est chargé (s'il contient du texte) en une seule opération. }

▲ **Principales méthodes :**

Clear	Efface tout le texte
CopyToClipBoard **CutToClipBoard** **PasteFromClipBoard**	Envoi ou réception de texte vers ou en provenance du presse papiers.

D'autres méthodes permettent de sélectionner et de manipuler une partie du texte.

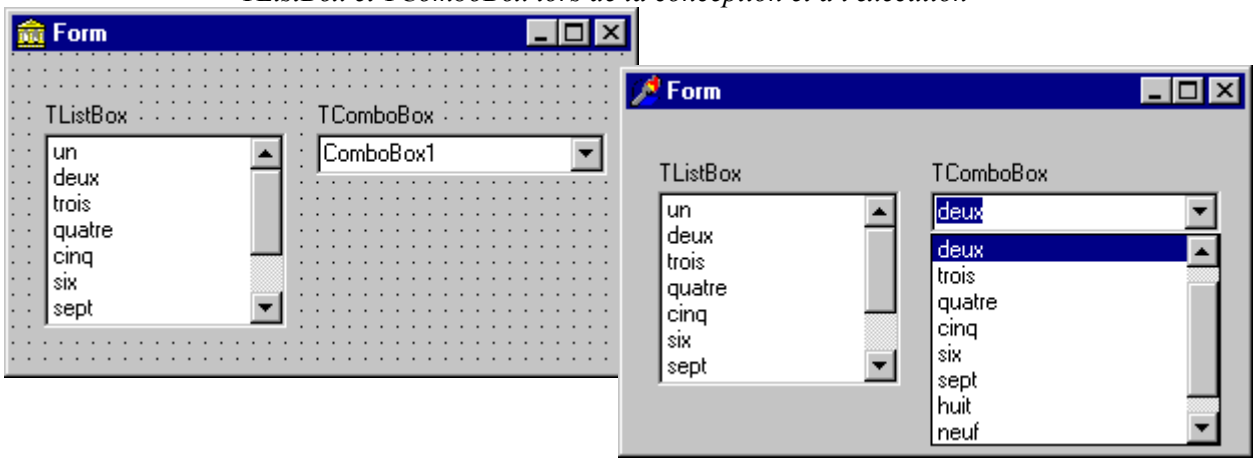
7.6 Les composants de type "boite de liste"

Delphi propose deux composants distincts pour réaliser des listes d'éléments contenues dans les boites :
Le composant TListBox qui affiche l'ensemble des éléments d'une liste afin que l'on puisse faire un choix ;
Le composant TComboBox qui ne présente, au repos, qu'une zone de saisie mais qui se déploie et affiche l'ensemble des choix lorsqu'on clique sur la flèche située à sa droite.

La propriété principale de ces deux composants est la propriété Items, c'est à dire l'ensemble des éléments, de type TStrings, composant les "lignes" de la liste.

C'est grâce à cette propriété que l'on peut gérer la liste (méthodes Add(), Delete,() Insert() , etc...).

TListBox et TComboBox lors de la conception et à l'exécution



▲ **Principales propriétés :**

BorderStyle	Style de la bordure
Canvas	Permet d'afficher des graphiques dans la liste
ItemIndex	Indique la ligne actuellement sélectionnée (surbrillance)
Items [i]	Permet de gérer chaque entrée de la liste de manière particulière.
MultiSelect	Autorise les multiples sélections.
Selected [i]	Indique si l'entrée est sélectionnée
SelectedCount	Nombre d'entrées sélectionnées
Sorted	Affiche la liste triée ou non

La propriété Count permet de connaître le nombre d'éléments de la liste.

La propriété ItemIndex permet de connaître la position de l'élément sélectionné (= -1 si aucun élément est sélectionné, 0 correspond au premier élément).

Pour une ComboBox la propriété Text permet, en plus, d'accéder à l'élément placé dans la zone de saisie (après sélection d'un élément de la liste ou par initialisation interne).

Les ComboBox sont surtout utilisées pour proposer à l'utilisateur des listes finies de paramètres.

▲ **Principales manipulations :**

Lors de la phase de conception il est possible d'initialiser un composant TListBox ou un composant TComboBox en accédant à un éditeur de liste de chaînes qui apparaît lorsque l'on active la propriété Items dans l'inspecteur d'objet.

On utilise cette possibilité lorsque l'on souhaite proposer à l'utilisateur une liste finie de choix.

On peut charger directement un fichier texte (puis le sauvegarder) en utilisant les méthodes LoadFromFile() / SaveToFile() de la propriété Items. Les différents retour chariot contenus dans le fichier texte tiennent lieu de délimiteurs d'items.

On ajoute ou on supprime des entrées dans les listes en utilisant les propriétés Add() ou Delete() de la propriété Items.

```
ListBox.Items.Add (Edit1.Text); {Ajout du contenu de la zone
d'édition Edit1 dans la liste}
ListBox.Items.Delete (ListBox.ItemIndex); {Suppression de la ligne
qui est sélectionnée }
```

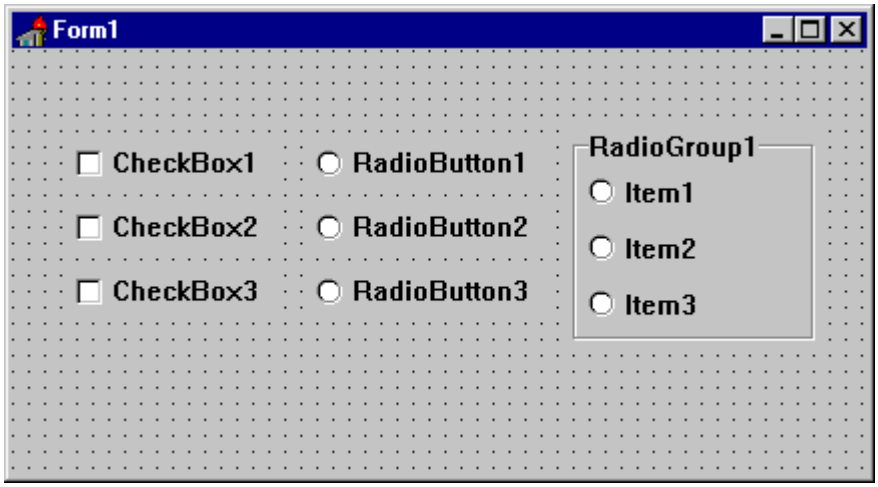
Il est possible d'empêcher l'insertion de lignes déjà existantes on peut utiliser la méthode IndexOf de l'objet Items. Cette fonction renvoie l'indice de l'item correspondant à la chaîne passée en paramètre si celle-ci existe déjà dans la liste et -1 dans le cas contraire.

```
If (ListBox.Items.IndexOf (Edit.Text) = -1 then
      ListBox.Items.Add (Edit.Text)
else
      MessageDlg (' L''élément existe déjà !!!' , mtWarning,  [mbCancel],0);
```

7.7 Présentation d'options

Delphi propose quatre composants permettant de proposer des options à l'utilisateur : Deux sont des composants "simples" : les "cases à cocher" (TCheckBox) et les "boutons radio" (TRadioButton). Les deux autres permettent de grouper des composants : TGroupBox permet de regrouper toutes sortes de composants et TRadioGroup ne regroupe que des boutons radios afin de faciliter leur gestion .

Cases à cocher, boutons radio et groupe de boutons radio



7.7.1 Les cases à cocher

Une case à cocher (TCheckBox) permet de présenter une option à l'utilisateur. Ce dernier peut la cocher pour sélectionner l'option ou supprimer la coche pour la désélectionner.

Les différentes cases à cocher ont un comportement indépendant (il peut y avoir plusieurs cases cochées).

C'est la propriété Checked (ou la propriété State) qui indique si la case est cochée ou non : lorsque l'utilisateur coche ou décoche une case à cocher, la valeur de la propriété Checked change et l'événement OnClick se produit.

Il est possible de désactiver une case à cocher en mettant la propriété Enabled à True.

7.7.2 Les boutons radio

Comme les cases à cocher, les boutons radios (TRadioButton) permettent de présenter un ensemble d'options. Mais cette fois-ci, les options s'excluent mutuellement (ce qui veut dire qu'un seul bouton radio peut être validé en même temps).

C'est la propriété Checked (ou la propriété State) qui permet de modifier l'apparence d'un bouton radio (sélectionné ou non). Un bouton radio peut être désactivé via la propriété Enabled.

En fait tout le problème vient du fait qu'il faut coordonner les états des différents boutons dès lors que l'on en modifie un. Le composant ne comporte pas de propriété ou de méthode permettant de gérer ces modifications.

Le fait de placer les différents boutons dans un composant TPanel ou un composant GroupBox permet , via certaines propriétés de ces derniers, ce type de gestion mais cela n'est pas très facile à réaliser. L'utilisation du composant TRadioGroup facilite nettement cette gestion.

7.7.3 La boite "groupe radio"

Le composant TRadioGroup simplifie la synchronisation entre les différents boutons radio.

A l'exécution, lorsque l'utilisateur sélectionne un bouton radio dans la boîte groupe, le bouton qui l'était précédemment est automatiquement désélectionné.

A la conception, les boutons radio sont ajoutés à la boîte via la propriété Items qui appelle l'éditeur de chaînes de caractères (le même que celui utilisé dans TMemo, TListBox ou TComboBox): chaque chaîne de la propriété Items provoque l'affichage d'un bouton radio dans la boîte groupe, avec la chaîne comme intitulé du bouton correspondant.

On peut ajouter ou supprimer dynamiquement des boutons radios en utilisant les possibilités offertes par la propriété Items.

La propriété ItemIndex détermine le bouton radio actuellement sélectionné.

Il est possible d'afficher les boutons radio sur une ou plusieurs colonnes en définissant la valeur de la propriété Columns.

On accède à un élément particulier de la boite par la ligne :

```
RadioGroup.Items [ i ]
```

L'élément sélectionné est accessible via l'expression :

```
RadioGroup.Items [ RadioGroup.ItemIndex ]
```

TRadioGroup (comme TGroupBox) est un composant "container" : il en possède toutes les caractéristiques (diffusions de certaines de ses propriétés).

7.8 Les boîtes de messages

Il est toujours possible de créer une fenêtre spécifique pour afficher un message ou poser une question à l'utilisateur.

Mais cette solution est lourde est consommatrice en ressources. Il est préférable, dans de nombreux cas, d'utiliser directement des possibilités proposées par Windows permettant la création rapide de boîte de messages.

Les boîtes de messages font partie de l'unité Dialogs et fonctionnent toutes en mode modal. Elles sont très pratiques pour afficher diverses informations, avertissements et messages d'erreur ou pour demander à l'utilisateur la conduite à tenir face à une alternative.

Pour accéder à une boîte de message il suffit d'invoquer une procédure particulière qui réalisera la totalité des opérations d'affichage.

7.8.1 Procédure ShowMessage

Le prototype est : ShowMessage (const Msg : string)

Il s'agit de la boîte de message la plus simple. Elle affiche une boite centrée dans la fenêtre comportant le texte passé en paramètre et un bouton 'OK' pour fermer la fenêtre.

Son défaut majeur c'est qu'elle affiche le nom du programme dans sa barre de titre au lieu d'un titre plus lisible.

La fenêtre est automatiquement dimensionnée à la taille de la chaîne à afficher (avec éventuellement affichage sur plusieurs lignes).

Ne pas oublier que l'on peut invoquer aussi directement la procédure d'affichage de Windows en appelant la méthode MessageBox de l'objet TApplication. Les possibilités de configuration de cette méthode sont plus importantes que celles de ShowMessage par contre l'affichage se fait dans le "style" Windows, c'est à dire avec le fond en blanc.

7.8.2 Fonction MessageDlg

Le prototype est : MessageDlg (const Msg : string ; AType : TMsgDlgType ; AButtons : TMsgDlGButtons ; HelpCtx : LongInt): Word ;

Cette boîte de message est plus complexe car :

- Elle affiche un message d'avertissement avec éventuellement une image prédéfinie ;
- Elle affiche un nombre variable de boutons proposant un choix à l'utilisateur.

La valeur renvoyée par la fonction indique le choix réalisé par l'utilisateur.

Les différents paramètres de la fonction permettent de configurer son apparition à l'écran:

Msg : Message à afficher .

AType : Détermine l'icône qui apparaîtra à coté du message. Les différentes valeurs possibles sont :

mtWarning	Une boîte de message contenant une icône point d'exclamation jaune.
mtError	Une boîte de message contenant une icône de stop rouge.
MtInformation.	Une boîte de message contenant une icône "i" bleu
mtConfirmation	Une boîte de message contenant une icône point d'interrogation vert.
mtCustom	Une boîte de message ne contenant pas d'image. Le titre de la boîte de dialogue est le nom du fichier exécutable de l'application.

AButtons : Détermine le ou les boutons qui apparaîtront dans la boîte de message.
Le ou les boutons doivent être indiqués entre crochet [....]. Les différentes valeurs possibles sont :

mbYes	Un bouton avec une marque verte et le texte 'Oui'.
mbNo	Un bouton avec un cercle rouge barré et le texte 'Non'.
mbOK	Un bouton avec une marque verte et le texte 'OK'.
mbCancel	Un bouton avec un X rouge et le texte 'Annuler'.
mbHelp	Un bouton avec un '?' cyan et le texte 'Aide'
mbAbort	Un bouton avec une marque rouge et le texte 'Abandonner'.
mbRetry	Un bouton avec deux flèches circulaires vertes et le texte 'Réessayer'.
mbIgnore	Un bouton avec un bonhomme vert marchant et le texte 'Ignorer'.
mbAll	Un bouton avec une double marque verte et le texte 'Tous'.

Outre les valeurs individuelles de l'ensemble, il existe aussi trois ensembles constants prédéfinis correspondant aux combinaisons les plus courantes :

mbYesNoCancel	Un ensemble plaçant les boutons Oui, Non et Annuler.
mbOkCancel	Un ensemble plaçant les boutons OK et Annuler.
mbAbortRetryIgnore	Un ensemble plaçant les boutons Abandonner, Réessayer et Ignorer.

HlpCtx : Permet de lier la boîte avec une aide en ligne. Paramètre facultatif (initialisé à 0).

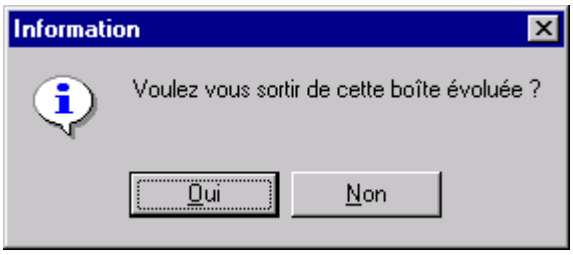
La valeur renvoyée peut prendre les valeurs suivantes : mrNone, mrOk, mrCancel, mrAbort, mrRetry, mrIgnore, mrYes, mrNo, mrAbort, mrRetry, mrIgnore, mrAll.

La fonction MessageDlgPos () est pratiquement identique mais permet de positionner la boite de dialogue à l'écran.

Exemple d'utilisation :

```
procedure TForm1.BitBtn1Click (Sender: TObject );
begin
    if MessageDlg('Voulez vous sortir de cette boîte évoluée ?', mtInformation, [mbYes, mbNo],
        0) = mrYes then
    begin
        MessageDlg ('Au revoir', mtInformation, [mbOk], 0);
        Close;
    end;
end ;
```

Provoque l'apparition de la boîte de dialogue :



7.8.3 Boites de saisie

Delphi propose aussi deux boites permettant la saisie d'une chaîne de caractères par l'utilisateur.

function InputBox (const ACaption , APrompt, ADefault : string): string

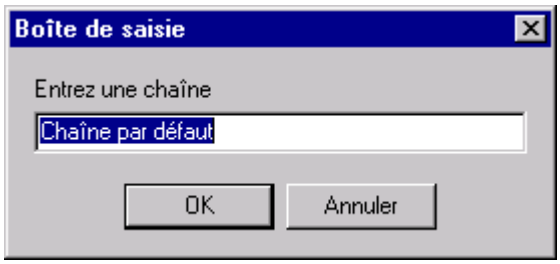
Le paramètre ACaption indique le titre de la boîte de dialogue.
Le paramètre APrompt est le message situé au dessus de la zone de saisie.
Le paramètre ADefault est la chaîne apparaissant dans la zone de saisie à l'ouverture de la boîte de dialogue (fréquemment : ' ' , chaîne vide).

Si l'utilisateur choisit le bouton 'Annuler', la chaîne par défaut est renvoyée. Si l'utilisateur choisit le bouton 'OK' , la valeur de la chaîne entrée dans la zone de saisie est renvoyée.

Exemple :

```
procedure TForm1.BitBtn1Click (Sender: TObject);
var
      ChaineEntree : String ;
begin

      ChaineEntree := InputBox ('Boîte de saisie',
'Entrez une chaîne',
                        'Chaîne par défaut');
end ;
```

Affiche la boite de saisie suivante :



function InputQuery (const ACaption, APrompt: string; var Value: string): Boolean ;

Même si Value remplace Adefault le rôle des trois paramètres est identique à celui qu'ils ont dans la fonction InputBox().

Par contre, lorsque l'utilisateur entre une chaîne dans la zone de saisie puis choisit 'OK', le paramètre Value prend la nouvelle valeur (d'où le passage de paramètre par variable).

La fonction InputQuery renvoie True si l'utilisateur a choisi 'OK' et False si l'utilisateur a choisi 'Annuler' ou a appuyé sur la touche Echap.

7.9 Accès aux boites de dialogue de Windows

7.9.1 Principe d'utilisation

Il est possible d'utiliser directement, au sein d'une application Delphi les boites de dialogues standards de Windows (contenues dans COMMDLG.DLL).

Cette possibilité évite d'avoir à recréer ce genre de dialogue. Les informations validées par l'utilisateur sont ensuite directement utilisables dans l'application Delphi.

Les différentes boîtes sont accessibles via des composants regroupés dans l'onglet 'Dialogues' de la palette de composants. Il s'agit de :

TOpenDialog	permet d'ouvrir un fichier, après avoir éventuellement modifié le lecteur,le répertoire, les sous-répertoires, etc ....
TSaveDialog	permet de sauvegarder un fichier.
TFontDialog	permet de modifier un texte (polices, tailles, couleurs et styles).
TColorDialog	permet de modifier les couleurs.
TPrintDialog	permet d'imprimer un document après configuration de l'impression.
TPrinterSetupdialog	permet de configurer une imprimante.
TFindDialog	permet de rechercher un texte.
TReplaceDialog	permet de remplacer un texte.

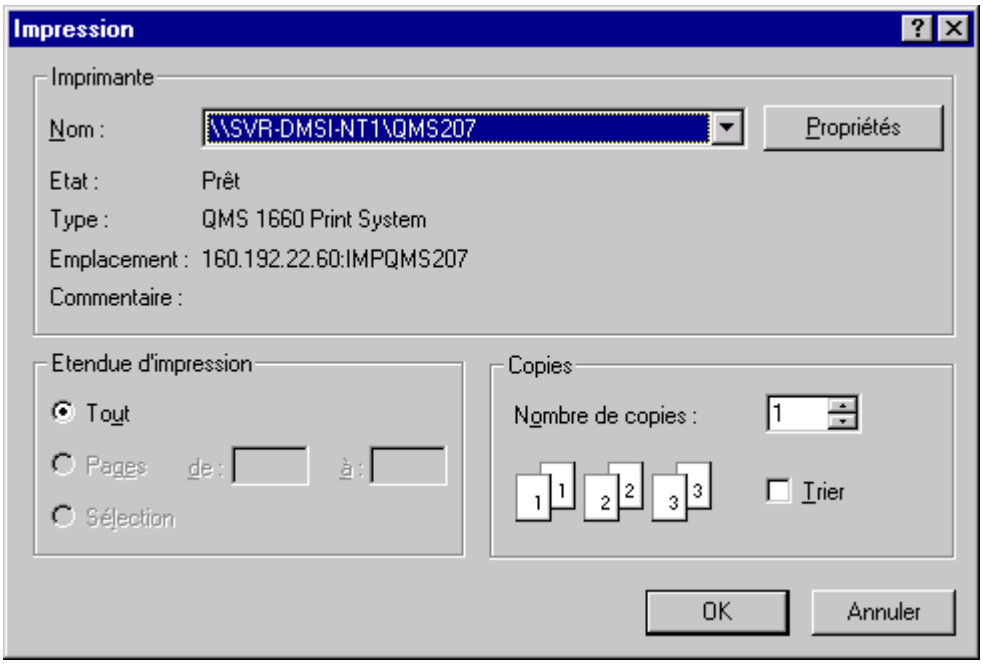
Toutes ces boites de dialogues sont accessibles en :

- Déposant le composant concerné (non visuel) sur la fiche.
- En l'exécutant à partir d'un gestionnaire d'événement (click sur un bouton ou à partir d'un menu la plupart du temps).

Par exemple le gestionnaire d'événement :

```
procedure TForm1.ImprimerClick (Sender: TObject);
begin
      PrintDialog1.Execute ;
end;
```

affichera la boîte de dialogue suivante :



La fonction Execute renvoie un booléen. Il est donc possible de vérifier la bonne exécution de l'appel de la boite de dialogue :

```
If TOpenDialog.Execute then
      < actions ...>
```

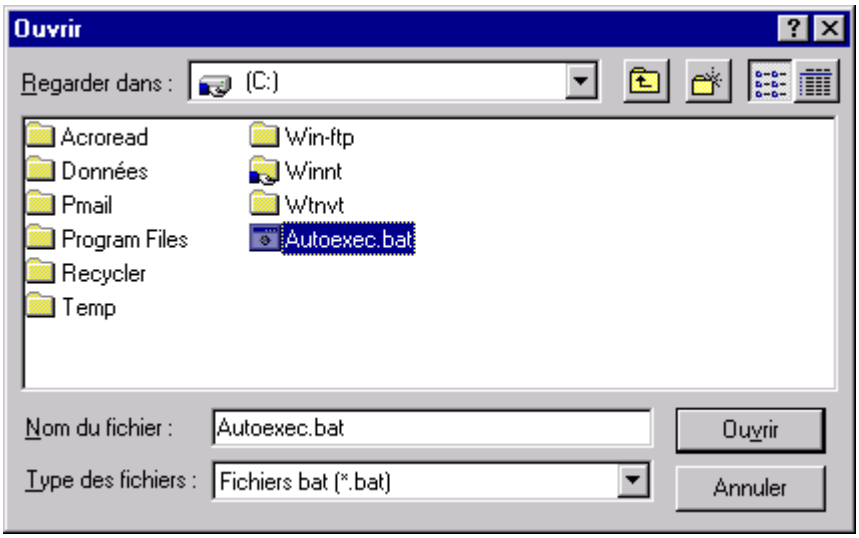
7.9.2 La boite de dialogue "OpenDialog"

Cette boite de dialogue est la plus utilisée et celle qui demande le plus d'attention. Elle permet à l'utilisateur de sélectionner un fichier situé dans n'importe quel répertoire (y compris sur un autre disque).

On peut configurer cette boîte de dialogue dès la conception, pour qu'elle affiche certains renseignements, grâce aux propriétés suivantes :

DefaultExt permet d'indiquer l'extension qui sera attribuée par défaut aux fichiers si celle-ci n'est pas explicitée.
Filename contient le nom du fichier qui apparaît dans la zone de saisie après sélection.
Files, accessible uniquement à l'exécution, contient la liste de tous les fichiers affichés dans la boite de liste, après éventuellement un filtrage.
Filter indique, grâce à une boite de dialogue spécifique, les différents masques d'extension pouvant être proposés à l'utilisateur.

InitialDir indique le répertoire par défaut à l'ouverture de la boite de dialogue.



Une boite OpenDialog configurée : des masques ont été définis (propriété Filter) ainsi qu'une extension par défaut et un répertoire initial.

C'est la valeur de la propriété FileName qui doit être utilisée dans le reste du programme pour réaliser les opérations d'ouverture du fichier.

7.9.3 Utilisation de composants système

Dans certains cas, l'on ne veut pas utiliser la boite de dialogue TOpenDialog (ou la boite TSaveDialog) dans son intégralité. Il est alors possible d'utiliser certains composants, regroupés dans l'onglet 'Système' de la palette de composants: TFileListBox, TDirectoryListBox et TDriveComboBox.

Ces composants permettent de manipuler les fichiers sur disque afin d'en obtenir des listes.

Par exemple en utilisant les filtres appropriés ainsi qu'en sélectionnant le répertoire on peut proposer à l'utilisateur un choix de fichiers à sélectionner sans qu'il ait la possibilité "d'aller chercher ailleurs".

Souvent ces composants sont utilisés avec la propriété Visible = False de manière à disposer de la liste de fichiers sans qu'elle apparaisse à l'écran (par exemple pour initialiser une ListBox ou une ComboBox).

7.10 Ajout de composants

Il est possible d'ajouter des composants dans la palette proposée par Delphi. Ces composants peuvent être des objets VCL créés par le programmeur, des objets VCL acquis auprès de sociétés tierces ou de contrôles VBX provenant du monde Visual Basic.

Pour ajouter un élément il suffit de fournir le chemin du composant à installer grâce au menu ' Options | Installer des composants ' . Lorsque cela est fait, Delphi recompile automatiquement la bibliothèque de composants pour y incorporer le nouveau composant.

S'il s'agit d'un contrôle VBX, il est systématiquement installé dans l'onglet 'VBX' de la palette.

Certains contrôles VBX , écrits avec Visual Basic 3.0 sont incompatibles avec Delphi.

Il est possible, par le même menu, d'enlever des composants à la palette.

Lorsqu'une application utilise des contrôles VBX et qu'elle est distribuée à un tiers, il faut fournir le fichier 'bivbx11.dll' et l'installer dans le répertoire /Windows/System.

8 Programmation avancée

Les possibilités de programmation vues dans le chapitre précédent suffisent la plupart du temps lorsque l'on souhaite réaliser une application efficace. Néanmoins il est parfois nécessaire de faire appel à d'autres techniques pour résoudre certains cas.

8.1 Variables et programmation par objets

8.1.1 Utilisation des propriétés en tant que variables

Un objet est composé à la fois de propriétés et de méthodes. Les propriétés peuvent être assimilées à des données, donc à des variables, les méthodes à des fonctions ou à des procédures. Un langage implémentant les caractéristiques permettant de manipuler des objets met en oeuvre, d'une manière plus ou moins stricte, trois principe fondamentaux :

- **L'encapsulation** des données : ce qui implique que l'on ne peut accéder à celles-ci qu'en invoquant les méthodes qui les manipulent (les références directes aux données sont interdites).
- **L'héritage** : il permet de "dériver" une classe à partir d'une classe déjà constituée. La classe dérivée reprend les caractéristiques de la classe parente et en rajoute quelques unes.
- **Le polymorphisme** : il permet de nommer des méthodes ayant le même type de comportement avec le même nom quel que soit l'objet sur lequel elles s'appliquent.

Delphi, via le Pascal Objet, implémente pratiquement tous ces mécanismes. Seul le principe d'encapsulation n'est pas pleinement respecté (suivant en cela le langage C++).

Implémentation de la notion d'encapsulation (P.O.O.) dans le Pascal Objet :

Le Pascal Objet ne respecte pas pleinement les prescriptions "objets" qui veulent que l'on ne puisse pas accéder directement à une donnée déclarée dans un objet. Selon le cas la donnée sera accessible directement ou non.

Pour cela, lors de la définition d'une classe, il est possible de faire figurer la déclaration d'une variable (ou d'une méthode), dans l'une des catégories suivantes :

Public	Les propriétés (= variables) sont accessibles sans restriction aucune, c'est à dire aussi bien au sein de l'objet créé, à partir des méthodes de la classe, qu'à partir d'une méthode ou d'une procédure située en dehors.
Private	Les propriétés ne sont accessibles qu'au sein de l'objet créé à partir de la classe. Mais si un tel objet est déclaré, alors les propriétés définies dans sa partie Private ne seront pas accessibles aux objets créés à partir d'une classe dérivée de la classe mère.
Protected	Comme pour la partie Private, les propriétés ne sont accessibles qu'au sein de l'objet. En revanche lors d'un héritage, toutes les propriétés définies dans la partie Protected seront accessibles aux objets dérivés, mais ceci toujours au sein de l'objet uniquement..

- En d'autres termes, la partie Protected est une possibilité de contrôle d'accès intermédiaire entre les déclarations Public et Private

De manière concrète, la déclaration d'une propriété au sein d'un objet se fait de la manière suivante :

```
Type
        MaClasse = Class
        Public                  { PARTIE PUBLIQUE }
                VarPublique : Real ;
        Private                 { PARTIE PRIVEE }
                VarPrivee : Real ;
        End ;
```

L'accès à ces propriétés diffère de l'endroit à partir duquel l'on tente de les atteindre.

- Ainsi, au sein de l'objet lui-même, l'accès aux propriétés, de quelque catégorie qu'elle soit, est autorisé :

```
    Procedure MaClasse.UneMethode ;
Begin
      VarPublique := 123 ;
      VarPrivee := 456 ;
End;
```

En dehors de l'objet, les possibilités sont différentes :

```
    Procedure AutreClasse. UneMethode ;
    Var
        uneInstance : MaClasse ;
    begin
        uneInstance := MaClasse . Create ;
        { Création d'un objet ' uneInstance' instance de la
          classe MaClasse }

        uneInstance . VarPublique := 123 ; { Modification de la
variable publique }
    End ;
```

Par contre écrire : uneInstance. VarPrivee := 'Variable privée' générera une erreur lors de la compilation car cette propriété est privée.

Si l'on veut pouvoir en modifier le contenu il faut créer une méthode permettant d'y accéder.

Par exemple :

```
        uneInstance.ChgeValVarPriv(456) ;
```

▲ **Déclaration d'une variable en tant que propriété ou une procédure en tant que méthode :**

On remarque que lorsque l'on crée une feuille au sein d'un projet Delphi, cette feuille est une classe particulière, constituée à partir de la classe TForm, qui sera modifiée en permanence au fur et à mesure que l'on y ajoute des composants.

```
type
      TForm1 = class(TForm)
      Button1: TButton;
      Memo1: TMemo;
private
{ Déclarations Private }
public
{ Déclarations Public }
end;

{ Exemple de code généré automatiquement par Delphi pour créer une classe particulière à partir de la classe de base TForm }
```

Il est donc possible d'utiliser cette possibilité pour ajouter des "variables" dans une ou l'autre des parties de la définition de la classe ainsi définie (en fonction de la visibilité que l'on souhaite donner à cette variable): selon le cas ces variables seront considérées comme des propriétés publiques ou privées et éventuellement "protected").

Il en est de même pour les procédures (ou pour les fonctions) que l'on peut déclarer au sein de la classe selon les mêmes critères.

8.1.2 Visibilité d'une variable

On a vu qu'une variable est utilisable au sein même du bloc (et de ses sous-blocs, s'il y en a) où elle a été définie. Cependant, si dans un sous-bloc, une variable de même nom est définie, alors cette dernière est prioritaire et cache la visibilité de la variable déclarée à un niveau supérieur.

▲ **Référencement à partir d'une autre unité :**

On a vu aussi que les variables étaient déclarées au niveau d'une unité (qui représente le bloc de plus haut niveau). Néanmoins, il peut arriver qu'il soit nécessaire d'accéder à une variable ayant été déclarée dans une autre unité.

Ceci se fera donc de la manière suivante :

Si l'unité Unit1 nécessite l'utilisation de variables déclarées dans une autre unité, appelée Unit2, alors il faut ajouter le nom de cette dernière dans la déclaration uses de Unit1 :

```
Unit Unit1;                    ou      Unit Unit1
Interface                              Interface
    uses Unit2;                            ...
Implementation                         Implementation
    ...                                    uses Unit2
```

Dans le premier cas, les unités utilisant (par une autre clause 'uses' ...) l'unité Unit1 ont l'accès aux données et méthode de l'unité Unit2.

Dans le deuxième cas, Unit1 a bien accès aux données et méthodes de Unit2 mais les unités référençant Unit1 n'ont pas cette visibilité.

A partir de cette déclaration il est possible de référencer une propriété, une méthode ou une variable appartenant à l'unité Unit2 de n'importe quel point de l'unité Unit1 mais en faisant référence explicitement à Unit2 lorsque l'on utilise la variable référencée :

```
...
Edit1.Text := Unit2.Edit1.Text ;
Unit2.Edit1.Clear ;
...
```

On remarque que deux composants peuvent avoir le même nom dans deux unités différentes (alors que ce n'est pas possible au sein d'une même unité).

▲ Référencement mutuel de deux fiches :

Un cas un peu plus complexe consiste à avoir à référencer les propriétés d'une fiche à partir d'une autre, et réciproquement (= modifier les propriétés de la fiche A à partir de la fiche B, et vice et versa).

Pour ce faire, trois solutions existent :

- Il est possible de déclarer l'unité B dans la section interface de l'unité A et l'unité A dans la section implementation de l'unité B.
- Il est possible de déclarer l'unité B dans la section implementation de l'unité A et l'unité A dans la section interface de l'unité B.
- Il est possible de déclarer l'unité B dans la section implementation de l'unité A ainsi que l'unité dans la section implementation de l'unité B.

1° cas	**2° cas**	**1° cas**
Unit UnitA ; Interface uses UnitB , ... Implementation ...	Unit UnitA ; Interface ... Implementation uses UnitB ...	Unit UnitA ; Interface ... Implementation uses UnitB , ...
Unit UnitB ; Interface ... Implementation uses UnitA	Unit UnitB ; Interface uses UnitA; Implementation ...	Unit UnitB ; Interface ... Implementation uses UnitA

Par contre la déclaration de l'unité B dans la section interface de l'unité A et de l'unité A dans la section interface de l'unité B aboutit à une erreur pour cause de "référence circulaire" (Delphi signale une erreur à la compilation).

8.1.3 Passage de paramètre entre fiches liées (mère - fille)

En toute rigueur il est aussi possible de passer la valeur d'un paramètre entre deux fiches. Pour cela il suffit de déclarer la variable dans la section interface de l'unité "fille" et de l'initialiser dans la fiche mère juste avant de créer la fiche fille :

Dans la section interface de la fiche fille on a :

```
var
    FeuilleFille: TFeuilleFille;
    chaine : string[80];
```

Dans la fiche mère :

On peut créer la fiche fille à partir d'un click sur un bouton (ne pas oublier d'insérer l'unité correspondante FFille dans la clause uses).

Le code correspondant à la fonction événement associée est alors :

```
procedure TFeuilleMere.Button1Click (Sender: TObject);
begin
    chaine := 'Bonjour tous ' ;
    { initialisation de la variable déclarée dans la fiche fille }
    FeuilleFille := TFeuilleFille.Create (Self );
    { création de la fiche }
    try
        FeuilleFille.ShowModal ;
    finally
        FeuilleFille.Free ;
    end ;
    Label1.Caption := chaine ;
end;
```

Pour que le paramètre soit visualisé dans une zone de saisie on a, dans la feuille fille, le code suivant :

```
procedure TFeuilleFille.FormCreate (Sender: TObject);
begin
    Edit1.Text := chaine ;
end;
```

De même si l'on souhaite récupérer la nouvelle valeur de la zone de saisie (si elle a été modifiée) dans la feuille mère, on a le code suivant :

```
procedure TFeuilleFille.FormClose (Sender: TObject; var Action: TCloseAction);
    begin
        chaine := Edit1.Text ;
    end;
```

Si l'identificateur est déjà utilisé dans la fiche où il n'est pas déclaré il faut le préfixer du nom de sa fiche d'origine :

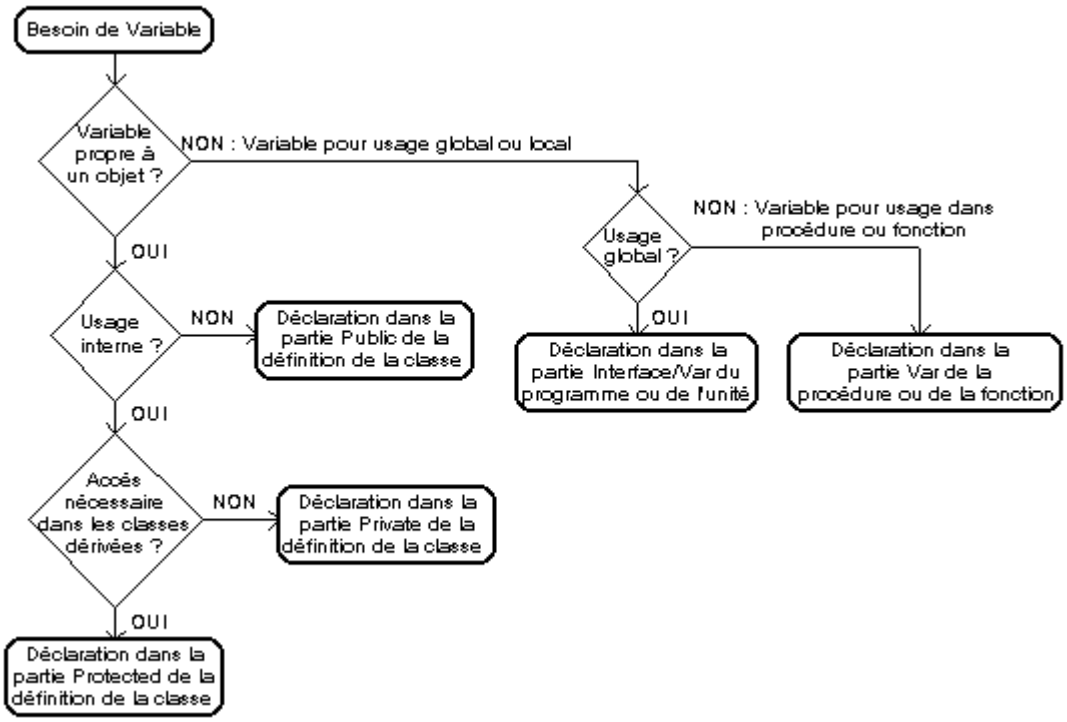
```
Form1.chaine := 'Bonjour tous';
```

Cette possibilité est simple à mettre en oeuvre mais elle va à l'encontre de la philosophie "objet".

8.1.4 Utilisation optimale des variables

Lorsqu'on doit utiliser une variable, on doit se poser certaines questions. de manière à ne pas pénaliser les performances globales de l'application.

Pour ce faire, il est possible de s'aider du schéma suivant :



Dans certains cas il est possible de créer une unité spécifique ne contenant que les variables globales. Cette unité sera référencée dans les différentes clauses 'uses' des unités qui en ont besoin.

8.1.5 Paramètres 'Sender' et 'Source'

Delphi propose deux paramètres prédéfinis qu'il utilise fréquemment dans les codes qu'il génère automatiquement mais qu'il est aussi possible d'utiliser en programmation : le paramètre Sender et le paramètre Source.

Ces deux paramètres sont de type TObject qui est la classe située en haut de la hiérarchie des objets Delphi. Ce qui fait que n'importe quel objet utilisé peut être considéré comme étant de type TObject.

- Sender représente le contrôle ou le composant qui répond à l'événement qui s'est produit.

On remarque que Sender fait partie des paramètres de pratiquement toutes les fonctions événements.

- Source est du même type mais est utilisé quand l'événement s'applique à l'objet lui-même.

On peut utiliser ces deux paramètres pour réaliser certains contrôles :

```
if Sender is TEdit then
 { Actions ... }
```

On peut réaliser un contrôle plus précis :

```
if Sender = Button1 then
      { Actions ... }
```

▲ Gestion de groupement de composants :

Il est possible d'avoir à gérer des groupements de composants identiques (par exemple des batteries de boutons). Le paramètre Sender permet de gérer aisément un événement partagé par tous ces composants.

Exemple :

Une fiche contient 3 boutons. Chacun a un nom et sa propriété Tag est initialisée à 1, 2 ou 3.

Le gestionnaire d'événement associé au click du premier bouton est partagé avec les autres boutons. Son code est :

```
procedure TForm1.Button1Click (Sender: TObject);
begin
      case TButton (sender).Tag of
            1: ShowMessage('1° bouton pressé ' );
            2: ShowMessage('2° bouton pressé ' );
            3: ShowMessage('3° bouton pressé ' );
      end;
end;

{ Chaque click sur un bouton générera le message approprié }
```

▲ L'opérateur 'is' :

L'opérateur is sert à effectuer des vérifications de type de manière dynamique. Grâce à lui il est possible de vérifier que le type réel d'une référence objet appartient bien à une classe précise.

La syntaxe de l'opérateur is est la suivante :

```
ObjectRef is ClassRef
```

Exemple :

```
If MonObjet is TButton then
    ...
```

Il est possible de réaliser un transtypage sur le paramètre Sender :

```
TButton (Sender) { considère l'objet comme un bouton }
```

ou mieux :

```
With Sender as TButton do
begin
    { Actions ..... }
end ;
```

▲ Paramètre 'self ' :

Self est un paramètre implicite transmis à chaque fois qu'il est fait appel à une méthode. Self est un pointeur sur l'instance utilisée pour appeler la méthode.

Ce paramètre garantit, entre autres, que les méthodes virtuelles correctes seront appelées pour une instance objet particulière et que les données d'instance correctes seront utilisées par la méthode objet.

8.2 Gestion des exceptions

La complexité d'une application fonctionnant en mode événementiel dans un environnement graphique ne permet pas de pouvoir prévoir, de manière algorithmique, toutes les causes d'erreurs.

Delphi, comme la plupart des environnements de développement modernes, signale les conditions d'erreur par des exceptions. Une exception est un objet contenant des informations indiquant quelle erreur (matérielle ou logicielle) s'est produite et où elle s'est produite.

Pour que les applications soient robustes leur code doit reconnaître les exceptions lorsque ces dernières se produisent et y répondre (sous peine de voir s'afficher un message d'erreur généralement peu explicite). On gère les exceptions, c'est à dire qu'on est capable de les détecter afin d'intervenir sur le déroulement du programme, lorsque l'on souhaite :

- protéger certains blocs d'instructions ;
- protéger les allocations de ressources systèmes ;

Il est possible de traiter les exceptions courantes ou de définir ses propres exceptions.

Cette gestion était déjà réalisable en Pascal, mais cela impliquait l'usage de directive telle que {$I+}/ {$I-} (pour capter les problèmes potentiels d'entrée/sortie).

La gestion des exceptions simplifie énormément les problèmes pouvant provenir d'une division par zéro, d'une tentative d'ouverture de fichier non-existant, de lecture sur une disquette non insérée, etc...

Delphi propose deux manières de gérer les exceptions. Il permet aussi de lever une exception de manière dynamique.

Les lignes de code susceptibles de pouvoir créer une exceptions doivent être placées dans un bloc d'instruction (dit ' bloc protégé ') matérialisé par le mot réservé ' Try' et se terminant par un 'end;'.

Contrairement à l'habitude il n'y a donc pas couplage incontournable de 'begin ' / 'end '. Bien en prendre conscience car cela peut être cause d'erreurs.

Lorsqu'une exception se produit dans le bloc ainsi matérialisé, l'exécution du programme est directement déroutée vers le bloc de 'traitement de l'exception' , matérialisé par les mots 'Except' ou 'Finally '.

Si le bloc d'instruction contenu dans la partie 'Try' contient plusieurs instructions, celles qui sont comprises entre l'instruction qui a générée l'exception et le bloc de traitement sont ignorées. Attention…

Si l'on souhaite voir le résultat du traitement de l'interruption, il faut enlever l'option "Stopper si exception" du menu 'Options | Environnement' onglet 'Préférences'. En effet si cette option est sélectionnée, une exécution du programme dans l'environnement Delphi se traduira par un arrêt du programme lorsque l'exception se déclenchera, malgré la mise en place du traitement.

8.2.1 Try... Except... End

Ce premier type de gestion d'exception permet de travailler avec des commandes susceptibles de générer des exceptions, telles qu'une division par zéro, une débordement de pile, etc... La syntaxe en est la suivante :

```
Try
 { Code susceptible de générer une exception }
Except
 { Code à exécuter dès qu'une exception s'est produite }
End ;
```

Supposons, que nous voulons créer une fonction permettant de calculer la valeur d'une fonction de type F (X). Le prototype d'une telle fonction peut être :

```
Function F (X : Real ; Var Y : Real) : Boolean ;
```

Avec Y = F (X), le booléen renvoyé permet de savoir si la fonction F est bien définie pour la valeur donnée de X.

Le problème dans l'écriture de cette fonction est de savoir quel est l'intervalle de définition de la fonction F. Mais à travers la gestion des exceptions, le problème peut être contourné:

```
Function F (X : Real ; Var Y : Real) : Boolean ;
begin
      Result := True ;
            { Par défaut on suppose que F (X) est défini }
      Try
            Y := 1 / (X * X - 4 * X + 3) ;
                  { Génère une exception pour X = 1 et X = 3 }
      Except
            Result := False ; { Result = valeur renvoyée }
            { Cette ligne n'est exécutée que si F (X)n'est pas définie }
      end ;
end ;

{ Un ' end ' pour le bloc de gestion des exceptions et un pour la fin de la fonction }
```

8.2.2 Try... Finally... End

Cette syntaxe est plus particulièrement utilisée lorsque l'on doit travailler sur des ressources dynamiques. L'usage en est le suivant :

```
{ création de la ressource dynamique }
Try
    { Code susceptible de générer une exception }
Finally
    { Libération de la ressource dynamique }
End ;
```

Dans ce cas, la clause Finally est toujours exécutée (qu'il y ait eu exception ou non).

Autrement dit :

- Si dans la clause Try, une ligne de code génère une exception, alors on passe directement à la clause Finally.
- Si aucune exception n'est générée dans la clause Try, alors, une fois la dernière ligne de code du bloc protégé exécutée, on exécute les instructions contenues dans la clause Finally.

Exemple :
L'on souhaite modifier le texte d'un bouton en réaction à un événement " click" dessus.

Le nouveau texte correspond au résultat d'une fonction F (X) définie dans l'exemple précédent. Ce qui peut générer une exception si le calcul de F (X) n'est pas réalisé dans son domaine de définition.

On peut résoudre le problème de la manière suivante :

```
Procedure TForm1.Button1Click (Sender : TObject) ;
Var
    Resultat : MaClasse ;
Begin
    Resultat := MaClasse . Create ; { Création d'une ressource }
    Try
        Resultat.VarPublique := F (3); { Génère une exception }
        Button1.Caption := FloatToStr (Resultat.VarPublique) ;
    Finally
        Resultat.Free ;        { Libère la ressource }
    End ;
End ;
```

Si une exception a lieu, alors le message suivant apparaîtra :

Ce qui est déjà plus sympathique qu'un blocage complet du système après affichage d'une fenêtre système sur fond blanc de mauvais aloi.

8.2.3 Raise

Le mot clé 'Raise' permet de générer une exception , volontairement, à partir du programme.

Cela se fait grâce à la syntaxe :

```
Raise <CLASSE D'EXCEPTION>.Create (<MESSAGE>) ;
```

Dans l'exemple précédent, on suppose que l'on connaît le domaine de définition de la fonction. Par conséquent nous pouvons avoir :

```
Function F (X : Real ; Var Y : Real): Boolean ;
Begin
    Result := True ;
    If ((X = 1) Or (X = 3)) Then
    Begin
        Raise EDivByZero.Create (' X doit être différent de ' +  '1 et de 3...');
        Result := False ;
    End
    Else
        Y := 1 / (X * X - 4 * X + 3) ;
End ;
```

Si X vaut 1 ou 3, Raise va provoquer l'exception EDivByZero. Cela se caractérise par l'apparition d'un message Windows :

8.2.4 Les différentes exceptions

Chaque exception est gérée par un objet particulier dérivée de la classe Exception. Delphi propose un nombre relativement important d'objets exceptions chargés de traiter différents types d'exceptions.

Chaque exception pouvant arriver est définie par un identificateur unique qu'il est possible d'utiliser dans la syntaxe : On < Exception_Id > do ...

Il est possible que plusieurs exceptions soient susceptibles de se produire pendant l'exécution du bloc d'instruction "protégé" par Try . On peut alors tester la valeur de l'exception qui s'est produite pour adapter précisément les instructions à exécuter dans le bloc dépendant de Except.

Liste des principales exceptions :

EconvertError	Exception lancée lorsque les fonctions StrToInt ou StrToFloat ne sont pas en mesure de convertir la chaîne spécifiée vers une valeur entière ou flottante valide.
EdataBaseError	Exception déclenchée lorsqu'une erreur de base de données se produit (Par exemple, si l'application tente d'accéder aux données d'une table qui n'est pas encore ouverte)
EDBEditError	Exception déclenchée lorsque les données ne sont pas compatibles avec le masque défini pour le champ.
EDBEngineError	Exception déclenchée quand une erreur BDE se produit.
EdivByZero	Exception liée aux calculs sur les entiers. L'exception se produit lorsque votre application tente une division par 0 sur un type entier.
EFCreateError	Exception déclenchée lorsqu'une erreur se produit à la création d'un fichier (par exemple, le nom du fichier spécifié peut être incorrect ou le fichier ne peut être recréé car il n'est accessible qu'en lecture).
EFOpen	Exception déclenchée lors d'une tentative de création 'un objet flux de fichier et le fichier indiqué ne peut être ouvert.
EGPFault	Exception liée au matériel déclenchée lorsque l'application tente d'accéder à une partie de la mémoire qui lui est interdite.
EinOutError	Exception déclenchée à chaque fois qu'une erreur d'entrée/sortie MS-DOS se produit. Le code d'erreur résultant est renvoyé dans le champ ErrorCode. La directive $I+ doit être activée pour qu'une erreur d'entrée/sortie déclenche une exception. Si une erreur d'E/S se produit alors que l'application se trouve dans l'état $I-, celle-ci doit appeler la fonction IOResult pour résorber l'erreur.
EintOverFlow	Exception liée aux calculs arithmétiques sur des entiers. Elle se produit lorsque le résultat d'un calcul est trop grand pour le registre qui lui est alloué, entraînant la perte de la donnée.
EinvalidGraphic	Exception déclenchée lorsque l'application tente d'accéder à un fichier qui n'est pas un bitmap, une icône, un métafichier, ou un graphique défini par l'utilisateur.
EinvalidGridOperation	Exception déclenchée lorsqu'une opération non permise est tentée sur une grille (par exemple, lorsque l'application essaie d'accéder à une cellule inexistante).
EinvalidPointer	Exception déclenchée lorsque l'application tente une opération non permise sur des pointeurs.

ElistError	Exception déclenchée lorsqu'une erreur se produit dans un objet liste, chaîne, ou liste de chaînes. Les exceptions liées aux erreurs se produisent si votre application fait référence à un élément se situant en-dehors de la portée de la liste.
EoutOfMemory	Exception signalant les erreurs du tas. Elle se produit lorsque l'application tente une allocation dynamique de mémoire et que l'espace mémoire disponible dans le système est insuffisant pour accomplir l'opération demandée.
EoutOfResources	Exception se produisant lorsque votre application tente de créer un descripteur Windows et que Windows n'est pas en mesure de fournir un descripteur pour l'allocation
EPrinter	Exception déclenchée lorsqu'une erreur se produit à l'impression.
ERangeError	Exception liée aux calculs sur les entiers. Elle se produit lorsque l'évaluation d'une expression entière dépasse les bornes admises pour le type entier de l'affectation.
EZeroDivide	Exception liée aux calculs sur des flottants. Elle se produit lorsque votre application tente de diviser une valeur flottante par zéro.

Exemple :

```
Try
    { Code susceptible de générer une exception }
Except
    On EDivByZero Do
    {Code à exécuter lorsqu'il se produit une division par zéro }
    On EDDEError Do
    { Code à exécuter s'il y a un problème de communication DDE}
    ...
End ;
```

Dans cette version, on ne prend en compte que certaines exceptions : la gestion est donc plus précise.

8.2.5 L'événement OnException du composant TApplication :

Le composant TApplication dispose de l'événement OnException qui permet de gérer toutes les exceptions qui ne sont pas prises en compte par ailleurs et qui, par défaut seraient traitées par la méthode HandleException (une boîte de message est affichée pour indiquer qu'une erreur a eu lieu).

Comme TApplication n'est pas accessible via l'inspecteur d'objet il faut écrire le code suivant:

```
procedure TForm1.GereException (Sender: TObject);
begin
     MessageDlg('Une erreur s''est produite', mtWarning,[mbOk], 0);
end;
{ Ne pas oublier de déclarer l'en tête de la procédure dans la section déclaration de l'unité }
```

L'initialisation du gestionnaire d'événement se fera alors sous la forme :

```
procedure TForm1.FormCreate (Sender : TObject);
begin
    ...
```

```
    Application.OnException := GereException ;
    ...
end ;
```

8.3 Contrôle de validité d'un champ

Il ne suffit pas de proposer des interfaces évoluées comportant de nombreuses fonctionnalités, encore faut-il pouvoir contrôler, de manière précise, les données entrées par l'utilisateur. La fiabilité de l'application en dépend.

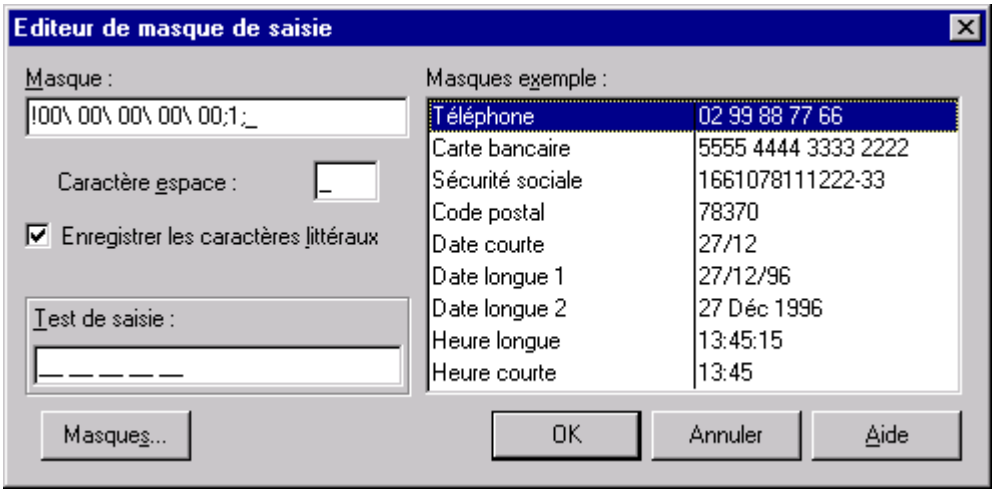
Plusieurs possibilités de contrôles sont mises à la disposition du programmeur pour qu'il produise une application robuste.

8.3.1 Composant TMaskEdit

Ce composant, dérivé du composant TEdit dont il reprend la plupart des caractéristiques, permet de définir des masques de saisie.

Sa propriété spécifique est la propriété EditMask qui donne accès à un éditeur de masque de saisi (Aide contextuelle)

Cet éditeur offre la possibilité d'utiliser quelques masques prédéfinis mais est surtout utiliser pour en créer de nouveaux (avec la possibilité de les enregistrer).



Le principal problème de ce composant est que la syntaxe utilisée pour définir des masques est lourde et difficile à assimiler.

Il n'empêche qu'il est intéressant d'emploi pour assurer la fiabilité de données aux formats bien précis nécessitant des séparateurs (numéros de téléphone, dates diverses, etc...).

Lorsqu'un masque avec séparateurs est défini, l'utilisateur n'a pas à saisir ces derniers. Il n'a plus qu'à saisir les nombres qui composent l'identifiant.

8.3.2 Utilisation de l'événement OnChange

Il est possible de vérifier la validité d'un champ, et ce en temps réel, en utilisant l'événement OnChange, disponible dans l'ensemble des composants autorisant la saisie.

Cet événement est déclenché dès qu'il se produit un changement dans la propriété majeure du composant (en général la propriété Text).

```
procedure TFeuille.Edit1Change(Sender: TObject );
begin
    if not IsValidIdent(Edit1.Text) then
    begin
        MessageDlg ('Le champ ne doit pas commencer par un chiffre',mtWarning, [mbOK], 0) ;
        Edit1.Clear ;
    end ;
end ;
{ La fonction IsValidIdent () vérifie que la chaîne entrée est un identificateur valide (donc elle ne peut
commencer par un chiffre). Ce n'est cependant pas suffisant car la saisie peut commencer par une
lettre et poursuivre par des chiffres }
```

8.3.3 Utilisation de l'événement OnExit

Cet événement n'est appelé que lorsque le composant cesse d'être le composant actif. Dans ce cas on peut tester l'intégralité de la nouvelle valeur saisie.

```
procedure TForm1.Edit1Exit (Sender: TObject);
var
    ret : Longint ;
begin
    if Edit1.Text <> '' then
    begin
        ret := StrToInt (Edit1.Text);
        if ret < 0 then
        begin
            ShowMessage('Il ne peut y avoir de valeurs négatives ');
            Edit1.Clear ;
            Edit1.SetFocus ;
        end ;
    end ;
end;

{ La première méthode, utilisant l'événement OnChange, est plus rapide car elle détecte l'erreur dès le
caractère ' - ' entré. La deuxième méthode est plus générale car elle s'assure de la validité totale de la
"chaîne " numérique entrée }
```

8.3.4 Contrôle du clavier

▲ **Utilisation des événements OnKeypress, OnKeyDown, OnKeyup :**

Ces événements permettent de tester la touche entrée au clavier avant qu'elle soit diffusée au composant destinataire. En créant un gestionnaire d'événement approprié, une variable Key est créée. Elle permet de tester les caractères entrés au clavier. Il est donc possible de n'accepter que les caractères "valides".

La variable Key est capable de gérer pratiquement toutes les touches du clavier : touches alphanumériques bien sûr mais aussi les touches étendues (' F1' à 'F12', flèches, PageUp et PageDown, etc .....).

Chaque touche du clavier est décrite par une variable prédéfinie (VK_...) permettant de tester directement la variable Key :

```
if Key = Vk_SPACE then
      ShowMessage ('Touche espace ');
if Key = Vk_NUMPAD0 then
      ShowMessage ('Touche 0 du pavé numérique);
etc .....
```

La liste complète des codes clavier utilisés par Windows figure en annexe 1.

OnKeyPress ne gère pas les touches étendues. Pour cela il faut utiliser l'événement OnKeyDown qui utilise aussi la variable Key.

Exemple :

Pour tester si une valeur entrée est un nombre positif (et donc éliminer toutes les valeurs commençant par ' - ' on est coincé car les solutions précédentes ne gèrent pas correctement le caractère ' - '. Une solution, dérivée de la méthode précédente, utilisant l'événement OnKeyPressed et testant le ' - ' est envisageable.

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
      if (Key = '-') then
      begin
            ShowMessage ('Nombre négatif interdit');
            Key := Chr (0);
            Edit1.SetFocus ;
      end ;
end;
```

{ L'instruction Key := Chr (0) retire le caractère non accepté du buffer du clavier }

Exemple de procédure s'assurant que les caractères entrés ne sont que de type alphabétiques :

```
procedure TForm1.Edit1KeyPress (Sender: TObject; var Key: Char);
begin
    If (Key >= 'A')AND (Key <= 'Z')then
```

```
        Key := Chr(Ord(Key)+ 32); { transformation en minuscule }
end;
```

Autre exemple permettant de saisir seulement des chiffres:

```
procedure TForm1.Edit1KeyPress (Sender: TObject; var Key: Char);
begin
    If Not(Key in [ '0' .. '9' ]) then
    begin
        Key := Chr (0);      { annulation du caractère non conforme }
        Showmessage ('On ne peut entrer que des chiffres ');
    end ;
end;
```

▲ **Utilisation de la propriété KeyPreview du composant Tform :**

Lorsque la propriété KeyPreview := True, les événements du clavier sont adressés d'abord à la fiche avant d'être diffusés aux composants. On peut donc créer des gestionnaires d'événements au niveau de la feuille afin de gérer le clavier quel que soit le composant destinataire.

8.3.5 Utilisation d'un gestionnaire d'exception

Il est possible, et même conseillé, d'utiliser les possibilités de gestion des exceptions offertes par Delphi pour rendre les applications plus robustes.

```
procedure TForm1.Edit1Change (Sender: TObject );
var
    ret : Longint ;
begin
    if Edit1.Text <> '' then
    begin
        try
            ret := StrToInt (Edit1.Text);
        except
            on EConvertError do
            begin
                MessageDlg ('Le champ doit être un nombre ', mtWarning, [mbOK], 0);
                Edit1.Clear ;
                Edit1.SetFocus ;
            end ;
        end ;
    end ;
end;
```

{ Pour s'assurer que le nombre entré est un nombre on est obligé de faire le test préalable "'Edit1.Text < > ' ' " car le fait d'effacer la zone de saisie en fin de procédure (Edit1.Clear)provoque un nouveau changement qui appelle de nouveau l'événement OnChange. Et dans ce cas il y a une erreur en boucle car StrTOInt () ne sait pas gérer la chaîne vide }

Il y a affichage d'un message dès qu'un caractère alphabétique est entré. Cependant la solution n'est pas encore optimale car elle oblige à rentrer de nouveau tout ce qui était entré auparavant. La solution utilisant OnKeyPressed est plus satisfaisante dans ce cas.

8.4 Affichage de nombres réels

Il est difficile d'afficher les nombres réels. Si l'on n'y prend pas garde ils sont affichés selon la notation scientifique, ce qui les rend d'une lecture malaisée.

▲ **Utilisation de la procédure Str () :**

Son prototype est :

```
procedure Str (X [: Width [: Decimals ]]; var S) ;
```

Les deux paramètres Width et Decimals sont optionnels.

Si on utilise cette fonction telle quelle on a :

```
nb := (123000 * 7)/ 11 ;
Str (nb , nombre);
Edit1.text := nombre ;
```

{ Le nombre affiché, vraisemblablement correct, est affiché en notation scientifique }

En utilisant les deux paramètres on obtient un affichage plus lisible :

```
nb := (123000 * 7) / 11 ;
Str (nb : 2 : 2 , nombre);
Edit1.text := nombre ;
```

{ l'affichage se fait avec deux décimales. Le paramètre Width n'est pas pris en compte car la valeur résulante dépasse 99. Le résultat affiché est, sous forme d'une chaîne : 78272.73 }

▲ **Utilisation de la fonction Format () :**

La fonction Format () permet de réaliser des affichages complexes, mélangeant texte et valeurs numériques de tous types.

Dans sa syntaxe et son mode de fonctionnement elle ressemble à la fonction Printf () chère au langage C.

En reprenant l'exemple précédent on a :

```
nb := (123000 * 7)/ 11 ;
Resultat := Format ('Résultat : %10F2 ' , [nb]);
Edit2.Text := Resultat ;
{ Affiche : ' Résultat : 78272,73' dans la zone d'édition }
```

Il est préférable d'utiliser la fonction FormatFloat () qui permet de spécifier le nombre de chiffres après la virgule.

```
nb := (123000 * 7)/ 11 ;
Resultat := FormatFloat ('#####.##' , nb);
Edit3.Text := Resultat;
```

9 Autres possibilités offertes

D'autres composants sont disponibles dans la bibliothèque VCL pour permettre de réaliser, plus ou moins facilement, des applications utilisant des mécanismes évolués proposés par Windows. Grâce à eux on pourra réaliser des opérations de "drag and drop", utiliser le presse-papiers, créer des applications MDI, etc…

9.1 Gestion d'une application MDI

Une application MDI est une application particulière dont la fenêtre principale peut contenir d'autres fenêtres appelées fenêtres MDI, ou plus simplement "fenêtres filles", identiques. Les caractéristiques principales de cette structure sont :

- L'affichage des différentes fenêtres est limité à la surface occupée par la fenêtre mère (il y a clipping des zones débordantes, mais Windows continue à gérer les surfaces ainsi effacées).
- La fermeture de la fenêtre mère entraîne celles de toutes les fenêtres filles ouvertes.
- Le fenêtres filles fonctionnent en mode amodal.
- Lorsqu'elles sont inactives, les fenêtres filles apparaissent sous formes iconifiées.

Une application MDI est surtout utilisée quand on veut pouvoir travailler sur plusieurs documents en même temps : chaque fenêtre fille peut gérer un document, indépendamment des autres fenêtres. Comme le mode de fonctionnement est le mode amodal, on peut facilement basculer d'une fenêtre à l'autre.

- Le mode de fonctionnement MDI est le mode de fonctionnement privilégié de Windows 3.1 (exemple : le gestionnaire de fichiers de Windows, les principales applications de bureautique). Windows 95 (ou WindowsNT) privilégie lui le mode SDI (celui de Delphi).

9.1.1 Réalisation d'une application MDI

▲ **Phase de conception :**
Une application MDI est composée de deux types de fiches : une fiche MDI mère et plusieurs fiches MDI filles.

On distingue ces deux catégories, dès la conception, en initialisant la propriété FormStyle :

- A fsMDIForm pour la fenêtre mère
- A fsMDIChild pour les fenêtres filles

Les fenêtres filles sont conçues indépendamment de la fenêtre mère mais ne doivent pas être auto-créées. En général on conçoit un modèle de fenêtre fille et on le crée dynamiquement, en fonction des besoins, au cours de l'exécution du programme.

On peut concevoir plusieurs modèles de fenêtres filles qui seront appelées en fonction des besoins.

Les propriétés du composant TForm utilisées pour la gestion d'une application MDI sont les suivantes :

ActiveMDIchild	Correspond à la fenêtre fille MDI active
FormStyle	Permet de définir la fenêtre MDI mère et les fenêtres MDI filles.
MDICount	Indique le nombre de fenêtres filles
MDIChildren [i]	Permet d'accéder à la fenêtre MDI fille passée en index (base : 0)
WindowMenu	Indique la rubrique du menu principal dans laquelle les noms des différentes fenêtres filles seront affichés pour faciliter leur accès.

La propriété ActiveMDIChild étant elle même du type TForm, on peut utiliser sur elle les propriétés de cette classe :

```
Caption := ActiveMDIChild.Caption;
```

{ Synchronise le texte apparaissant dans la barre de titre de la fenêtre mère avec celui de la fenêtre fille active }.

L'initialisation de la propriété WindowMenu de la fenêtre mère fait que, lors de l'exécution, chaque création de fenêtre fille ajoutera une ligne dans le menu déroulant correspondant, cette ligne faisant référence à la fenêtre fille pour permettre d'y accéder directement.

Exemple :

Si l'item 'Fenêtres' a été créé dans le composant TMenu1 de l'application (et qu'il s'agit d'un item faisant partie du menu principal, c'est à dire celui affiché en permanence en bandeau en haut de l'application), alors l'initialisation :

```
WindowsMenu = Fenetre1
```

fera qu'à l'exécution, les noms des différentes feuilles filles seront affichés dans l'ordre de création dans ce menu.

▲ Création d'une feuille fille MDI :

Une fois le modèle de feuille réalisé et référencé par la clause 'Uses', la création de chaque fenêtre fille se fait grâce à un gestionnaire d'événement associé à un composant (bouton ou item de menu) :

```
procedure TForm1.BCreateClick(Sender: TObject);
begin
    FFille:= TFFille.Create (self );
    try
Ffille.Show ;  {Pas Showmodal car l'application doit être amodale}
    finally
        inc (nb);
        FFille.Caption := 'Fenêtre MDI ' + IntToStr (nb);
    end ;
end;
```

▲ **Manipulation des différentes fenêtres par l'utilisateur :**
L'utilisateur peut activer les différentes fenêtres peuvent en cliquant sur une partie visible.

Elles peuvent aussi être placées en cascade ou en mosaïque si l'on utilise les méthodes Cascade et Tile dans la fenêtre mère (et que l'on a prévu les gestionnaires d'événements adéquats).

Si la propriété WindowsMenu de la feuille mère a été initialisée, on peut accéder aux différentes feuilles filles par le menu adéquat.

Une fenêtre MDI fille ne peut être cachée (en invoquant la méthode Hide): elle ne peut qu'être minimisée via son menu système.

Lorsque l'on demande la fermeture d'une fenêtre fille, elle est minimisée et apparaît en icône sur le plan de travail de la fenêtre mère.

▲ **Manipulation des différentes fenêtres par programmation :**
La fenêtre active est référencée par la propriété ActiveChild .

On peut accéder à une fenêtre quelconque par la propriété MDIChildren [ ] .

```
MDIChildren[2].SetFocus ;  { Active la 3° fenêtre }
```

9.1.2 Menus adaptés aux fenêtres filles

Les fenêtres filles peuvent disposer de leurs propres éléments de menu. Les éléments de ce menu remplacent ceux du menu principal si la propriété GroupIndex de chaque MenuItem n'est pas modifiée (Par défaut cette valeur est toujours à 0, ce qui fait que quand une fenêtre fille est créée, ses éléments de menu remplacent complètement le menu principal).

Pour éviter cela il faut donner des valeurs particulières à tous les objets MenuItem, ceux du menu de la feuille mère comme ceux du menu de la feuille fille (ces valeurs peuvent être les mêmes pour tous les éléments d'un même menu déroulant mais une valeur ne peut être inférieure à celles définies dans le menu déroulant placé à gauche).

A l'exécution :

- Si des éléments du menu de la feuille fille ont des valeurs GroupIndex égales à celles d'éléments de menu de la feuille mère, ils remplacent ces derniers.
- Si des éléments de menu de la feuille fille ont des valeurs GroupIndex supérieures à celles des éléments du menu de la feuille mère, les éléments concernés sont affichés après ceux du menu de la feuille mère. S'ils ont des valeurs intermédiaires ils viendront s'insérer entre les menus en fonction des valeurs GroupIndex.

9.2 Le "Drag and Drop"

Le "Drag and Drop" (littéralement : "tirer et lâcher") est un mécanisme commun sous Windows. Il permet de réaliser des copies de données (texte ou graphique) entre composants au moyen de la souris:

On sélectionne à la souris la donnée à copier puis, tout en conservant le bouton gauche de la souris enfoncé, on déplace celle-ci vers le composant de destination. Lorsque l'on relâche le bouton de la souris, la copie est réalisée.

Ce mécanisme est très fréquemment utilisé dans le gestionnaire de fichiers de Windows (explorateur) pour réaliser des copies et des déplacements de fichiers ou de répertoires au sein de l'arborescence (voire vers un autre lecteur).

Généralement, durant toute l'opération, le curseur de la souris change de forme.

La plupart des composants fournis par Delphi implémentent ce mécanisme. Ils peuvent être "source" et / ou "destination".

▲ Initialisation du mécanisme :

Les contrôles implémentant le mécanisme de Drag and Drop possèdent la propriété DragMode. Cette propriété peut prendre deux valeurs :

dmAutomatic	l'opération commence automatiquement quand l'utilisateur clique sur le contrôle.
dmManual	l'opération n'est lancée qu'en gérant les événements de bouton de souris enfoncé (valeur par défaut).

Exemple :

Soit un projet composé de deux composants TListBox. ListBox1 est initialisée avec quelques chaînes de caractères. DragMode := dmManual et DragCursor := crCross (forme du curseur pendant l'opération de "drag").

Pour faire glisser un contrôle manuellement, il faut faire appel, au sein du gestionnaire d'événement associé à OnClick, à la méthode BeginDrag().

Cette méthode requiert un paramètre booléen appelé Immediate qui influence sur le déclenchement du mécanisme :

True	l'opération glisser commence immédiatement (ce qui revient à DragMode := dmAutomatic).
False	l'opération ne commence pas avant que l'utilisateur déplace effectivement la souris, même sur une courte distance.

L'appel à BeginDrag (False) permet au contrôle d'accepter les clics de la souris sans lancer une opération glisser.

Dans l'exemple précédent on a donc :

```
procedure TForm1.ListBox1Click(Sender: TObject);
begin
     if ListBox1.DragMode = dmManual then
          ListBox1.BeginDrag (True);
end;
```

▲ **Poser d'une donnée sur un composant :**
Lorsque le curseur de la souris arrive sur le composant destiné à recevoir la donnée, il faut dans un premier temps faire en sorte que celui-ci l'accepte. Cette procédure permet de réaliser certains contrôles (un composant peut n'accepter des données que provenant d'une source précise).

Pour cela il faut créer le gestionnaire d'événement associé à l'événement OnDragOver du composant de destination et positionner la variable Accept à True.

(L'événement OnDragOver se déclenche dès que le curseur passe sur le composant).

Dans l'exemple précédent on a :

```
procedure TForm1.ListBox2DragOver(Sender, Source: TObject; X, Y:
      Integer; State: TDragState; var Accept: Boolean);
begin
      if Source is TListBox then Accept := True;
end;

{ La vérification se fait grâce à la variable 'source' qui indique le type de composant source de la
donnée }
```

La vérification est encore plus précise si l'on utilise la forme :

```
      if Source = ListBox1 then Accept := True;
```

qui permet de spécifier précisément le composant source.

Il ne reste plus alors qu'à réaliser le transfert effectif de la donnée, à l'aide du gestionnaire d'événement associé à l'événement OnDragDrop.

Dans l'exemple habituel on a :

```
procedure TForm1.ListBox2DragDrop (Sender, Source: TObject; X, Y: Integer);
begin
      ListBox2.Items.Add (ListBox1.Items [ ListBox1.ItemIndex ]);
      ListBox1.Items.Delete (ListBox1.ItemIndex );
end;

{Dans l'exemple on réalise un "couper / coller" mais ce n'est pas une obligation}
```

9.3 Le composant TTimer

Le composant TTimer est très utile quand on veut réaliser certaines opérations à des intervalles précis, en tâche de fond.

- Ces opérations sont exécutées selon la périodicité indiquée dans la propriété Interval (en milliseconde).
- Elles ne sont exécutées que lorsque le composant est actif (Enabled := True).

Par exemple on peut vouloir indiquer dans un composant TPanel, utilisé en tant que barre d'état l'action réalisée par un bouton.

Le code résultant est alors :

```
procedure TForm1.Button1Click (Sender: TObject);
begin
    Panel1.Caption := 'J"ai appuyé sur le bouton' ;
end;
```

{ Mais une fois l'affichage réalisé il reste affiché tant qu'une action sur un autre bouton au comportement similaire ne vient pas modifier la légende }

On peut alors utiliser un composant TTimer initialisé à Enabled := False (de manière à ce qu'il ne se déclenche pas systèmatiquement dès le lancement de l'application). Le gestionnaire d'événement associé au bouton est alors :

```
procedure TForm1.Button1Click (Sender: TObject);
begin
    Panel1.Caption := 'J"ai appuyé sur le bouton' ;
    Timer1.Enabled := True ;
end;
```

{ L'instruction déclenche le timer. Son action se produira lorsque la durée spécifiée par Interval sera écoulée }

Le gestionnaire d'événement associé à l'événement OnTimer du timer est :

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    Panel1.Caption := '' ;
    Timer1.Enabled := False ;
end;
```

{ Désactivation du timer pour éviter qu'il ne continue à fonctionner indéfiniment }.

Même si l'utilisation des timers s'avère pratique, il ne faut pas en abuser car un timer consomme beaucoup de ressources. En règle générale il ne faut pas avoir plus de trois ou quatre timers dans une application.

9.4 Constitution d'une barre d'état

Il est fréquent qu'une application propose, en bas de sa fenêtre principale une barre affichant certains renseignements (date, répertoire courant, action en cours.....). Cette barre est appelée barre d'état.

▲ **Utilisation de composants TPanel :**
Une barre d'état peut être constituée à partir de composants TPanel.

On utilise un composant principal (aligné, grâce à la propriété Align := alBottom, de manière à se redimentionner automatiquement si la fenêtre l'est. Puis l'on pose d'autres TPanel au dessus (en

utilisant d'autres types de bordures de manière à ce qu'ils apparaissent facilement). Chacun de ces composants peut afficher une information particulière.

```
    TPanel.Caption := DateToStr (Date);
{ Affichage de la date système }
```

Dans l'exemple précédent, sur le composant TTimer, le composant TPanel utilisé pouvait faire partie d'une barre d'état.

▲ **Utilisation du composant THeader :**

Le composant THeader permet de réaliser des barres d'états "professionnelles" comportant plusieurs sections séparées par des barres verticales. Il est cependant un peu plus difficile à manipuler.

Lors de la phase de conception on peut le configurer en indiquant le nombre de sections souhaité.

Pour cela il faut indiquer, dans la propriété Sections (de type TStrings) le nom des sections envisagées (en fait on peut se contenter, dans l'éditeur de chaînes qui apparaît, d'entrer des lignes vides). Le nombre de lignes impose le nombre de sections. Mais la taille de chacune d'entre elle ne peut être initialisée qu'à l'exécution.

A l'exécution, dans l'événement OnCreate de la feuille on peut terminer la configuration et indiquer les valeurs qui seront affichées.

Exemple d'une barre d'état comportant deux sections: une pour afficher la date courante et l'autre le répertoire courant de l'application.

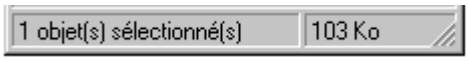
```
procedure TForm1.FormCreate(Sender: TObject);
begin
    Header.SectionWidth [ 0 ] := 150 ;
    Header.SectionWidth [ 1 ] := 400 ;

    Header.Sections [ 0 ] := DateToStr (date);
    Header.Sections [ 1 ] := ExtractFilePath (Application.ExeName);
end;

{ Bien sûr on pourrait afficher dans la deuxième section les informations concernant l'action en cours
(éventuellement en utilisant la propriété Application.Hint) puis les effacer par le renseignement par
défaut grâce à un timer ..... il n'y a pas de limite à l'imagination }.
```

▲ **Utilisation du composant TStatusBar :**

Sous Windows 95 ou NT, il est possible, et c'est fortement conseillé, d'utiliser l'objet TstatusBar. Ce type de barre d'état ne peut être utilisé qu'avec des fenêtres redimensionnables.

Toutes les caractéristiques des sections de la barre d'état peuvent être configurées lors de la phase de conception, grâce à la boite d'édition de la propriété *Panels* (bouton '…'). Lors de l'exécution, le texte de chacune des section peut être modifié par l'instruction suivante :

```
...
StatusBar1.Items[0].Text := 'Ceci est le premier volet' ;
StatusBar1.Items[1].Text := 'Ceci est le volet n°2' ;
...
```



9.5 Jauges

Les jauges, comme les barres d'icônes et les barres d'état sont à la mode. Delphi ne pouvait pas ne pas proposer des composants permettant de réaliser certaines initialisations de variables par l'utilisation de ce type de gadget.

Deux composants de Delphi peuvent être utilisés : un ne réalise que des affichages d'informations alors que l'autre est manipulable par l'utilisateur et permet réellement la modification de valeurs.

▲ **Le composant TScrollBar :**
Ce composant est en fait l'ascenseur traditionnel du monde Windows. Mais il peut être aisément détourné de sa fonction première pour être utilisé en tant que jauge interactive.

Principales propriétés :

Kind	Permet de déterminer si l'on souhaite que la jauge soit verticale ou horizontale.
LargeChange	Détermine le pas du déplacement lorsque l'on clique sur la barre de défilement
Max Min	Valeurs maximale et minimale du curseur
Position	Position courante du curseur
SmallChange	Détermine le pas du déplacement lorsque l'on clique sur les flèches situées aux extrémités de l'ascenseur.

Exemple d'une jauge pouvant initialiser une valeur (affichée ici dans une zone de saisie) et pouvant être initialisée par action sur un bouton. Les valeurs d'initialisation sont :

Min := 0 , Max := 100 ; LargeChange := 10 ; SmallChange := 1 .

Les gestionnaires d'événements créés pour assurer la gestion de la jauge sont :

```
procedure TForm1.ScrollBar1Change (Sender: TObject);
begin
     Edit1.Text := IntToStr (ScrollBar1.Position);
end;

{ Affiche la valeur de la position du curseur de l'ascenseur }


procedure TForm1.BDeplaceClick (Sender: TObject);
begin
     if StrToInt (Edit1.Text)<= ScrollBar.Max then
          ScrollBar1.Position := StrToInt (Edit1.Text);
end;

{ Déplace le curseur de l'ascenseur en fonction de la valeur entrée}
```

▲ **Composant TGauge :**

Delphi propose, dans la palette 'Exemple', le composant TGauge permettant de réaliser une jauge affichant des informations à l'écran.

Par rapport à la jauge précédente celle-ci n'est pas interactive (lecture seule) mais elle permet d'afficher des valeurs en pourcentage et est surtout utilisée pour afficher la progression d'une action (chargement d'un fichier par exemple).

Elle n'est pas documentée dans l'aide mais son fonctionnement est facile à comprendre.

Comme dans le cas précédent on dispose des propriétés Max et Min pour déterminer les valeurs maximale et minimale possibles. Par contre, c'est la propriété Progress qui permet de connaître la valeur courante (cette valeur est affichée, en pourcentage au milieu de la jauge).

Kind permet de définir son orientation. BackColor indique sa couleur lorsque la progression est à 0 %, ForeColor la couleur qui occupe petit à petit la surface de la jauge au fur et à mesure de la progression.

▲ **Composant TProgressBar :**
Ce composant est un composant standard Windows 95 et NT. Il s'utilise simplement et de la même manière qu'une Tgauge.

9.6 Le composant TStringGrid

Ce composant, assez complexe à manipuler, permet de créer des tableaux de chaînes de caractères à l'écran. Il offre la possibilité de définir des lignes et des colonnes de titres et permet même de rendre celles-ci fixes quand on se déplace dans le tableau.

Ses principales propriétés sont :

Canvas	Pour pouvoir dessiner dans la grille
Cells [i , j]	Propriété principale permettant d'accéder individuellement à chaque cellule(base : 0, 0)
Col Row	Valeurs de la colonne et de la ligne courantes
ColCount	Nombre de colonnes du tableau
RowCount	Nombre de de lignes du tableau
ColWidths [i]	Largeur d'une colonne
DefaultColWidth	Valeur par défaut de la largeur des colonnes
Fixedcolor	Couleur des lignes et des colonnes fixes
FixedCols	Nombre de colonnes fixes
FixedRows	Nombre de lignes fixes
Selection	Renvoie les coordonnées de la cellule active
TopRow	Détermine la première ligne non fixe qui apparaît en haut

La méthode MouseTopCell () renvoie les coordonnées de la souris (en pixel) ainsi que la cellule sur laquelle se trouve le curseur.

Exemple d'utilisation :

Soit un composant de type TStringGrid, appelé SGInfos, composé de 2 colonnes et de 5 lignes. La première colonne a été configurée en tant que colonne fixe et présente donc une couleur particulière. A l'exécution on peut utiliser le tableau comme suit :

```
with SGInfos do
begin
      ColWidths [ 0 ] := 90 ;
      ColWidths [ 1 ] := 150 ;

      Cells[0,0] := 'Auteurs :';
      Cells[0,1] := 'Edition :';
      Cells[0,2] := 'Année :';
      Cells[0,3] := 'Tome :';
      Cells[0,4] := 'Num. collection :';

      Cells[1,0] := TBibliAuteur.Value ;
      Cells[1,1] := TBibliEdition.Value ;
      Cells[1,2] := TBibliAnnee.AsString ;
      Cells[1,3] := TBibliTome.AsString ;
      Cells[1,4] := TBibliNum_Collection.AsString ;
end ;
SGInfos.Visible := True ;
```

{ Les renseignements initialisant la deuxième colonne sont puisés dans une base de données (voir chapitres suivants). Telle qu'elle est configurée ce tableau s'affiche sous la forme d'un post-il quand on souhaite obtenir tous les renseignements sur un enregistrement de la table en double-cliquant sur celle-ci }

10 Programmation des bases de données

Une des spécificités de Delphi est qu'il permet de créer facilement des applications de gestion de bases de données.

Ces applications peuvent fonctionner :

- Directement avec des bases de données aux format Paradox, dBase ou Interbase.
- Via des pilotes ODBC, qui permettent la connexion - en local ou à travers un réseau - avec la plupart des bases de données reconnaissant ce protocole (Access et FoxPro en particulier).

La version "client / serveur" de Delphi permet quant à elle l'accès aux principaux serveurs de bases de données SQL : Oracle, Sybase, Informix. On peut donc manipuler des données dans un environnement distribué (sur un réseau . Si l'on dispose des pilotes ODBC adéquats on peut ainsi accéder à l'ensemble des produits du marché).

10.1 Les bases de données

Une base de données est une structure servant à gérer et à conserver physiquement des données. Lorsque le nombre de données devient important, on parle plutôt de système de gestion de bases de données car il faut mettre en oeuvre un ensemble de dispositifs permettant :

- De créer ou détruire les entités dans lesquelles les données seront stockées.
- D'assurer l'intégrité des différentes données.
- D'éviter la redondance des informations en adoptant des structures adaptées.
- De permettre l'accès aux données.
- De gérer les accès concurrents aux données (cas d'un environnement multi-utilisateurs).

Il existe plusieurs types d'architecture permettant la gestion des données. On parle en particulier de "bases de données hiérarchiques" et de "bases de données relationnelles".

Seules les bases relationnelles nous intéressent car c'est selon le modèle relationnel que fonctionnent les différentes bases reconnues par Delphi.

10.1.1 Base de données relationnelle

Les bases de données relationnelles gèrent les données selon le mode, mis au point par E.F. Codd, basé sur l'utilisation du langage SQL.

Ce mode sépare l'aspect physique des données (stockage sur disque) de leur représentation logique (présentation à l'utilisateur). Les données apparaissent alors sous forme de tables qui masquent la complexité des mécanismes d'accès en mémoire. Ce modèle libère l'utilisateur de tous les détails de stockage et permet un accès purement logique aux données.

Chaque table permet de stocker un type particulier de données. Une donnée stockée constitue un enregistrement dans la table (= une ligne). Elle peut être constituée d'un nombre variable d'informations élémentaires. Chaque information élémentaire constitue un champ de l'enregistrement (= une colonne).

Les besoins actuels nécessitent l'utilisation fréquente de dizaines de tables, chacune permettant le stockage d'une donnée structurée particulière. Il y a lieu d'éviter la redondance des données élémentaires stockées. L'ensemble de tables ainsi constitué s'appelle une base de données.

On appelle clé un identifiant permettant de déterminer d'une façon unique un enregistrement dans une table. Une clé peut être :

- Un des champs de la table si les valeurs qu'il peut prendre sont toutes différentes ;
- Un ensemble de champs permettant de définir un identifiant unique ;
- Un champ spécial qui n'est pas une donnée proprement dite mais dont on peut être certain de l'unicité (identifiant numérique en général).

- Il ne faut pas confondre une clé et un index. :

  - Un index est une information sur l'ordonnancement des différents enregistrements dans une table (il peut ne pas correspondre à l'ordonnancement physique des enregistrements). Il sert à réaliser des tris. On peut définir plusieurs index pour une même table.
  - La clé sert à déterminer précisément un enregistrement. Elle est unique.

Par exemple on peut définir le champ "Nom" comme index pour pouvoir avoir une vision "triée" des différents enregistrements. Mais "nom" ne peut être une clé.

Lorsqu'un index correspond à une clé on dit qu'il est "primaire". Dans les autres cas il est dit "secondaire".

Toutes les données (enregistrements) d'une table sont définies par les mêmes champs.

Au niveau de la programmation on peut assimiler la définition d'une donnée à une structure.

10.1.2 Notion d'enregistrement courant

On peut se déplacer au sein d'une table. Le gestionnaire de base de données met en place un pointeur permettant de désigner la position courante dans la table. Ce pointeur désigne un enregistrement courant qui est le seul actif : il est donc le seul, à un moment donné, à être lisible et / ou modifiable.

Au niveau physique, le gestionnaire de base de données met en place une zone mémoire tampon dans lequel il recopie systématiquement l'enregistrement courant. Lorsque l'utilisateur réalise des modifications, c'est sur cette copie qu'elles interviennent. Il n'y a que quand l'utilisateur valide ses modifications (implicitement, lorsqu'il change d'enregistrement courant, ou explicitement) que la zone mémoire tampon est recopiée sur le disque et qu'elle écrase la valeur de l'enregistrement courant.

A l'ouverture d'une table, l'enregistrement courant est le premier enregistrement.

10.1.3 Tables liées

Un des intérêts majeurs des bases de données est qu'elles permettent d'éviter la duplication des informations (d'où gain de temps à la saisie et gain de place): il ne peut y avoir plusieurs fois la description d'un même élément.

Pour que cela soit possible il faut établir des liens entre les différentes tables de la base. Ces liens permettent de retrouver des informations dans une table à partir d'une autre table.

Les liens entre table ne peuvent être réalisés qu'à l'aide de champs définis comme des clés. Cela garantit que les informations lues dans une table sont bien celles correspondant à la donnée de la table "maître".

10.1.4 Requêtes et vues

Les champs d'une base de données peuvent être visualisés et peuvent servir à en calculer d'autres.

▲ **Les requêtes :**
Pour accéder aux différentes informations l'utilisateur doit exécuter différentes requêtes qui seront ensuite interprétées par le gestionnaire de base de données.

Dans le cas d'une base utilisée dans le contexte client / serveur, seule la requête est transmise au serveur. Celui-ci la traite puis renvoie les résultats au client.

Une requête est formulée selon un langage particulier, qui est d'ailleurs un des fondements des bases de données relationnelles : le langage SQL.

▲ **Les vues :**
Les requêtes permettent de ne visualiser que les enregistrements ou les données qui remplissent certains critères. Le gestionnaire de base de données est en mesure de réaliser ces opérations et ne fournir à l'utilisateur que les "vues" qu'il souhaite des différentes tables.

Une vue peut être affichée sous forme de tableau (chaque ligne correspond à un enregistrement et les colonnes correspondent aux champs sélectionnés) ou sous forme de fiche : un seul enregistrement est à l'écran en même temps.

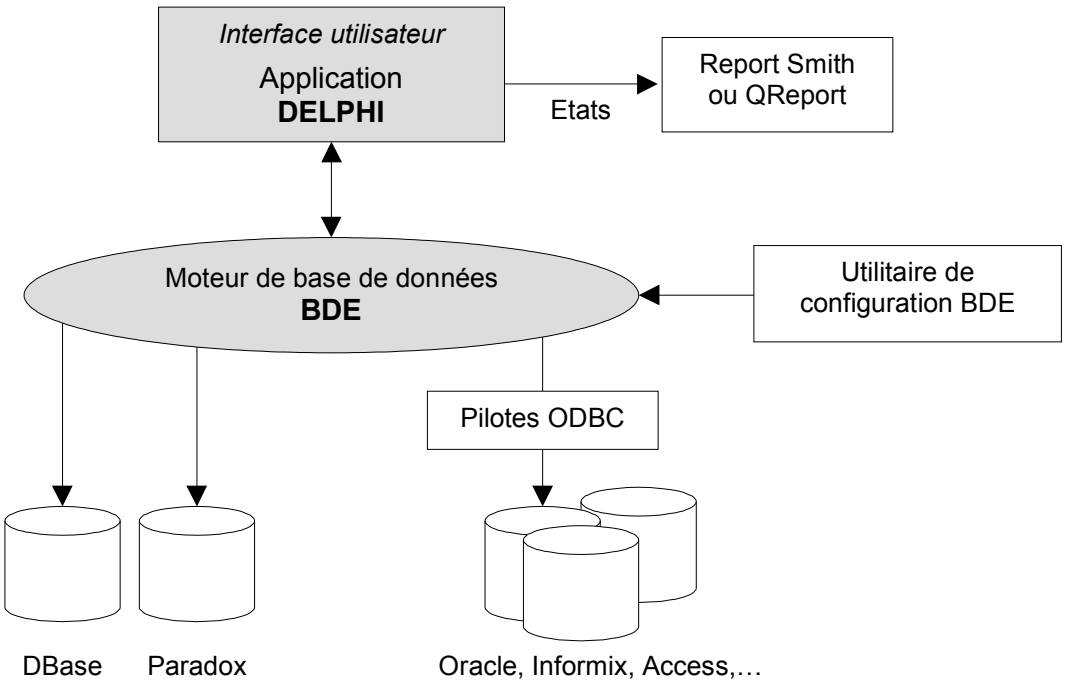
Lorsqu'une table est triée selon un index secondaire, elle présente à l'utilisateur une "vue" qui diffère de son implémentation physique. Normalement on ne peut modifier les données affichées dans une vue. Mais Delphi soutient le principe des "requêtes vivantes" (requests live) qui permet de modifier les données affichées sous forme de vues de manière à ce que ces modifications remontent jusqu'aux tables d'où les données sont extraites.

Il est possible affiche des données provenant de plusieurs tables à la fois. Pour ce faire il a fallu que la requête formulée réalise ce que l'on appelle des "jointures" en utilisant les différents champs liant les différentes tables entre elles.

10.2 Architecture interne permettant l'accès aux bases de données

Delphi met en place un certain nombre d'utilitaires et de mécanismes internes pour qu'une application ait accès aux données gérées par les différents SGBDR.

Il fournit en particulier un moteur de base de données interne appelé BDE (Borland Database Engine), qui permet de créer et de gérer des bases de données locales. C'est BDE qui assure par ailleurs la communication avec les autres bases de données.



L'architecture interne de Delphi pour la gestion des bases de données

Un moteur de base de données est un ensemble de routines chargées de gérer les différentes données stockées dans les tables (accès aux tables, ajout, modification et suppression d'enregistrement, gestion des accès concurrents). C'est le cœur d'un SGBDR. Il ne permet pas de créer les tables et n'assure pas l'interface avec les utilisateurs. Ces fonctionnalités sont réalisées par d'autres utilitaires.

Le moteur de base de données BDE est l'élément central d'une application de gestion de base de données créée avec Delphi.

Il est inclus directement dans les composants spécifiques fournis avec Delphi. Un programmeur n'a donc pas à s'en occuper et il n'apparaît pas dans l'arborescence de l'application créée. Par contre l'exécutable généré est plus important.

L'application Delphi créée est essentiellement constituée d'une l'interface utilisateur permettant d'accéder de manière transparente à BDE.

Les structures des différentes tables utilisées dans l'application sont mises au point par un utilitaire spécial : DBD (DataBase Desktop).

Cet utilitaire, accessible par une icône spécifique ou par le menu 'Outils ', permet de créer les différentes tables de l'application (dénominations, types et tailles des différents champs et définition des clés), de définir les différents index, de créer les liens entre les différentes tables.

DBD correspond en fait au concepteur de tables de Paradox auquel il manque toutes les fonctionnalités de création d'interface et d'édition d'état.

Il est possible d'utiliser des tables déjà conçues par d'autres SGBDR (Paradox ou dBase).

L'utilitaire Report Smith (ou le composant Qreport) permet de générer des états complexes, en mode "wysiwig", à partir des tables gérées par l'application Delphi.

Les pilotes ODBC permettent l'accès à différentes bases de données et serveurs SQL non reconnus directement par Delphi.

10.3 Le module de conception de base de données DBD

Le module Base de Données (DBD) permet de créer, restructurer et d'interroger des tables. Il peut être utilisé de manière totalement autonome avant toute réalisation sous Delphi.

Dans le cadre de la conception d'une application DBD doit être utilisé juste après la réalisation du modèle conceptuel de données. Il permet de réaliser les tables qui découlent de ce modèle.

L'utilitaire DBE est en fait une version "allégée" du SGBDR Paradox auquel seules certaines fonctionnalités ont été ôtées (création de l'interface utilisateur, génération d'états) car reprises par Delphi et Report Smith. Les utilisateurs de Paradox n'auront aucune difficulté à prendre en main DBD.

D'ailleurs pour aller au delà des informations fournies sur DBE par l'aide en ligne (nettement trop concise dans ce domaine) il est souhaitable de se munir d'un livre d'apprentissage à Paradox.

10.3.1 Configuration de DBD

Comme DBD fonctionne sur la même base que Paradox, il gère les tables d'une application de la même manière. De fait une application de gestion de base de données stocke les différentes tables de données ainsi que les tables de gestion internes (tables d'index, d'intégrités référentielles, etc.....) propres au SGBDR dans un répertoire particulier.

Il faut donc créer un répertoire particulier par application créée. Ce répertoire est appelé "répertoire de travail". Sa détermination (définition du chemin absolu du répertoire de travail) permet à DBD, et ultérieurement à BDE, de gérer rapidement l'ensemble des tables de l'application.

Un utilitaire spécial, accessible par une icône du groupe Delphi, permet de définir le "répertoire de travail" dans lequel DBD générera et stockera les différents objets constituants une base.

Il est néanmoins possible de définir ce répertoire de travail à partir du menu ' Fichier | Répertoire de travail ' .

On peut aussi définir un "répertoire privé" dans lequel DBD stocke des tables temporaires et éventuellement des tables générées en cas d'incident lors de vérification d'intégrité.

10.3.2 Notion d'alias

Il est possible de définir un alias au répertoire de travail. Cet alias sera utilisé ultérieurement dans le développement de l'application et permettra d'éviter d'avoir à fournir le chemin absolu du répertoire de travail.

Alias : MonAppli = c:\MaBase\Tables

La notion d'alias est une notion très souple. Elle permet de développer une application complète en s'affranchissant de la configuration physique de la machine support. Il suffit en effet de configurer l'alias (grâce à l'utilitaire adéquat) pour prendre en compte un chemin d'accès aux différentes tables différent (de celui utilisé lors de la phase de conception mais aussi, éventuellement, en cas de déplacement de l'application sur d'autres machines).

10.3.3 Utilisation de DBD

Il est possible de lancer directement DBD à partir du groupe Windows 'Delphi ' (Menu Démarrer| Programmes|Borland Delphi3) ou à partir de Delphi (menu 'Outils'). Dans ce dernier cas il faut cependant disposer de suffisamment de ressources car le lancement peut être bloqué.

Aspect de DBD à l'ouverture

Au démarrage, DBD propose un ensemble de menus réduit.

Le menu 'Fichier' permet de définir s'il s'agit :

- De créer une nouvelle table (ou une nouvelle requête SQL ou QBE). Dans le cas une table, une fenêtre s'ouvre pour choisir le format de la base (dBase ou Paradox).
- D'ouvrir une table (ou des requêtes SQL ou QBE).

Le premier bouton permet d'ouvrir une table (les deux autres permettent d'ouvrir des requêtes SQL ou QBE existantes).

Si un répertoire de travail a été défini, c'est celui qui apparaît par défaut s'il s'agit d'ouvrir ou sauvegarder une table.

10.3.4 Création d'une table

Lorsque l'on demande la création d'une table, une importante boite de dialogue s'ouvre. C'est à l'intérieur de cette boite que l'on peut définir les différents champs de la table ainsi que les clés, les index, et - si il y a plusieurs tables - les règles d'intégrité référentielle.

On peut aussi définir les champs qui devront être obligatoirement être renseignés, les valeurs minimales et maximales, ainsi qu'un masque de saisie éventuel.

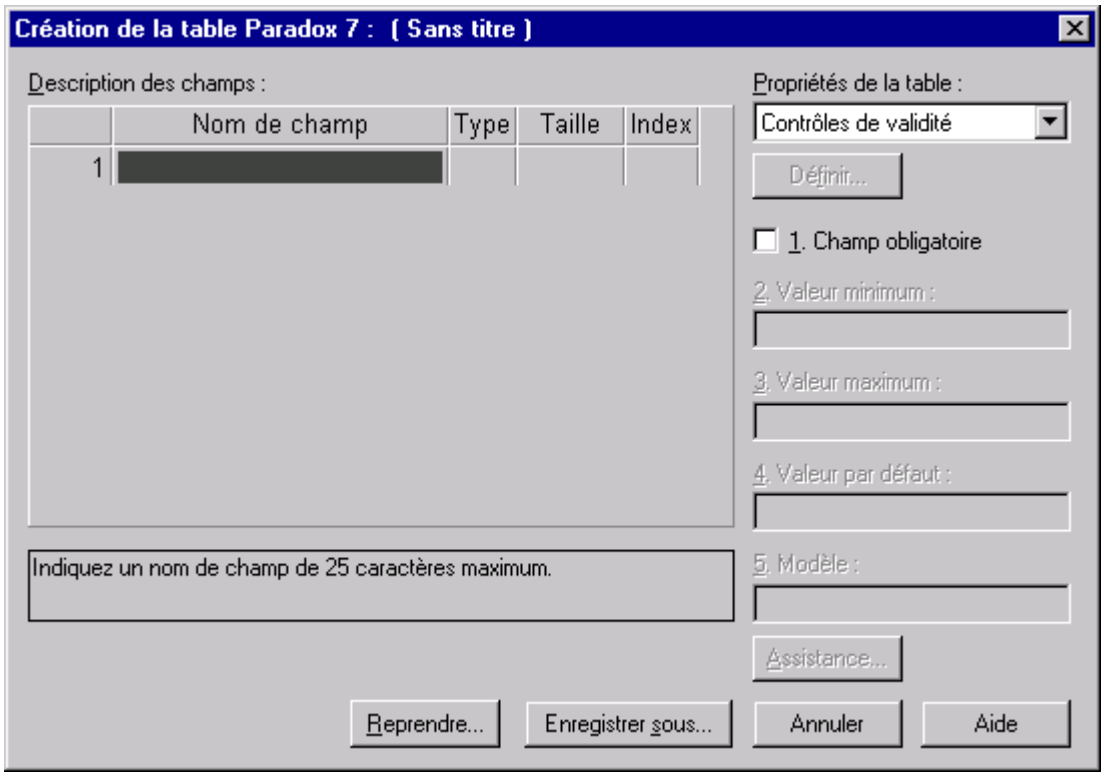
La table est ensuite créée lorsque l'utilisateur lui donne un nom (via "enregistrer sous ...").

Par défaut, DBD crée cette table dans le répertoire de travail (mais il est possible de spécifier un autre répertoire).

Si des clés, index, liens, etc… sont spécifiés, DBD crée de nombreux fichiers internes contenant les informations de gestion correspondantes (c'est pour cela qu'il faut bien spécifier un répertoire de travail particulier pour chaque application).

DBE reconnaît plusieurs types de données lors de la définition des différents champs. Aux types traditionnels (chaînes de caractères, entier, flottant, etc ...) il faut rajouter un type "date", un type "format monétaire" et un type "auto-incrémenté". Ce dernier est très utile lorsque l'on souhaite utiliser un index interne, différent des champs de données, dont on souhaite garantir l'unicité sans avoir à réaliser des algorithmes permettant sa génération..

DBE accepte les espaces dans les noms des champs. Cependant ces espaces ne sont pas acceptés ensuite par Delphi, ce qui provoque de gros problèmes. Il y a donc lieu d'éviter d'utiliser les espaces à ce niveau de la conception des tables.



La fenêtre de conception des tables

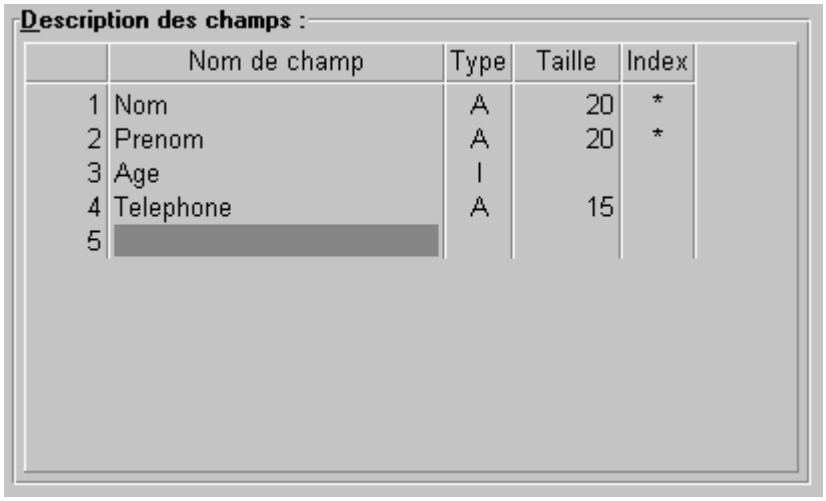
Extension des différents fichiers :

.**cfg**	fichier de configuration (nom d'alias etc...)
.**db**	table au format Paradox
.**dbf**	table au format dBase
.**fam**	liste de tables liées
.**mb**	champs mémos dans une table Paradox
.**px**	index primaire dans une table Paradox
.**qbe**	requête QBE enregistrée
.**sql**	requête SQL enregistrée
.**tv**	paramètres de la fenêtre de la table
.**val**	contrôle de validité
.**xnn**	index secondaire
.**ynn**	index secondaire
.**xgn**	index secondaire composite
.**ygn**	index secondaire composite

▲ **Description des différents champs :**

La création de la table se fait en entrant simplement les différentes informations dans la partie " description de champs " :

- Nom du champ (éviter les caractères accentués ; maximum 25 caractères).
- Type (un appui sur la touche 'espace' affiche la liste des types reconnus).
- Taille (pour les types qui en ont besoin, comme les types alphabétiques).
- Index (permet de définir les clés par double click dans le champ correspondant).



Le numéro qui apparaît à gauche est généré automatiquement. En réalisant des "cliquer-glisser" sur ces numéros, il est possible de modifier l'ordre des champs.

Pour supprimer un champ de la description il faut le sélectionner et appuyer sur les touches 'Ctrl + Suppr'.

Le mot "index" utilisé dans cette boite de dialogue peut induire en erreur car, à ce niveau, il s'agit bien de définir les champs entrant dans la définition de la clé de la table.

▲ **Propriétés de la table :**
Sur la droite de la fenêtre apparaissent des éléments de dialogue permettant de définir certains contrôles sur les tables.

Ces différentes possibilités sont accessibles via une boite de dialogue déroulante dénommée "Propriétés de la table". Par défaut, BDE propose les fonctionnalités suivantes :

Il est possible de déterminer quels seront les champs devant être obligatoirement renseignés par l'utilisateur lors de l'exécution de l'application. BDE ne validera l'entrée des données que si ces champs sont remplis.

Les champs 'Minimum' et 'Maximum' permettent de déterminer les valeurs des données entrées pour les champs sélectionnés.

'Défaut' permet de donner une valeur par défaut à un champ (cette valeur sera celle qui sera validée si l'utilisateur n'en entre pas une autre).

'Modèle' permet de définir des masques de saisie.

Ces modèles sont surtout utiles pour homogénéiser les entrées de données de types date ou numéro de téléphone.

Ils permettent aussi de définir les caractères acceptés dans des champs alphabétiques et permettent les entrées automatiques (le fait d'entrer un caractère ou une séquence donnée génère automatiquement le reste de la donnée).

Il n'est pas vraiment utile de réaliser ces modèles (à la syntaxe assez déroutante et surtout différente de celle utilisée dans les éditeurs de masque de Delphi) dans la mesure où ils ne sont pas pris en compte ensuite par Delphi.

▲ **Autres propriétés :**
La boite de dialogue "Propriétés de la table" permet aussi d'accéder à un ensemble de fonctionnalités permettant de définir :

- Des tables de référence permettant de lier les tables entre elles.
- Des index secondaires.
- Des règles d'intégrité référentielles entre tables.
- Des mots de passe permettant de sécuriser les accès aux tables.

La table de référence permet d'indiquer que les valeurs que l'on entre dans un champ doivent déjà exister dans le premier champ d'une autre table ou de copier des valeurs automatiquement à partir de la table de référence vers une table en cours de modification (remplissage automatique)

Pour établir une liaison plus solide entre deux tables, on définit une relation d'intégrité référentielle.

Un index secondaire est un champ ou un groupe de champs utilisé dans plusieurs cas:

- Pour réaliser une table de correspondance des valeurs du champ spécifié ;
- Pour pouvoir effectuer un ordre de tri différent de celui imposé par la clé primaire de la table ;

On ne peut créer un index secondaire que si une clé primaire a été spécifiée auparavant.

Une table peut posséder plusieurs index secondaires (à la limite on peut définir autant d'index secondaires qu'il y a de champs dans la table).

On peut définir des index secondaires composites.

On ne peut pas créer d'index secondaire sur un champ mémo, mémo formaté, graphique, OLE, logique, binaire ou un champ d'octets.

L'intégrité référentielle est un mécanisme interne à BDE qui sert à assurer qu'une valeur de champ entrée dans une table (dite "table enfant") existe déjà dans un champ spécifié dans une autre table (dite " table parent").

Pour pouvoir définir une règle d'intégrité référentielle entre deux tables celles-ci doivent comporter une clé primaire et se trouver dans le même répertoire.

Lorsque l'on établit une règle d'intégrité référentielle, la règle est bidirectionnelle, ce qui signifie que les entrées ou les modifications des données des deux tables associées à cette règle suivent la règle. La règle d'intégrité référentielle est gérée dans le fichier d'extension .VAL pour chacun de ces fichiers.

▲ **Remarques générales sur l'utilisation de DBD :**
Il faut définir les liens d'abord dans DBD, grâce à ces différentes fonctionnalités, avant de les invoquer lors de la conception de l'application Delphi.

A partir du moment où une table est créée, il est possible de l'ouvrir, sous forme d'un tableau, puis de l'éditer afin d'y entrer des enregistrements.

Cette possibilité est intéressante pour générer facilement des jeux d'essai, lorsque l'on crée une application Delphi, pour y réaliser des modifications et ..... pour tester la validité des fonctions créées sous Delphi.

10.3.5 Utilisation ultérieure de DBD

DBD peut toujours être invoqué à partir de Delphi, alors que le développement de l'application est déjà entamé, pour préciser certains contrôles, pour créer de nouveaux index qui s'avèrent nécessaires voire pour restructurer les tables.

Cependant il faut garder à l'esprit les règles suivantes :

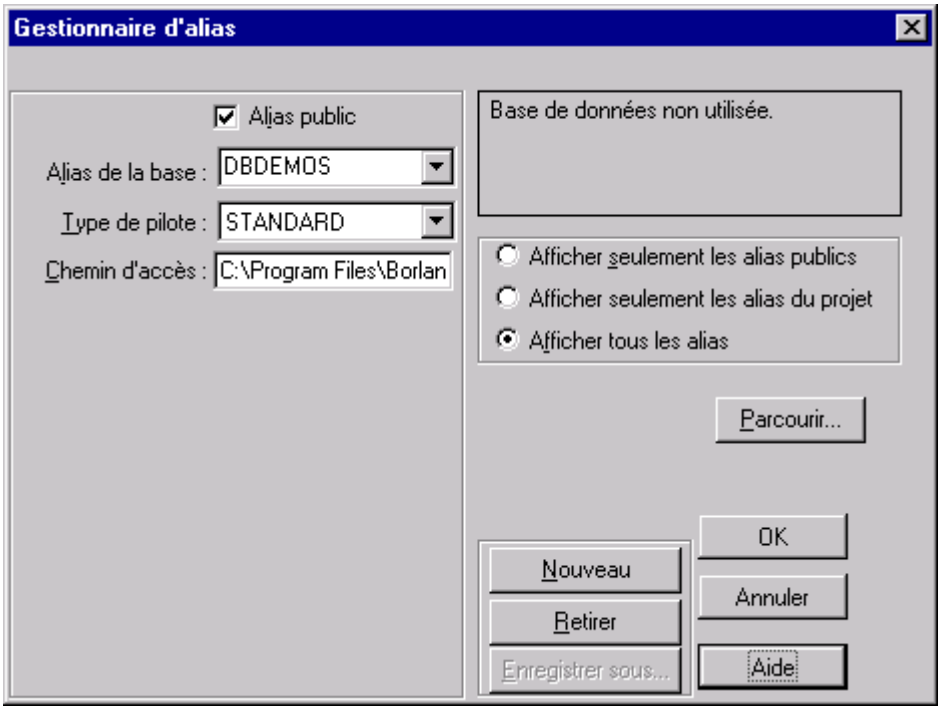
Il est toujours possible de restructurer, ultérieurement, une table. Cela grâce au menu 'Outils | Utilitaires | Restructurer une table' (lorsqu'aucune table est ouverte) ou grâce à la quatrième icône (lorsqu'une table est ouverte).

Un contrôle est réalisé pour vérifier que toutes les données déjà stockées peuvent être récupérées" dans la nouvelle structure de la table. Le cas contraire des tables temporaires sont créées dans le répertoire privé spécifié.

- Lorsqu'une table est ouverte dans un des environnements en vue d'une modification , elle ne peut l'être dans l'autre.
- Par exemple si l'on ouvre, dans DBD, une table afin de la restructurer, alors qu'elle est déjà ouverte dans l'application en cours de conception dans Delphi, un message d'erreur apparaîtra.

10.4 Configuration de BDE

Un autre utilitaire, accessible dans le menu "Outils | Gestionnaire d'alias " permet de créer|modifier des alias.



Les alias sont, on l'a vu, très utiles pour réaliser facilement des applications Delphi importantes. Cependant cela oblige à utiliser les pilotes IDAPI, livrés avec Delphi dans des disquettes de distribution (stockées sur CD ROM).

De fait l'utilisation des alias impose un surcoût d'environ 1 Mo sur disque ainsi qu'une procédure d'installation spécifique de l'application.

Pour des applications peu importantes il est préférable d'indiquer "en dur" les chemins d'accès aux différentes tables. C'est plus contraignant, au niveau de la programmation, lors de la réalisation de l'application mais cela permet de créer des applications plus légères et plus faciles à installer.

▲ **L'utilitaire de configuration de BDE :**
Les différents paramètres de configuration sont stockés dans le fichier IDAPI.CFG.

10.5 Distribution d'applications Delphi

Une application Delphi utilisant BDE peut être "redistribuée". C'est à dire qu'un programmeur possédant une licence de Delphi (version "Desktop" ou version "Client / serveur") peut créer une application et la distribuer, à titre payant, à un nombre indéterminé d'exemplaires sans avoir à payer de royalties à Borland. Il faut néanmoins qu'il respecte certaines règles et qu'il utilise toutes ou parties des disquettes de distribution stockées dans un répertoire particulier du CD ROM de Delphi (les disquettes contiennent des modules permettant d'installer BDE, Report Smith et SQL Links).

Le répertoire REDIST est divisé en trois sous-répertoires contenant les fichiers compressés redistribuables et les utilitaires d'installation :

BDEINST - Borland Database Engine
SQLINST - SQL Links pour Windows (Delphi Client/Serveur uniquement)
RPTINST - ReportSmith Runtime

Les utilitaires d'installation de ces programmes configurent la commande PATH du DOS des systèmes cibles (et les autres définitions système) .

Déployer une application Delphi signifie la confier à l'utilisateur final et lui fournir tous les éléments lui permettant de l'installer et de l'utiliser dans son environnement de travail en accord avec la législation en vigueur.

Une application Delphi peut exiger l'installation de toute ou partie de :

Le fichier .EXE de l'application (et toutes les DLL spéciales éventuellement).
Borland Database Engine (BDE), si l'application est une application de bases de données.
Le cas échéant le support de SQL Links,d'InterBase SQL Link, d'InterBase Local, si l'application utilise InterBase Local ou le support du Runtime de ReportSmith.

Le répertoire BDEINST contient l'ensemble des utilitaires nécessaires à la distribution de BDE. Il faut créer deux disquettes contenant les différents fichiers et les fournir à l'utilisateur (la possession de ces deux disquettes le rend habilité à utiliser l'application).

Le fichier INSTALL.EXE de la première disquette est l'utilitaire d'installation. Il installera tous les fichiers de BDE. Le programme d'installation fera également les modifications de WIN.INI, de la base de registre, et des autres paramètres de configuration du système nécessaires pour que les utilisateurs puissent installer le BDE en toutes circonstances.

La licence Delphi exige que tous les fichiers du BDE redistribuable soient installés avec l'utilitaire d'installation. L'application peut ne pas avoir besoin de tous ces fichiers, et l'on peut ensuite supprimer les fichiers inutiles afin de préserver l'espace disque. Mais si tous les fichiers nécessaires ne sont pas conservés, les applications existantes risquent de ne pas fonctionner.

Rappel :
Pour une application Delphi n'utilisant pas le moteur de bases de données BDE, il n'y a pas lieu de préparer de disquettes de distribution. Mais dès lors que l'on souhaite distribuer une application utilisant ce moteur - même si elle n'utilise pas les alias - il faut fournir avec les disquettes contenant l'exécutable de l'application et les tables, les disquettes de distribution pour être en règle avec les lois en vigueur (en aucun cas les sources ne doivent être fournis).

11 Création d'une application de base de données

Delphi propose un grand nombre de composants permettant d'accéder aux bases de données (construites au préalable par DBD ou par d'autres SGBDR).

En fait il est possible d'utiliser la puissance de Delphi selon deux manières totalement différentes :

- Soit on utilise pleinement les composants proposés, ce qui se traduit par une facilité certaine de conception. Mais les applications résultantes sont assez lourdes et surtout nécessitent l'utilisation de pilotes qu'il faut ensuite livrer avec l'application. On parlera dans ce cas de méthode de développement "statique".
- Soit on utilise plus particulièrement des puissantes possibilités de programmation qui permettent de créer et de gérer les tables sans utiliser certains des composants prédéfinis. La conception de l'application est plus délicate mais elle est plus rapide à l'exécution et plus "légère". On parlera dans ce cas de méthode de développement "dynamique".

- Dans la suite du cours, c'est surtout le développement statique qui va être évoqué.

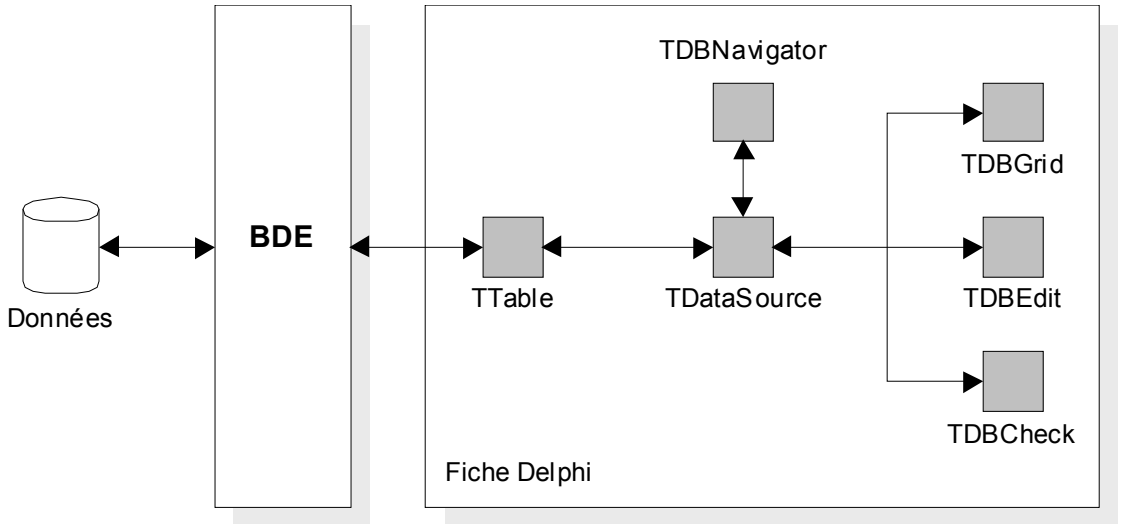
11.1 Architecture des bases de données Delphi

Delphi utilise des composants spécialisés permettant de gérer les transferts de données entre le disque et l'interface utilisateur. Chaque composant a un rôle bien défini est il est important de connaître son action :

- Les composants TTable et TQuery, invisibles, permettent de se connecter aux données stockées via BDE. Ils permettent de connaître la structure d'une table ou d'une requête SQL.
- Les composants TDBGrid, TDBEdit, TDBCheck, etc ... permettent de constituer une interface graphique orientée "base de données". Certains de ces composants sont similaires à des composants standards déjà présentés auxquels certaines propriétés et méthodes ont été rajouté pour assurer la connexion aux bases de données.

Entre les deux séries de composants, un composant spécifique, TDataSource, permet de réaliser la connexion entre les composants invisibles et les composants de l'interface.

Un composant particulier, TDBNavigator, permet de se déplacer dans une table.



Liens entre les différents composants proposés par Delphi : il faut considérer le composant TDataSource comme le "tuyau" permettant la connexion entre les composants représentants virtuellement la couche logicielle BDE (composants TTable ou TQuery) et les composants de l'interface. Tous les problèmes de bufférisation des transferts de données, de contrôle des accès, etc… sont gérés automatiquement par ces composants.

Les composants assurant la gestion interne des tables sont regroupés dans l'onglet 'AccesBD' de la palette de composants. Ceux permettant de créer l'interface graphique et de manipuler les données sont placés dans l'onglet 'ContrôleBD '.

11.2 Construction d'une application de gestion de base de données

En mode statique la construction d'une application chargée de gérer une base de données se réduit pour une grande part à paramètrer les composants lors de la création de l'interface. Cela fait, lors de l'exécution, tout est opérationnel.

- On peut par ailleurs utiliser un expert particulier pour construire des bases simples. Mais ses possibilités - en dehors de l'aspect pédagogique- sont, somme toute, relativement limitées.

- Il est rappelé que la structure complète de la base doit avoir été définie au préalable et que les différentes tables (ainsi que les clés utilisées et les liens éventuels qui les unissent) doivent avoir été créés dans DBD.

11.2.1 Différents composants permettant l'accès à BDE

Dans un premier temps seuls les composants TTable et TDataSource nous intéressent :

▲ **Le composant TTable :**
Ce composant fournit un accès "vivant" aux tables de la base de données, via BDE.

On dit "vivant" car, dès la phase de conception, le composant est actif et permet de visualiser le contenu de la table gérée.

Il s'agit d'un composant "non visuel" qu'il faut déposer sur la feuille correspondant à l'interface graphique puis qu'il s'agit de configurer.

Dans la liste des propriétés qui suit, les propriétés accessibles à la conception sont soulignées :

Active	Indique si la table est ouverte ou non. Le fait de basculer Active à True ouvre la table est et équivalent à : Table1.Open ;
DataBaseName	Indique l'alias ou le chemin d'accès à la table
Fieldefs	Donne des renseignements sur les différents champs de la table.
Fields [i]	Permet d'accéder à un champ particulier par son indice (base 0).
IndexName	Spécifie un index secondaire. Celui-ci doit avoir été définit dans DBD.
MasterField	
MasterSource	Utilisé dans le cas de tables liées.
Modified	Indique si un champ de l'enregistrement courant a été modifié.
ReadOnly	Empêche l'utilisateur de modifier les données.
RecordCount	Indique le nombre d'enregistrements.
State	Indique l'état courant de la table.
TableName	Nom de la table avec laquelle le composant est lié.

En général on bascule la propriété Active à True pendant toute la phase de conception pour pouvoir visualiser le contenu de la table. Cependant, il est préférable qu'une table ne soit ouverte qu'à bon escient dans l'application. De fait, une fois la configuration réalisée, on remettra cette propriété à False et on ouvrira explicitement la table dans le programme (Table1.Active := True ou Table1.Open).

Pour plus de facilité, réaliser la configuration du composant dans l'ordre suivant:

- Indiquer son nom logique. On utilisera pour cela le nom de la table à laquelle le composant est lié (Par exemple si la table est la table CLIENTS on appellera le composant TClients).
- Initialiser la propriété DataBaseName avec le nom d'alias souhaité. Dès que cela est fait, la propriété TableName affiche le noms de toutes les tables contenues dans la base. Il n'y a plus alors qu'à sélectionner la bonne table.

▲ Le composant TDateSource :

Ce composant est l'interface entre un composant gérant l'accès aux données (comme TTable) et les différents composants réalisant l'affichage des données.

Comme TTable, c'est aussi un composant non-visuel.

Une source de données n'est associée qu'à un composant de gestion de données, mais on peut lui connecté autant de composants de l'interface graphique qu'on le désire.

Ses principales propriétés sont :

Active	Indique si le composant est actif ou nom.
DataSet	Indique le composant de gestion de données auquel TDataSource est connecté.
State	Indique l'état du composant de gestion de donnée.

Dans la mesure du possible on donne au DataSource un nom rappelant celui du composant TTable auquel il est connecté (exemple : DSClients est connecté à TClients).

Il est possible de désactiver le composant TDataSource lorsque l'on souhaite éviter les rafraîchissements continuels des composants graphiques qui lui sont connectés. Cela afin d'accélérer les recherches dans la table.

11.2.2 Composants constituant l'interface utilisateur dérivés de composants usuels

Les différents composants qui permettent la réalisation de l'interface utilisateur sont, pour la plupart, des composants dérivés des composants déjà étudiés dans les parties précédentes (par exemple : TDBEdit est dérivé de TEdit, etc...). Leur principale spécificité est qu'ils sont "sensibles aux données". C'est à dire qu'ils sont en mesure d'afficher le contenu d'un champ d'une table ou de modifier ce dernier.

Pour ce faire ils possèdent deux propriétés supplémentaires :

Datasource Indique le composant TDataSource sur lequel le composant est connecté
DataField Spécifie le champ précis de la table auquel le composant accède.

Les différents composants, affichent, dès la phase de conception (dans le cas où toutes les connexions sont réalisées et que le composant TTable est actif), les champs qui correspondent du premier enregistrement de la table.

A l'exécution ils affichent les différentes données qui correspondent à l'enregistrement courant.

On a ainsi les composants suivants :

TDBEdit Permet l'affichage ou la modification d'un champ d'une table.
TDBText Ne permet que l'affichage d'un champ d'une table.
TDBListBox Propose un choix d'options dont la validation deviendra la valeur du champ dans l'enregistrement courant.
TDBComboBox Idem sous forme d'une ComboBox.
TDBCheckBox Permet d'afficher sous forme de case à cocher des champs booléens.
TDBRadioGroup Permet d'afficher le contenu de certains champs sous forme de choix exclusifs.

Les différents composants ont par ailleurs les comportements suivants :

TDBEdit : A l'exécution, les données modifiées sont copiées dès que l'utilisateur a appuyé sur 'Tab' ou a changé de zone d'édition. Si l'utilisateur appuie sur 'Echap' avant d'appuyer sur 'Tab' les modifications sont abandonnées et la valeur du champ reprend la valeur initiale.

TDBCheckBox : Possède la propriété ValueChecked qui indique quelles sont les chaînes de caractères qui feront que la case sera sélectionnée.

Exemple :

```
DBCheckBox1.ValueChecked := 'True;Oui ';
{ La case à cocher sera "cochée" si le champ est égal à
True ou à Oui }
```

TDBRadioGroup : Possède les propriétés spécifiques suivantes :

- Items qui permet de donner une légende à chaque bouton.

- Values qui est la valeur qui sera mise dans le champ lorsque le bouton correspondant sera sélectionné.

(Il est possible que les propriétés Items et Value correspondent aux mêmes chaînes).

Exemple :

Création d'une boîte groupe radio connectée à une table. Le champ contient les valeurs 'O', 'N' ou 'P'. Les intitulés de boutons radio, 'Oui', 'Non' et 'Peut-être', correspondent aux trois chaînes ajoutées dans la propriété tableau Items. Les valeurs effectivement stockées dans le champ sont 'O', 'N' et 'P' ; elles sont ajoutées dans la propriété Values:

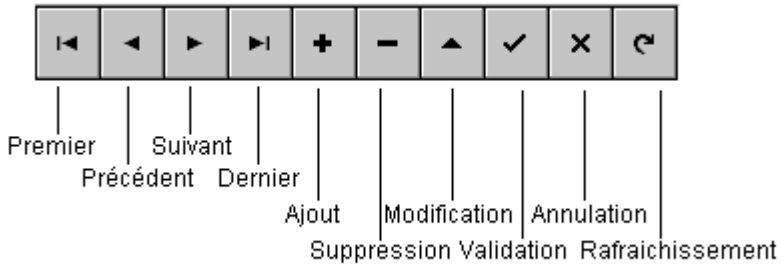
```
procedure TForm1.FormCreate (Sender: TObject );
begin
    with DBRadioGroup1 do
    begin
        Items.Add ('Oui' );
        Items.Add ('Non' );
        Items.Add ('Peut-être' );
        Values.Add ('O' );
        Values.Add ('N' );
        Values.Add ('P');
    end;
end;
```

{ Lors de l'exécution, trois boutons radio apparaissent dans la boîte groupe avec les différentes propriétés 'Items' comme légende. Si l'enregistrement courant dans l'ensemble de données contient l'une des valeurs définies dans la propriété 'Values', le bouton radio correspondant est activé. Lorsque l'utilisateur sélectionne un bouton radio, la chaîne correspondante dans la propriété 'Values' est placée dans le champ }.

11.2.3 Composants particuliers à la gestion des bases de données

En plus des composants décrits précédemment, Delphi fournit d'autres composants permettant de créer des interfaces évolués :

▲ **Le composant TDBNavigator :**
Ce composant est constitué de plusieurs boutons permettant la navigation aisée au sein d'une base de données. Il permet aussi la création, l'effacement et la modification de l'enregistrement courant mais ces possibilités ne sont guère sécurisées.



Pour fonctionner, le Navigateur doit être connecté à la DataSource associée à une table.

▲ **Modification de l'apparence du navigateur :**
On utilise rarement le navigateur (et une grille) pour réaliser des insertions et suppressions de champs. Il n'y a pas assez de contrôles de réaliser et l'entrée des données n'est pas très pratique et peut générer des erreurs (on préférera utiliser une fenêtre spéciale contenant l'ensemble des champs de saisie sous forme de formulaire).

On utilise donc principalement le navigateur pour réaliser des déplacements à l'intérieur de la base. Il faut donc éliminer les boutons inutiles (cas de manipulation dangereuse) et ne conserver que ceux que l'on souhaite mettre à la disposition des utilisateurs. Pour cela on sélectionne le composant Navigateur et on affiche la liste des options accessibles via la propriété VisibleButtons. Il suffit alors de faire basculer à False les boutons que l'on ne veut pas afficher. L'effet est immédiat dans la fenêtre de conception et il n'y a plus qu'à redimensionner le navigateur.

▲ **Connaissance du bouton utilisé :**

Le défaut majeur de ce composant est qu'il ne permet pas de connaître quel est le bouton qui est activé. En fait, si on étudie le gestionnaire d'événement créé en association avec l'événement OnClick, on découvre qu'il crée une variable appelée Button qui permet de connaître le bouton qui a été "cliqué" :

```
procedure TForm1Navigator1Click (Sender : TObject ; Button :TNavigateBtn);
    var
        BtnName : string ;
    begin
        case Button of
            nbFirst : BtnName := 'nbFirst ' ;
            nbPrior : BtnName := 'nbPrior' ;
            ...
            NBPost : BtnName := 'nbPost' ;
        end ;
        MessageDlg (' Bouton cliqué : ' + BtnName + ' !! ' ,
mtInformation, [mbOk], 0) ;
end ;
```

▲ **Le composant TDBGrid :**

Ce composant est le plus utilisé dans le cadre de la gestion de base de données car il permet d'afficher le contenu intégral d'une table sous forme d'un tableau. Il dispose pour ce faire de beaucoup de possibilités.

Néanmoins, s'il permet la création de nouveaux enregistrements et l'édition (en utilisation conjointe avec un composant TDBNavigator) ces possibilités ne seront pas souvent mises en œuvre car elles demandent beaucoup d'attention de la part de l'utilisateur. Celui préférera utiliser des écrans d'accueil de types formulaires de saisie.

Dans le cas où l'on effectue de l'édition avec un tel composant il faut savoir que les données insérées ne sont transférées que lorsqu'il y a changement d'enregistrement.

Si une des données envoyées ne peut être acceptée, Delphi provoque une exception et les modifications ne sont pas appliquées.

Les propriétés les plus importantes du composant sont :

Field [i] : Permet d'accéder au i° champ de la grille ;
FixedColor : Spécifie la couleur de la grille ;
Options : Les nombreuses options permettent de modifier l'aspect de la grille.

Read Only : Permet de déterminer si les données ne sont qu'affichées ou si l'on peut les modifier.

Il est possible de contrôler le dessin de chaque cellule en positionnant la propriété DefaultDrawing à False. Il faut alors créer des fonctions attachées à l'événement OnDrawDataCell et utilisant toute une panoplie de propriétés et méthodes orientées "graphisme" .

Il est difficile de savoir quelle est la ligne et quelle est la colonne que l'on vient de sélectionner dans une grille. La petite routine qui suit permet de résoudre ce problème et peut être adaptée à de nombreux cas.

Cette routine est basée sur l'événement DrawDataCell. Elle indique la colonne (ça c'est facile) mais surtout la ligne qui viennent d'être sélectionnées.

Le gestionnaire d'événement associé est alors :

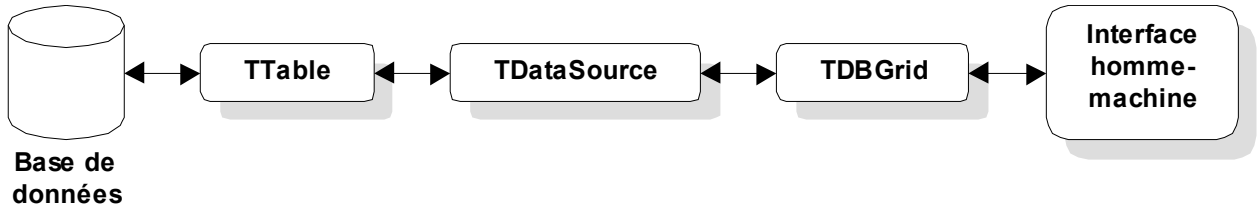
```
var
    Row, Col : integer ;
    { variables globales pour être utilisées ailleurs dans le
programme }

procedure TForm1.DBGrid1DrawDataCell (Sender: TObject;
const Rect: TRect ; Field: TField; State: TGridDrawState );
var
    RowHeight : Integer;
begin
    if gdFocused in State then
    begin
        RowHeight := Rect.Bottom - Rect.Top;
        Row := (Rect.Top div RowHeight)- 1;
        Col := Field.Index;
    end;
end ;
```

Attention : Row et Col indiquent les indices de la ligne et de la colonne sélectionnées. La première case en haut à gauche est donc à Row = 0, Col = 0. En tenir compte.

11.2.4 Création d'une application

Pour réaliser une application, il suffit de positionner les différents composants, ainsi que les différents composants de l'interface utilisateur et de les connecter entre eux.

On a alors le schéma suivant :



- Les propriétés DataBaseName et TableName permettent de connecter TTable à la base de données.
- La propriété DataSet du composant TDataSource le connecte à laquelle il est lié.
- La propriété DataSource des différents composants graphiques les connecte à la source de données auxquels ils sont liés.

Exemple :

Ouvrir un projet.
Initialiser un composant TTable avec les propriétés :

DatabaseName = DBDEMOS (alias prédéfini).
Name = TClients
TableName = CLIENTS.DBF

Initialiser un composant TDataSource avec les propriétés :

DataSet = TClients
Name = DSClients

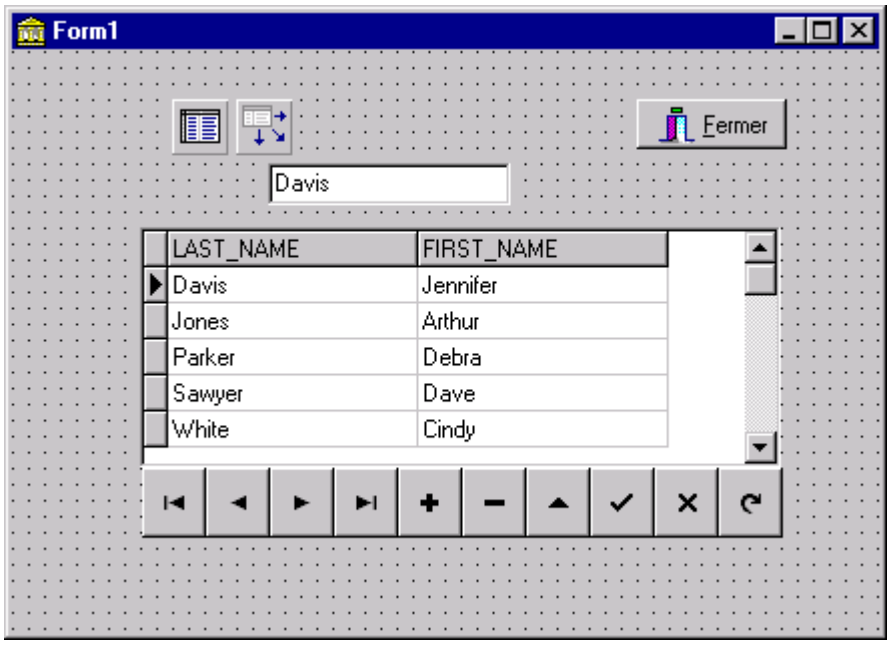
Déposer et initialiser les composants suivants :

Un composant TDBGrid
DataSource = DSClients
Name = DBGClients

- Un composant TDBEdit
  DataSource = DSClients
  Name = DBEClientsName
  DataField = LAST_NAME { ce composant n'affiche qu'un champ }
- Un composant TDBNavigator
  DataSource = DSClients
  Name = TDBClients

En positionnant la propriété Active du composant Table True, les données de la table apparaissent dans les différents clients. Si l'on exécute ce programme, l'accès à la base de données est opérationnel. On peut se déplacer et même insérer ou supprimer des enregistrements (mais ce n'est pas conseillé de cette manière).

Il faut faire attention aux différents noms logiques donnés aux composants. Pour une base manipulant plusieurs tables il faut absolument se fixer certaines règles. Ici on a choisi la règle suivante : chaque nom de composant commence par les initiales majuscules de son type et est suivi par le nom de la table auquel il est connecté.



Apparence à l'issue de la première phase de conception

11.2.5 Les composants TDBLoockupComboBox et TDBLoockupListBox

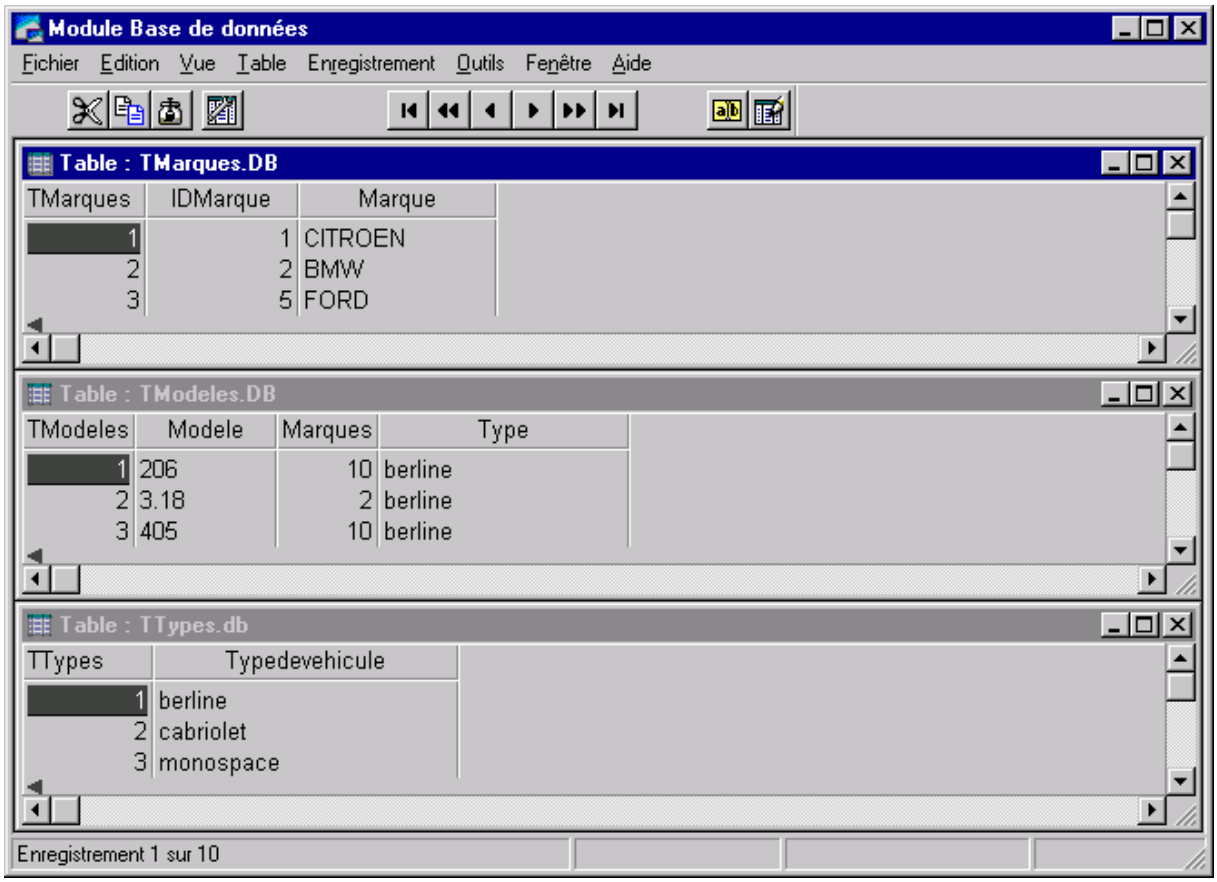
Ces composants sont similaires aux composants précédents mais la liste de valeurs proposées à l'utilisateur est déterminée dynamiquement à partir d'un deuxième ensemble de données. Ces composants remplacent les TBDLookupCombo et TDBLookupList de DELPHI 1 et 2 qui sont néammoins toujours disponibles dans l'onglet Win 3.1.

Pour ces composants, de nouvelles propriétés doivent être renseignées :

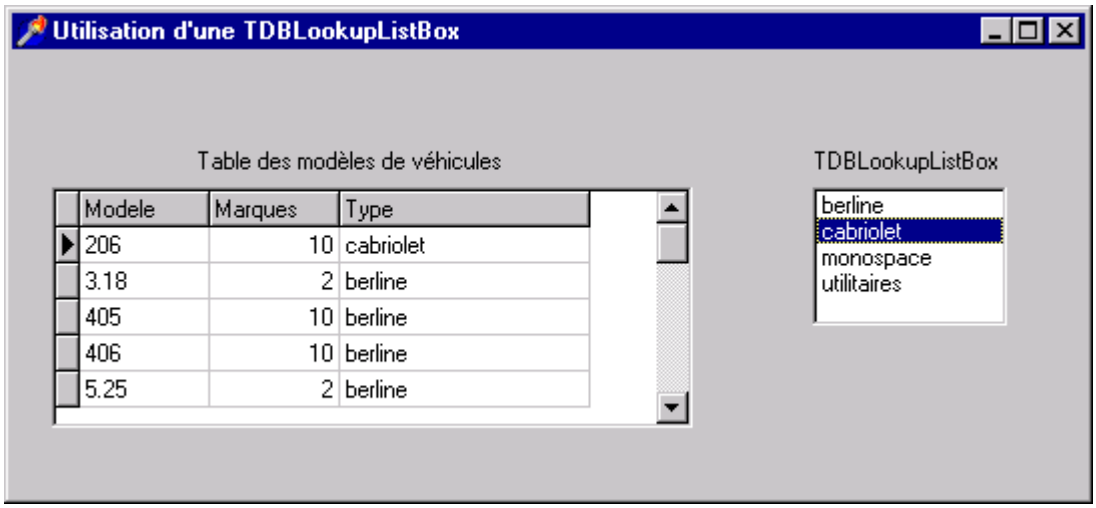
ListField	identifie les champs dont les valeurs sont affichées dans le contrôle de référence.
ListFieldIndex	indique quel champ de la propriété ListField est utilisé pour la recherche incrémentale.
ListSource	identifie une source de données pour les données affichées dans le contrôle de référence.
DataField	identifie le champ dont la valeur est représentée par le contrôle de référence.
DataSource	lie le contrôle de référence à l'ensemble de données contenant le DataField.
KeyField	identifie le champ de l'ensemble de données ListSource qui doit correspondre à la valeur du champ DataField.

Exemples:
Soient les tables suivantes:

Tmarques contenant une marque de véhicule
Tmodeles contenant les modèles de véhicules
Ttypes contenant les différents types de véhicule



Utilisation du composant TDBLookupListBox:

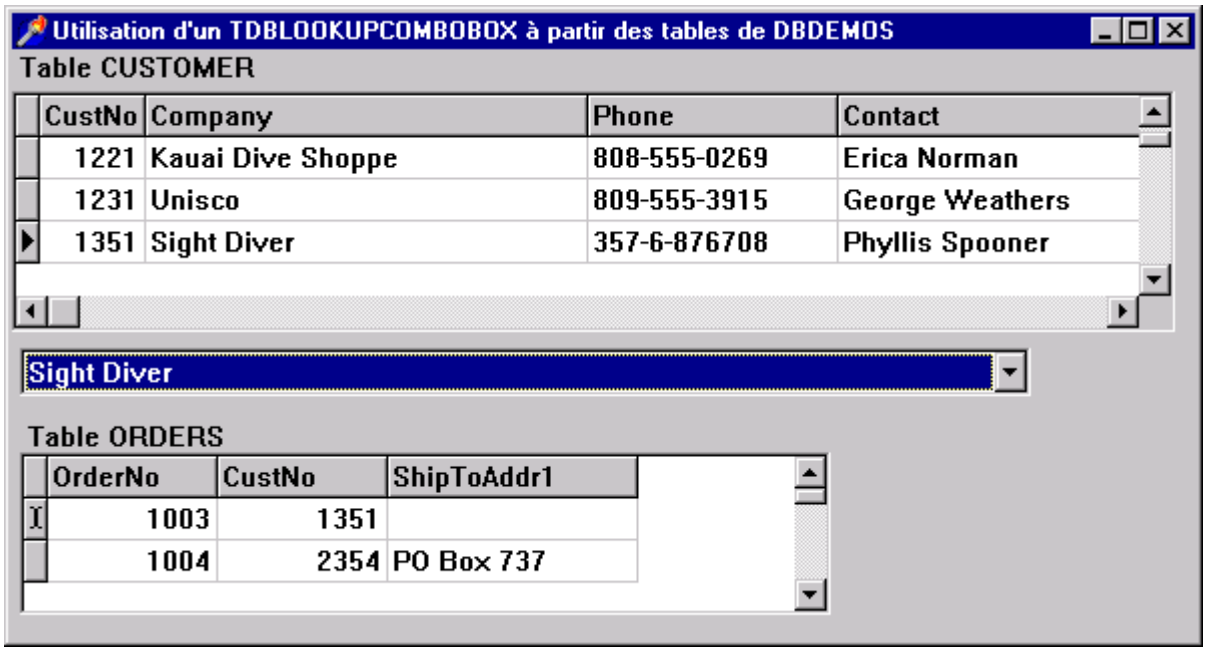


Dans le composant TDBLookupListBox, il suffit de renseigner les champs suivants:

ListSource	DataSourceType
ListField	Typedevehicule
KeyField	Typedevehicule
DataSource	DataSourceModeles
DataField	Type

pour obtenir la mise à jour immédiate du champ type de la table Tmodeles en cliquant simplement sur les valeurs de la TDBListBox.
Remarque: la mise à jour de la table est effectuée dès que l'on se déplace dans cette dernière ou par un **post** explicite.

Utilisation du composant TDBLookupComboBox:



Le composant TDBLookupComboBox est renseigné de la façon suivante:

ListSource	DSCustomer
ListField	Company;CustNo
KeyField	CustNo
DataSource	DSOrder
DataField	CustNo

Mêmes remarques que pour la TDBLookupListBox en ce qui concerne la mise à jour de la table.

Si l'on veut que le deuxième ensemble de données soit en fait le même que l'ensemble initial il faut utiliser deux nouveaux composants TTable et TDatasource pour réaliser l'opération.
Pour initialiser une combo-box avec les différentes marques contenues dans la table TMarques, il faut :

Utiliser deux composants TTable dont la propriété TableName = Marques.db.
Utiliser deux composants TDataSource, chacun connecté sur un composant TTable.
Utiliser un composant TDBLoockupComboBox dont les différentes propriétés sont initialisées comme suit :

DataSource	DataSource1
DataField	Marque { ce qui correspond à une TDBComboBox traditionnelle }
ListSource	Datasource2
KeyField	Marque
ListField	Marque

Rendre les deux tables actives.

A l'exécution, la liste déroulante est initialisée avec les différentes marques contenues dans la table.

11.3 Autres fonctionnalités offertes

11.3.1 Tables liées

Il est possible de réaliser sans aucun codage le lien entre deux tables.

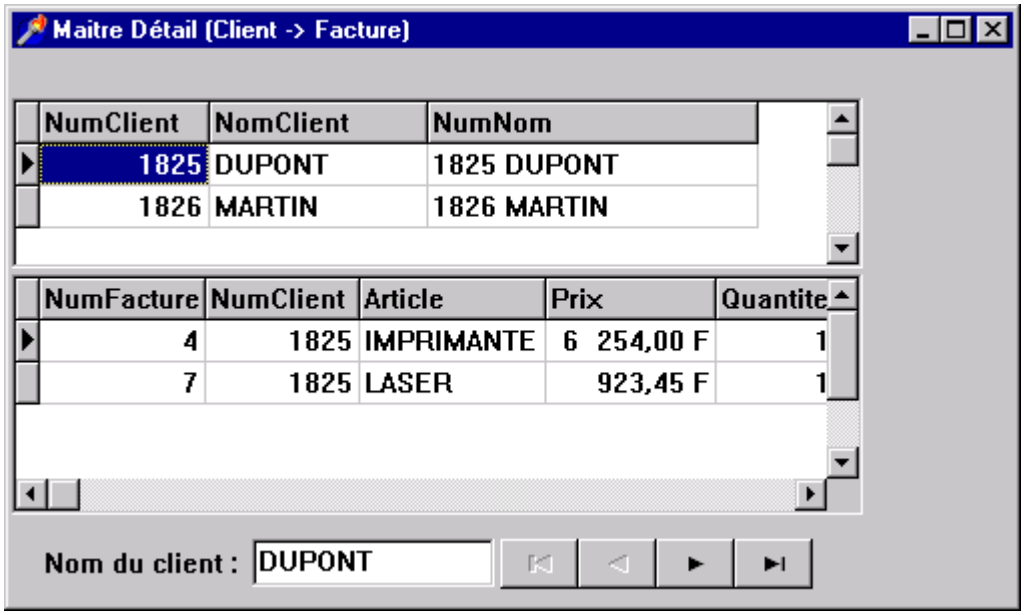
Pour ce faire, il faut que les deux tables aient été liées au préalable au sein de BDE.

Pour lier deux tables (dans le cadre d'une relation "1 à n ") il suffit de préparer les deux tables comme indiqué précédemment et de laisser les propriétés Active = False.

Une fois cela réalisé il faut initialiser, dans la table dépendante les propriétés suivantes :

MasterSource Indique le composant DataSource correspondant à la table "maître".
MasterField Indique le champ assurant le lien entre les deux tables (Il ne peut pas être choisi, il est proposé par la boite de dialogue après interrogation de BDE. Si le lien n'a pas été défini dans BDE il n'y a donc pas de possibilité d'initialiser ce champ).

Une fois cela réalisé on peut mettre les propriétés Active = True et exécuter l'application. Chaque fois que l'on clique sur un enregistrement de la table maître, les enregistrements liés dans la table "fille" sont affichés automatiquement.



11.3.2 Consultation de la structure d'une base

Il est possible que l'on ne sache pas où est localisée une base que l'on doit utiliser. On peut demander à l'utilisateur d'indiquer le chemin de la base (grâce au composant OpenDialog par exemple). Il faut ensuite vérifier que cette base est bien celle que l'on cherche à utiliser. Pour cela, on peut vérifier la structure de la base (nature et nombre des champs).

- Le premier test consiste à vérifier le nombre de champs qui composent la base de données. On utilise la méthode Count de l'objet FieldDefs. Concrètement, nous aurons quelque chose comme :

```
Table := TTable . Create (Application);
...
If (Table.FieldDefs . Count <> 5)Then
      MessageDlg ('La base n''est pas valide : elle doit ' +
       'contenir 5 champs...', mtInformation, [mbOk], 0)
Else
Begin
      ... { Suite des tests ... }
End ;
```

- Un autre test consiste à vérifier les caractéristiques de chaque champ. Il est possible de vérifier trois caractéristiques : le nom du champ, son type et sa taille. On accède aux différents champs par la propriété Items[i] .

```
...
With Table. FieldDefs . Items [ 0 ] Do
Begin
      If (Not ((Name = 'Nom')And (DataType = ftString)
            And (Size = 20 ))) Then
            MessageDlg (' Le premier champ de la base n''est ' +
             'pas valide...', mtInformation, [mbOk],0) ;
End ;
```

{ En travaillant sur Table.FieldDefs.Items [0], nous pouvons accéder aux caractéristiques du premier champs de la base }.

11.3.3 Effacements

Pour effacer d'un seul coup tous les enregistrements d'une table on peut appeler la méthode EmptyTable du composant TTable.

- L'appel de la méthode DeleteTable efface physiquement la table.

12 Accès aux données

On a vu qu'il été possible d'utiliser les composants TDBGrid et TDBNavigator pour pouvoir construire rapidement des petites applications de gestion de base de données. Mais ces deux composants - utilisés seuls - révèlent assez rapidement leurs limites : s'ils permettent de lister facilement le contenu d'une table et de s'y déplacer aisément, il faut utiliser les autres composants et surtout ne pas hésiter à réaliser des fonctionnalités d'accès aux données pour créer des applications de type professionnel.

Dans tous les cas, il faut prendre en considération le fait que Delphi gère les différentes tables d'une base en mettant en œuvre, pour chacune d'entre elles, le principe de l'enregistrement actif : c'est celui qui apparaît marqué d'un sélecteur dans une grille, c'est aussi celui qui apparaît dans les autres composants (TDBEdit entre autres). Toutes les demandes de modification dans la table s'appliquent à cet enregistrement actif (modification, suppression, etc…).

12.1 Gestion d'un ensemble de données

12.1.1 Modes de gestion

Les différents ensembles de données (table ou requête SQL) d'une base de données sont caractérisées par leur état, également appelé mode.

Une application peut placer les différents ensembles de données dans des modes différents en appelant une méthode appropriée.

Les modes possibles sont les suivants :

Mode	Description
Inactive	L'ensemble de donnée est fermée
Browse	Etat par défaut à l'ouverture. Il permet de visualiser les données et de se déplacer au sein de l'ensemble de données mais n'autorise pas les modifications ni les ajouts.
Edit	Permet la modification de l'enregistrement actif
Insert	Permet l'insertion d'une nouvelle ligne à l'emplacement courant
Setkey	Permet à certaines méthodes (FindKet, GoToKey, etc ...)de rechercher des valeurs dans l'ensemble de données.
CalcFields	Mode utilisé uniquement avec des champs calculés.

12.1.2 Déplacement dans un ensemble de données

Pour parcourir un ensemble de données, BDE met en place un curseur interne (matérialisé dans une grille par un sélecteur). C'est ce curseur qui définit l'enregistrement actif que l'on visualise ou que l'on édite en vue de modifications.

Les différents ensembles de données peuvent être parcourus à l'aide des méthodes suivantes:

Méthode	**Action**
First	Déplace le curseur sur le premier enregistrement
Last	Déplace le curseur sur le dernier enregistrement
Next	Déplace le curseur à l'enregistrement suivant
Prior	Déplace le curseur à l'enregistrement précédent
BOF	Renvoie True lorsque le curseur est au début de l'ensemble de données
EOF	Renvoie True lorsque le curseur est à la fin de l'ensemble de données
MoveBy (n)	Déplace le curseur de n enregistrements (n peut être positif ou négatif).

Exemples :

```
Table1.Open ; { rend la table active }
Table1.Last ; { positionne le curseur à la fin de la table }
```

BOF ne renvoie True que s'il est fait explicitement appel à First ou quand un appel à Prior échoue (même principe pour EOF).

Exemple :

```
TTable.Open ;
TTable.First ;
While not TTable.EOF do
begin
      < actions > ;
      TTable.Next ;        { ne pas oublier !!!! .....}
end ;
```

{ Code très répandu qui réalise une action donnée sur tous les enregistrements jusqu'à l'arrivée en fin de table }

Lorsque l'on réalise une action itérative sur tous les enregistrements d'une table, l'application est ralentie par toutes les actions de rafraîchissement de l'écran qu'elle doit réaliser.

Il faut donc penser à inhiber le composant TDataSource lié à la table pendant toute la durée du traitement et de ne le réactiver qu'à l'issue :

```
DSMarque.Enabled : = False ;
...
      { traitement précédent }
....
DSMarque.Enabled := True ;
```

12.1.3 Modification des états

Différentes méthodes permettent de faire basculer un ensemble de données d'un mode à l'autre.

Méthode	**Action**
Open	Ouvre l'ensemble de données (*)
Close	Ferme l'ensemble de données
Cancel	Annule l'opération en cours et place l'ensemble de données en mode Browse
Edit	Place l'ensemble de données en mode Insert
Append	Positionne le curseur à la fin de l'ensemble de données et le place en mode Insert
Post	Expédie l'enregistrement nouveau à la base de données et place l'ensemble de données en mode Browse
Delete	Supprime l'enregistrement en cours et place l'ensemble de données en mode Browse

(*) Dans le cas ou l'ensemble de données n'a pas sa propriété Active = True lors de la phase de conception.

La méthode Post, qui en fait actualise les tables, joue un rôle primordial et il faut bien comprendre son mécanisme :

En mode Edit, Post modifie l'enregistrement en cours;
En mode Insert, Post insère un nouvel élément.

Les méthodes First, Next, Prior, Last génèrent un appel implicite à Post si l'ensemble est en mode Edit ou Insert.

Les méthodes Insert et Append effectuent elles aussi des appels implicites à Post.

Dans les autres cas il faut appeler Post explicitement.

Post n'est pas appelée implicitement par Close. Il faut utiliser l'événement BeforeClose pour expédier explicitement les données en attente.

La méthode Cancel peut défaire les modifications apportées à un enregistrement (tant qu'un appel à Post n'a pas été réalisé explicitement ou implicitement).

On a donc les lignes de code suivantes :

```
TTable.Edit ;  { mise en mode Edition }
< Action sur l'enregistrement >
TTable.Post ;  { actualisation de la table }
```

On pourrait aussi bien avoir :

```
TTable.Edit ;        { mise en mode Edition }
< Action sur l'enregistrement >
TTable.Next ;        { actualisation de la table par appel
                     implicite à Post }
```

Au niveau d'une grille, le fait de changer d'enregistrement actif provoque un appel implicite à Post.

Un appel à Post annule le mode Insert ou Append. Il faut donc repasser dans un de ces modes si on doit poursuivre des opérations de modifications.

Le fait de supprimer un enregistrement fait que le suivant prend sa place. Il ne faut donc pas, dans ce cas là "passer à l'enregistrement suivant" car un enregistrement serait sauté :

```
TMarques.Open ;
While not TMarques.EOF do
begin
      if TMarquesMarque.Value = TEdit.Text then
            TMarques.Delete
      else
            TMarques.Next ;
end ;
```

{ Chaque fois que le champ Marque a la valeur de la zone d'édition, l'enregistrement correspondant est effacé. L'enregistrement actif devient son successeur .. . il ne faut donc pas réaliser TMarques.Next dans ce cas }

Si le programmeur gère lui même une clé et que la table est affichée selon un index secondaire (par exemple par ordre alphabétique sur le champ 'Nom '), le programmeur ne peut atteindre la fin "réelle" de la table car l'instruction TTable.Last déplacera le curseur interne sur le dernier enregistrement de la "vue" constituée avec l'index secondaire.

Exemple :

On souhaite insérer une nouvelle marque à la table TMarques. Cette table est actuellement gérée par un index secondaire l'affichant par ordre alphabétique et utilise un identifiant créé par programmation (utilisé pour pouvoir accéder aux différentes tables secondaires de la base de données). Pour insérer un nouvel enregistrement il va falloir réaliser les opérations suivantes :

```
var
    Index : Integer ;

begin
    TMarques.Open ;
    TMarques.IndexName := '' ; { suppression de l'index secondaire }
    TMarques.Last ;    { positionnement en fin de table }
    Index := TMarquesIdMarque.Value ;
    TMarques.Append ; { ajout en queue d'un nouvel enregistrement }
    TMarquesMarque.Value := 'NOUVELLE MARQUE' ;
    TMarquesIdMarque.Value := Index + 1 ;
    TMarques.Post ;
    Tmarques.IndexName := 'Marque' ; { réactivation de l'index secondaire}
end ;
```

12.1.4 Modification d'enregistrements complets

Il est possible d'initialiser ou de modifier champ par champ un enregistrement (en appelant chacun d'entre eux avec la propriété Value). Cette méthode devient lourde dès lors qu'il y a beaucoup de champs à modifier.

On peut alors utiliser trois méthodes permettant de réaliser globalement les modifications souhaitées :

AppendRecord ([Tableau de valeurs])

Ajoute un enregistrement en fin de table et effectue un appel à Post.

InsertRecord ([Tableau de valeurs])

Idem mais l'insère à l'emplacement du curseur interne.

SetFields ([Tableau de valeurs])

Modifie les valeurs des champs de l'enregistrement actif. Ne fait pas appel à Post.

Exemples :

```
TMarques.AppendRecord ([ 9999, 'Daewood" ] );
TIdentite.SetFields([ nil , nil, 30 ] );
```

Pour pouvoir utiliser ces méthodes il faut connaître parfaitement la structure de la table (ordre et nature des différents champs).

Dans le deuxième exemple, le fait d'initialiser certains paramètres à 'nil ' fait que les valeurs courantes correspondantes ne sont pas modifiées .

Il est aussi possible d'initialiser un champ à la valeur Null.

Pour utiliser la méthode SetFields, il faut au préalable mettre la table dans le mode Edit puis, à l'issue réaliser un Post.

12.1.5 Evénements associés à la manipulation d'un ensemble de données

Chaque ensemble de données peut réagir à plusieurs événements. la création de gestionnaires d'événements associés permet d'affiner le comportement de l'application :

Evénement	**Action**
BeforeOpen, **AfterOpen**	Appelés avant ou après l'ouverture d'un ensemble de données
BeforeClose , **AfterClose**	Appelés avant ou après la fermeture d'un ensemble de données
BeforeInsert, **AfterInsert**	Appelés avant ou après le passage en mode Insert
BeforeEdit, **AfterEdit**	Appelés avant ou après le passage en mode Edit
BeforeCancel, **AfterCancel**	Appelés avant ou après l'annulationdu mode précédent
BeforeDelete, **AfterDelete**	Appelés avant ou après la suppression d'un enregistrement
OnNewRecord	Appelé lors de la création d'un nouvel enregistrement . Utile pour donner des valeurs par défaut.

Exemple :

On peut définir un gestionnaire d'événement associé à l'événement BeforeDelete :

```
procedure TForm1.TMarqueBeforeDelete(Dataset :TDataset );
begin
      if MessageDlg (' Confirmez-vous la suppression de l''enregistrement '),mtWarning ,
[mbYesNoCancel], 0)then
           Abort ;
end ;
     { Abort est une procédure qui permet de sortir de l'exécution prévue sans générer une exception }
```

12.2 Accès aux données par programmation

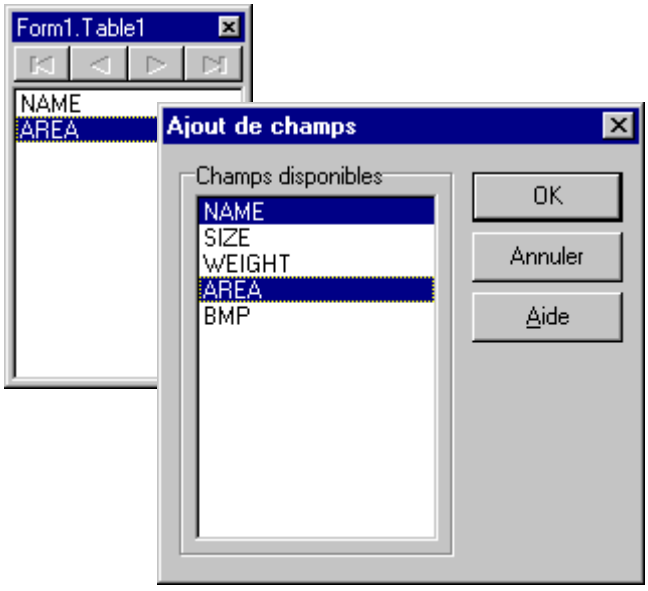
On s'est contenté, jusqu'à présent d'utiliser les propriétés de base proposées par Delphi pour permettre l'accès aux données. Si celles-ci sont intéressantes, et permettent de réaliser certaines interfaces sans avoir pratiquement à coder quoi que ce soit, on se rend compte que tout cela reste limité tant que l'on ne peut accéder directement aux différents champs de l'enregistrement courant.

Pour que cela soit possible, Delphi offre la possibilité de créer , à la demande, des objets spécifiques - appelés objets TField - qui encapsulent chaque champ d'un enregistrement.
Une fois un objet TField créé, il peut être configuré finement, dès la phase de conception, à l'aide de l'inspecteur d'objet puis manipulé par programmation.

12.2.1 Les objets TField

Le fait de "double-cliquer" sur l'icône représentant un composant TTable sur la fiche de conception ouvre une boîte de dialogue (appelée "éditeur de champ") qui permet de créer puis d'accéder à chaque champ de la table pris individuellement. Pour cela chaque champ est encapsulé dans un objet spécifique, de type TField , que l'on va pouvoir :

- Initialiser lors de la phase conception
- Modifier par programmation pendant l'exécution



▲ **L'éditeur de champ :**
Au départ la fenêtre de l'éditeur de champ est vide. L'activation du bouton 'Ajouter' visualise le nom de tous les champs contenus dans la table. Il faut alors sélectionner à la souris ceux que l'on souhaite utiliser, en tant que champ individualisé, dans l'application.

Il est possible de "supprimer" des objets champs voire de tous les effacer (cette suppression n'a, bien entendu, aucun effet sur la table).

Le fait de ne pas sélectionner un champ (en créant un objet TField) fait qu'il ne pourra pas être accessible ultérieurement par programmation.

Par contre on peut créer un objet TField, associé à un champ donnée, puis -on le verra plus loin - le rendre 'invisible'. Dans ce cas :

- Il n'est pas affiché dans le composant TGrid associé à la table ;
- On peut néanmoins y accéder par programmation. Cette possibilité est surtout intéressante lorsque l'on doit manipuler des clés numériques dont l'utilisateur n'a pas à avoir connaissance.

Le fait de pouvoir définir des objets TField fait que l'on peut accéder aux différents champs d'une table sans utiliser de composants constituant une interface utilisateur.

Il ne faut créer les objets TField que si l'on en a réellement besoin, car ils gonflent l'exécutable résultant. Si on ne souhaite que réaliser de l'affichage simple, il ne sont pas nécessaires.

Le bouton 'Définir' de l'éditeur de champs permet de créer des champs calculés (voir plus loin).

Le nom de chaque objet champ est généré automatiquement : il est constitué par le nom logique de la table auquel Delphi a rajouté, sans espace, le nom du champ dans la table.

C'est ce nom qui apparaît dans l'inspecteur d'objet lorsque le champ est sélectionné dans l'éditeur de champs.

Exemple :

Le champ Nom dans la table TMembres sera encapsulé dans un objet TField nommé TMembresNom.

Il est bien entendu possible de modifier - à l'aide de l'inspecteur d'objet - ce nom mais ce n'est en général pas utile et il est de plus facilement mémorisable.

▲ **Différents types de composants TField :**
En fait, il existe plusieurs types de composants TField. Le type correspondant à un champ donné est détecté automatiquement par Delphi.

Les principaux types d'objets TField sont :

TStringField	Textes de taille fixe jusqu'à 255 caractères
TIntegerField	Nombres entiers compris entre -2 147 483 648 et 2147 483 647
TSmallintField	Nombres entiers compris entre -32768 et 32767
TWordField	Nombres entiers compris entre 0 et 65535
TFloatField	Nombres réels
TCurrencyField	Valeurs monétaires.
TBooleanField	Valeur True ou False
TDateTimeField	Valeur date et heure
TDateField	Valeur date
TTimeField	Valeur heure
TBlobField	Champ arbitraire de données sans limitation de taille
TBytesField	Champ arbitraire de données sans limitation de taille
TMemoField	Texte de taille arbitraire
TGraphicField	Graphique de taille arbitraire tel qu'un bitmap

▲ **Principales propriétés :**
Une fois les objets champs ajoutés, il suffit de les sélectionner un par un pour pouvoir accéder, via l'inspecteur d'objet, à leurs propriétés propres :

Alignment	Permet de réaliser des alignements à l'affichage.
As....	Propriétés permettant de réaliser les conversions de types (voir plus loin)
Calculated	Indique si le champ est un champ calculé ou non
DisplayLabel	Titre du champ. Dans un composant TGrid, le titre des colonnes peut ainsi être plus explicite - et contenir des accents et des espaces - que celui créé avec DBD.
DisplayText	Valeur du champ tel qu'il est réellement affiché (par exemple pour un champ de type entier, il s'agit de la chaîne alphanumérique correspondant au nombre)
DisplayWidth	Permet de donner des largeurs différentes aux colonnes d'un composant TGrid
EditMask	Permet de réaliser des masques de saisie. La syntaxe utilisée est la même que celle du composant TEditMask.
IsNull	Renvoie vrai si la valeur du champ de l'enregistrement courant est nulle.
ReadOnly	Rend le champ uniquement accessible en lecture ou non.
Value	Valeur réelle du champ
Visible	Rend la colonne correspondant au champ visible ou non dans un composant TGrid.

La propriété Value (accessible seulement à l'exécution) est la propriété principale d'un objet TField car elle permet d'accéder en programmation à la valeur du champ de l'enregistrement courant.

On pourra ainsi avoir des instructions de type :

```
TMarques Marque.Value := 'AUDI' ;
```

ou :

```
EMarques.Text := TMarquesMarque.Value ;
```

Toute la puissance ultérieure développée par Delphi dans la gestion des données vient de sa capacité à gérer, par programmation les champs individuellement.

12.2.2 Conversion d'une valeur affectée dans un champ

Par défaut, Delphi reconnaît automatiquement le type de valeur que l'on souhaite affecter à un champ et effectue les conversions possibles. S'il y a incompatibilité, une erreur est générée.

Il est aussi possible de forcer une conversion de manière à ce que Delphi convertisse la valeur entrée dans le type souhaitée.

Par exemple, pour un champ de type "chaîne", Delphi générera une erreur si l'on entre des valeurs numériques. Si l'on réalise la conversion de la valeur entrée en chaîne de caractères, il n'y aura pas d'erreur.

On convertit la valeur d'un composant TField en utilisant une de ses propriétés As ..... . Les possibilités de conversions sont résumées dans le tableau ci-dessous :

Type de conversion

Type de TField	AsString	AsInteger	AsFloat	AsDateTime
TStringField	Type chaîne par définition	Convertir en entier si possible	Convertir en flottant si possible	Convertir en date si possible
TIntegerField **TSmallintField** **TWordField**	Convertir en chaîne	Type entier par définition	Convertir en flottant	Interdit
TFloatField **TCurrencyField** **TBCDField**	Convertir en chaîne	Arrondir à la valeur entière la plus proche	Type flottant par définition	Interdit
TDateTimeField **TDateField** **TTimeField**	Convertir en chaîne. Le contenu dépend du fomat d'affichage du champ	Interdit	Convertir date en nombre de jours depuis 01/01/0001. Convertir l'heure en fractions de 24 heures	Type date ou heure par définition. Aucune date ou heure en l'absence de spécification
TBooleanField	Convertir en chaîne "True" ou "False"	Interdit	Interdit	Interdit
TBytesField	Convertir en chaîne	Interdit	Interdit	Interdit

TVarBytesField **TBlobField** **TMemoField** **TGraphicField**	(n'a généralement lieu d'être que pour TMemoField)			

Exemples :

L'instruction suivante convertit la valeur du composant TField appelé Table1MonChampNum en une chaîne et l'affecte au texte du contrôle Edit1:

```
Edit1.Text := Table1MonChampNum.AsString ;
```

Inversement, l'instruction suivante affecte le texte du contrôle Edit1 au champ Table1MonChampChaine en tant que chaîne :

```
Table1MonChampChaine.AsString := Edit1.Text ;
```

Une exception se produit si une conversion non supportée est effectuée en mode exécution.

Si le champ est du type de la valeur correspondant au champ, il n'est pas nécessaire d'utiliser les fonctions de conversion. Cependant l'utilisation de telles possibilités sécurise les entrées et évite d'avoir à utiliser la syntaxe :

```
Table1MonChampChaine.Value := Edit1.Text ;
```

12.2.3 Différentes méthodes d'accès aux champs

Il est possible d'accéder aux différents champs selon diverses méthodes plus ou moins simples d'emploi :

▲ **Accès par l'objet TField :**
Cette méthode est la plus simple et c'est celle que l'on a étudié ci-dessus.

Une fois les différents objets champ créés, on peut accéder aux différentes valeurs des champs par la propriété Value.

Accès par la propriété Fields [i] :
Il est possible d'accéder aux différents champs d'une table en utilisant la propriété Fields [i] (i étant le rang du champ dans la table, 0 correspondant au premier champ).

Cette propriété n'est accessible qu'à l'exécution et en mode lecture (sauf en mode SetKey). Elle nécessite la connaissance complète de la structure de la table.

La propriété Fieldname de Fields permet de connaître le nom d'un champ.

```
titre := Table1.Fields [ 1 ].FieldName ;
```

{ Pour connaître les différents champs constituant une table, on peut réaliser une boucle comme celle-ci :}
```
for i := 0 to Table1.FieldCount do
        ListBox1.Items.Add (Table1.Fields[i].Fieldname ;
```

Accès par la méthode FieldByName (const FieldName : string) :

Cette méthode permet d'accéder à un champ en fournissant son nom.

Exemple :

```
with Table1 do
    begin
        FieldByName ('CustNo' ).AsString := '1234';
    end ;
```

{ Cette méthode est plus sûre que l'utilisation d'un indice de champ mais nécessite néanmoins de connaître les différents noms des champs.
On aurait en effet pu avoir :

```
    Fields[0].AsString := '1234';
```

}

Il y a équivalence stricte entre toutes les méthodes d'accès à un champ :

Si on a une table Identite comportant le champ Nom en deuxième position on peut accéder à ce champ par les instructions suivantes :

```
    var
        chaine [ 30 ] ;
    begin
        chaine := TIdentiteNom.Value ;
        chaine := TIdentite.Fieldbyname ('Nom' ).AsString ;
        chaine := TIdentite.Fields [ 1 ].AsString ;
    end ;
```

{ Notez que les deux dernières méthodes ne nécessitent pas la création d'un objet TField et que, en toute rigueur, la dernière - accès par un indice - est la plus rapide. }

12.3 Recherches dans une table

On a vu qu'il était possible de rechercher de manière itérative une information dans une table pour réaliser un traitement quelconque. Bien entendu, si la table est importante la recherche peut être longue.

Le composant TTable dispose de plusieurs méthodes permettant de rechercher rapidement des valeurs.

12.3.1 Utilisation des méthodes Goto

Les méthodes GotoKey et GoToNearest permettent de réaliser une recherche dans une table utilisant une clé.

Sous Paradox les champs clés doivent être placés en tête de la table.

Pour pouvoir effectuer une recherche avec ces méthodes il faut, au préalable, placer la table en 'mode de recherche' par la méthode SetKey (ce qui a pour effet d'empêcher les modifications sur les champs).

On donne la valeur à rechercher dans le champ clé et on réalise la recherche en appelant la méthode GoToKey (recherche stricte) ou GoToNearest (recherche les correspondances proches).

On a alors la routine suivante :

```
TTable.Close ;
TTable.Open ;              { vidage des buffers }
TTable.SetKey ;
TTable.Fields [ 0 ] . AsString := Edit1.Text ;
      { ou TTable.FieldByName ('Nom' ).AsString := Edit1.Text }
if not TTable.GotoKey then
      ShowMessage (' Enregistrement non trouvé ');
...
```

La recherche débute au début de la table et le curseur se positionne sur l'enregistrement correspondant à la valeur passée en paramètre à la 2° ligne.

GotoKey est une fonction booléenne qui renvoie True quand la recherche s'est bien passée (d'où la possibilité de tester cette valeur de retour).

Si la clé est composée de plusieurs champs et que l'on ne désire réaliser la recherche que sur un des champs, il faut initialiser la propriété KeyFieldCount avec l'indice du champ souhaité (1 = 1° champ).

L'instruction ' TTable.Fields [0] . AsString := Edit1.Text ; ' ne modifie pas la valeur du champ dans l'enregistrement courant. Elle sert juste à initialiser la recherche.

12.3.2 Utilisation des fonctions Find

Les fonctions FindKey() et FindNearest() combinent en un seul appel les actions réalisées par les séquences d'instructions de la famille précédente.

Exemple :

```
TTable.Findkey ([ Edit1.Text ]);
```

a le même effet que :

```
TTable.Setkey ;
TTable.Fields [ 0 ].AsString := Edit1.Text ;
TTable.GoToKey ;
```

12.3.3 Recherche selon un index secondaire

Pour effectuer une recherche sur un champ qui n'est pas la clé primaire on doit spécifier le nom du champ à utiliser par la propriété IndexName.

```
TMarques.Close ;
TMarques.IndexName := 'indsecond' ;
TMarques.Open ;
TMarques.SetKey ;
TMarques.FielByName ('Marque ').AsString := Edit.Text ;
TMarques.GoToNearest ;
```

Il faut que l'index secondaire ait été créé au préalable dans DBD (son nom apparaît dans l'inspecteur d'objet dans le champ IndexName).

Si entre deux recherches on change d'index il faut fermer la table avant de réaliser le changement d'index (propriété IndexName de la table).

13 Requêtes SQL

Les applications Delphi peuvent utiliser le langage de requêtes SQL pour accéder aux données suivantes :

- Les tables aux formats Paradox ou dBase.
- Les bases de données gérées par Interbase.
- Les serveurs SQL accessibles à distance via SQL Link.

13.1 Généralités sur SQL

13.1.1 Qu'est-ce que SQL ?

SQL, ou Structured Query Language, est un langage de requête qui permet d'accéder aux données contenues dans une base de données de type relationnel.

SQL est en fait un des piliers sur lequel les gestionnaires de base de données relationnels (SGBDR) ont été conçus. D'ailleurs on appelle ces produits des "Serveurs SQL".

C'est un langage abstrait qui permet de se détacher de l'aspect physique des données (emplacement du fichier les contenant, taille de l'enregistrement, méthodes de gestion,...). En effet, SQL est un langage définit selon une norme (norme ANSI) indépendamment de tout logiciel.

- Aujourd'hui tout système de gestion de base de données propose le langage SQL pour accéder aux données. Il devrait normalement tenir compte de la normalisation ANSI. Cependant chaque éditeur rajoute des "améliorations" personnelles au langage SQL de base (syntaxe, fonction de concaténation de chaîne,...) ce qui fait qu'il y a autant de SQL que d'éditeur.

Qui plus est, il y a plusieurs versions du standard SQL. La dernière s'appelle SQL2 mais SQL3 est en cours de définition.

- Delphi, en tous cas dans sa version Desktop, propose un sous-ensemble du SQL standard. Ce qui veut dire que les fonctionnalités qu'il propose sont reconnues par tous les gestionnaires de base de données… ce qui veut dire aussi que toutes les fonctionnalités SQL ne sont pas disponibles.

SQL permet la gestion complète d'une base de données relationnelle (création, suppression ou modification des tables). Mais on utilise surtout SQL pour réaliser des requêtes afin d'obtenir des données correspondants à certains critères.

13.1.2 Qu'est-ce qu'une requête ?

Une requête est une phrase qui adopte une syntaxe bien particulière fixée par le langage SQL, et qui permet de trier, selon certains critères, les enregistrements appartenant à une ou plusieurs tables.

La réponse à cette question est stockée dans une nouvelle table qui comporte tout ou partie des enregistrements initiaux. Les enregistrements retenus sont ceux correspondant aux critères définis dans la requête. Ils sont affichés à l'écran sous une forme déterminée volatile ("vue").

13.2 Syntaxe SQL

La syntaxe générale de SQL est relativement fruste. Elle est réduite à l'utilisation de quelques mots clés et quelques opérateurs. Néanmoins elle permet de réaliser des extractions très complexes sur plusieurs tables en même temps.

On distingue deux catégories d'instruction SQL :

- Celles qui permettent la définition des données (création de table, modification de leurs structure, etc ...).
- Celles qui permettent de manipuler les données (sélection, insertion, suppression).

Quelques règles générales sont appliquées dans tous les cas :

- Chaque table doit être référencée par un nom unique.
- Chaque champ d'une table est référencé par un nom unique. Lorsque l'on veut spécifier un champ dans une table on utilise la syntaxe : < table > . < champ> .
- Le caractère spécial '*' signifie 'tous les champs'.

- Le SQL de Delphi accepte les extensions pour permettre de définir une table par son nom de fichier.

- Si le nom d'un champ correspond à un mot réservé de SQL il doit être mis entre guillemet ou entre apostrophes.

13.2.1 Définition des données

Des ordres SQL permettent de créer, modifier et supprimer des tables. On peut aussi créer et supprimer des index. Les ordres supportés par Delphi sont :

▲ **CREATE TABLE :**

Cet ordre permet de créer une table en définissant, le cas échéant, un index. Il permet de donner un nom à la table et définit le nom et le type des différents champs.

Exemple :

```
CREATE TABLE "Employes.db"
(
NOM Char (20),
PRENOM Char (20),
```

```
AGE SmallInt ,
PRIMARY KEY (NOM, PRENOM )
)
```

▲ **ALTER TABLE :**

Cet ordre permet de modifier la structure d'une table existante :

Exemple :

```
ALTER TABLE "Employes.db" ADD TPH Char (15)
```

Ajoute un champ à la table précédente.

▲ **DROP TABLE :**

Supprime une table.

Exemple :

```
DROP TABLE "Employes.db"
```

▲ **CREATE INDEX et DROP INDEX :**

Ces ordres permettent de créer ou d'effacer des index.

Exemple :

```
CREATE INDEX IndNom ON " Employes.db " (NOM )

DROP INDEX "Employes.db". PRIMARY
    { Supprime l'index primaire }

DROP INDEX "Employes.db".IndNom
    { Supprime l'index secondaire }
```

13.2.2 Manipulation de données

Ce sont les ordres de manipulation de données que l'on utilise le plus car ce sont eux qui permettent de générer des requêtes.

▲ **SELECT :**

L'instruction SELECT est utilisée pour afficher un ou plusieurs champs, situés dans une ou plusieurs tables, à partir de critères spécifiés.

Lorsque plusieurs champs doivent être sélectionnés, il doivent être séparés par une ' , ' .

Dans le cas où SELECT œuvre sur plusieurs tables, on parle de jointure.

Exemple :

```
SELECT Nom, Prénom      sélectionne les champs "nom" et "prénom"
SELECT *                sélectionne tous les champs
```

▲ **FROM :**

La clause FROM spécifie la ou les tables dans lesquelles les données doivent être recherchées.

S'il y a plusieurs tables, elle doivent être séparées par une ', ' .

Exemple :
FROM Liste_Personnel
FROM Liste_Personnel, Liste_Véhicule

WHERE :

La clause, facultative, WHERE spécifie les critères de sélection.

Une condition s'exprime sur un ou plusieurs champs. On peut utiliser le mot réservé IN pour indiquer une liste.

Exemple :
WHERE Nom = 'Toto'

ORDER BY :

La clause, facultative, ORDER BY spécifie le champ dans l'ordre duquel les lignes qui seront affichées.

Exemple :
ORDER BY Nom

On peut préciser ASC pour un ordre ascendant (ou alphabétique)
DESC pour un ordre descendant (ou inverse).

13.2.3 Format des requêtes

Le format général d'une requête est le suivant :

SELECT < liste des champs à sélectionner > **FROM** < liste des tables sources >
WHERE <condition(s)> **ORDER BY** <ordre>

Requêtes SQL	Actions
SELECT Nom, Prénom **FROM** Employes **WHERE** Nom = 'Toto' **ORDER BY** Nom	Affiche les champs 'Nom ' et 'Prenom' de la table 'Employes' des enregistrements où Nom=Toto. Trier par Nom.
SELECT * FROM Liste_Personnel ;	On obtient une copie de la table Liste_Personnel.
SELECT * FROM Liste_Personnel **ORDER BY** Prénom ;	Idem trié par prénom.
SELECT Reference,NomPiece,Prix **FROM** Fournisseurs **WHERE** Prix >30 **AND** Prix <60 ;	Affiche les champs 'Reference', 'NomPiece','Prix' de la table 'Fournisseurs' des enregistrements où le Prix est compris entre 30 et 60.

SELECT Customer.Company,Orders.OrderNo, Orders.SaleDate **FROM** Customer, Orders **WHERE** Customer.CustNo = Orders.CustNo ;	Jointure entre deux tables : Affichage des champs 'Customer' de la table 'Company' et 'OrderNo' et 'SaleDate' de la table 'Orders' pour lesquels le contenu du champ ' CustNo ' de la table 'Customer' et égal au contenu du champ 'CustNo' de la table 'Orders'.
SELECT Nom, Reference **FROM** Fournisseurs **WHERE** Ville = 'Lyon' **AND** (NomPiece='valve' **AND** Reference ='10') ;	Affiche les champs 'Nom' et 'Reference' de la table 'Fournisseurs' des enregistrements où le champ 'Ville' = Lyon et où on a aussi 'NomPiece' = valve et 'Reference' = 10.
SELECT Nom, Reference, Ville **FROM** Fournisseurs **WHERE** Reference = '30' **AND** Ville **NOT IN** ('Paris');	Affiche les champs 'Nom', 'Reference' et 'Ville' de la table 'Fournisseurs' des enregistrements où 'Reference' = 30 et où 'Ville' n'est pas Paris.

13.2.4 Les opérateurs SQL

Les opérateurs supportés par le SQL local de Delphi sont :

Type	Opérateur
Arithmétique	+, -, *, /
Comparaison	<, >, =, <>, IS NULL, LIKE
Logique	AND, OR, NOT

- L'opérateur LIKE permet de faire une recherche à l'aide d'un 'modèle' au lieu de réaliser une recherche stricte (comme avec l'opérateur ' = '). Lorsque l'on utilise LIKE dans une requête on peut réaliser des recherches par mots clés.

Exemples :

```
Select * from TMembres
where Nom = 'Dupond'
{ Affiche les membres s'appelant Dupond }

Select * from TMembres
where Nom LIKE '%Du%'
{ Affiche tous les membres contenant le modèle 'Du' (Dupond, Durand, mais aussi LeDu, Tordu, etc .....}
```

Pour employer l'opérateur LIKE il faut utiliser le caractère '%' signifiant ' n'importe quoi' ('%Du%' = n'importe quoi précédant 'Du' suivi de n'importe quoi).

LIKE ne distingue pas les majuscules des minuscules, à la différence de ' ='.

13.3 Utilisation du composant TQuery

TQuery est un composant, du même type que TTable, qui permet de transmettre des commandes SQL à BDE (ou à tout autre serveur de bases de données).

TQuery peut être utilisé pour réaliser :

- des requêtes multitables ;
- des requêtes complexes avec appel de sous requêtes ;

Comme TTable , TQuery se pose sur la fiche et est un composant non-visuel. Il faudra aussi lui connecter un composant TDataSource pour que les données, résultat d'une requête, puissent être affichées dans les différents composants prévus (TGrid en général).

13.3.1 Configuration du composant

Les principales propriétés du composant sont celles que l'on a étudié avec le composant TTable. Sa seule propriété supplémentaire est en fait sa propriété principale : la propriété SQL, de type TStrings (avec toutes les sous-propriétés et méthodes que cela implique), qui contient la requête SQL.

Il existe aussi dans ce composant une propriété DataSource mais elle sert à lier le composant à un autre ensemble de données (Cela sera vu ultérieurement). En général ce champ reste vide.

Delphi, contrairement à d'autres produits SQL , permet la modification des données affichées par une requête. Pour cela il faut que la propriété RequestLive du composant TQuery soit à True.

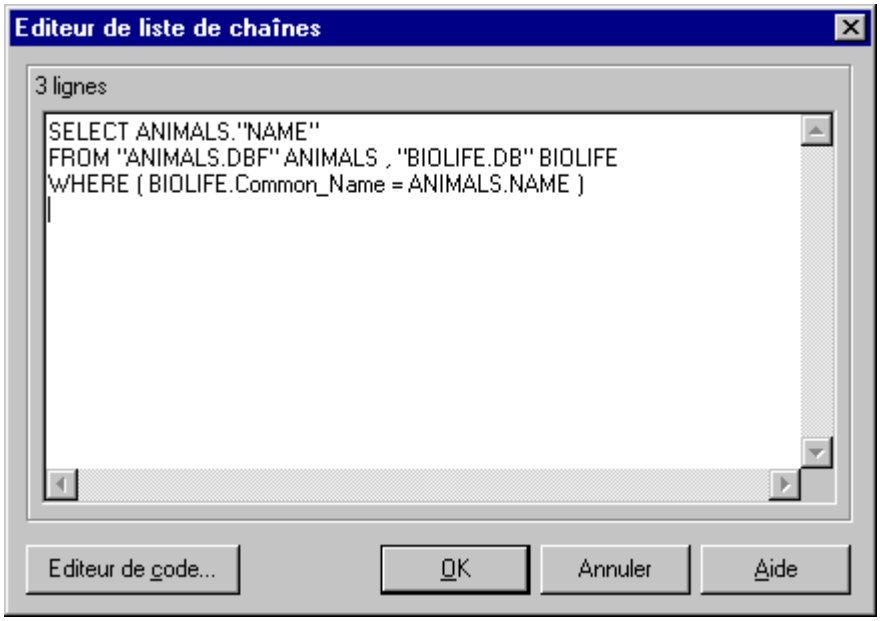
Si un composant TTable utilise les mêmes données, il faut penser à rafraîchir les données qu'il affiche (TTable1.Refresh est appelé par l'événement AfterPost de TQuery).

Une fois une requête rédigée, il est possible de créer des objets champs à partir d'elle. Pour cela il faut, comme pour le composant TTable, double-cliquer sur le composant afin d'afficher l'éditeur de champs.

13.3.2 Rédaction de requêtes SQL

Il existe (au moins) trois possibilités de rédiger, dès la phase de conception, les requêtes SQL nécessaires :

▲ **En utilisant directement la propriété SQL du composant TQuery :**
Le fait d'activer cette propriété ouvre une boite de dialogue dans lequel il est possible d'écrire la requête SQL souhaitée.



L'éditeur permettant de créer des requêtes SQL.

Cette méthode est surtout utilisée pour créer des requêtes SQL simples. Elle peut l'être aussi par les spécialistes du langage SQL.

Il est aussi possible d'importer un fichier d'extension '.txt' , contenant une requête SQL, via la méthode LoadFromFile :

```
TQuery1.SQL.LoadFromFile ("Raquete1.sql');
```

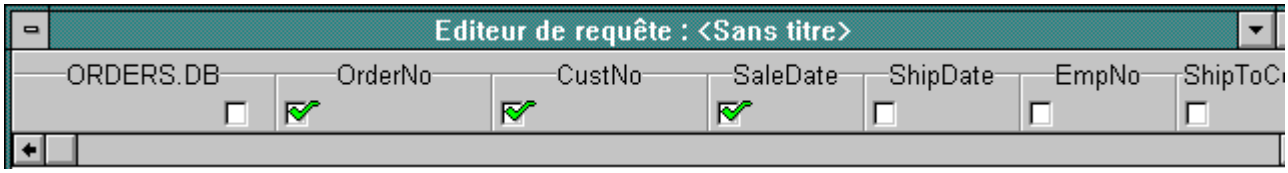
▲ **En utilisant l'utilitaire QBE inclus dans l'utilitaire DBD :**

QBE (Query By Exemple) est un langage permettant normalement de créer des requêtes de manières intuitives grâce à un utilitaire spécifique.

QBE est très facile d'emploi mais ne dépasse pas le cadre des produits de bas de gamme que l'on trouve dans le monde PC. Il n'est donc pas standardisé et il est donc préférable d'utiliser les requêtes SQL.

Pour plus de précision sur l'utilisation de QBE il est préférable de se référer à un manuel de formation à Paradox.

Lorsque l'on réalise une requête QBE il suffit de sélectionner la table concernée puis de "cocher" les champs que l'on veut visualiser. On peut éventuellement définir des critères au niveau de chaque champ sélectionné.

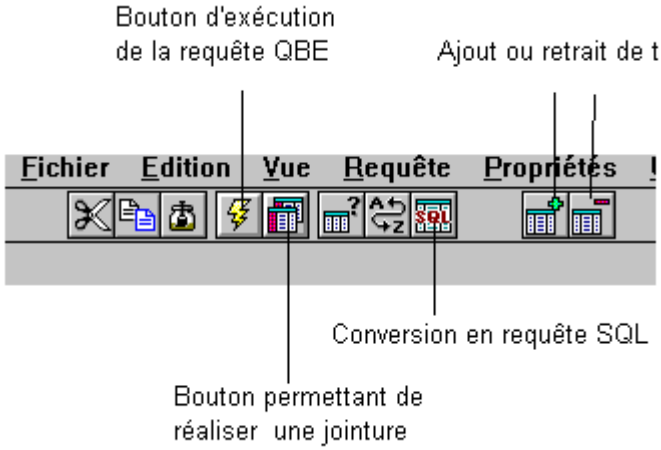


Une requête QBE sur une table

Il est possible de réaliser des jointures entre deux tables en les sélectionnant toutes les deux et en indiquant quel est le champ permettant de réaliser la jointure.

Deux tables liées par une jointure.

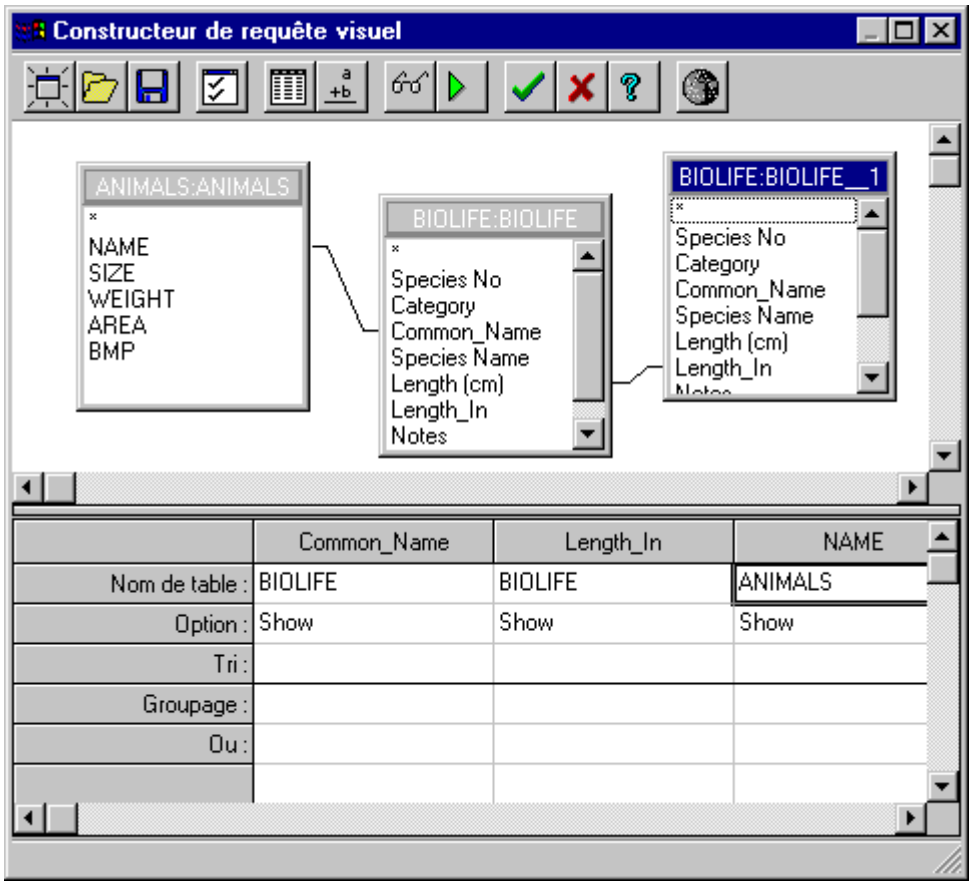
Une fois la requête réalisée il est possible de l'exécuter pour vérifier sa validité (bouton "éclair" de la barre d'icônes). On peut alors convertir directement cette requête QBE en requête SQL par activation du bouton concerné puis, par copier-coller en utilisant la combinaison de touches Maj + Ins), intégrer le texte SQL généré dans l'éditeur SQL de Delphi.



Cette méthode permet de concevoir des requêtes SQL complexes et fiables.

Cette méthode n'est utilisable que pour les tables de format Paradox ou dBase.

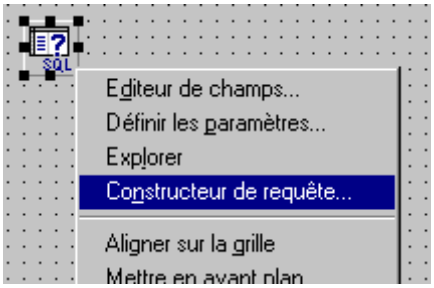
▲ **En utilisant le générateur visuel de requêtes :**
Delphi 3 propose un générateur de requêtes entièrement visuel très puissant (version C/S).



Ce générateur permet de visualiser les différentes tables et de définir les liens qui existent entre elles (un peu à la manière du produit équivalent proposé par Access).

A partir de là il est possible de générer des requêtes SQL complexes sans avoir à passer par DBD et QBE.

Pour faire lancer le générateur visuel de requêtes, choisissez l'entrée de menu "Constructeur de requête" du pop-up menu d'un TQuery.



13.4 Programmation SQL

13.4.1 Programmation d'ordre de description de données

On peut exécuter des ordres de définition de données en programmation. Pour cela il faut utiliser les propriétés et méthodes de la propriété SQL du composant TQuery, de typeTStrings.

1. Rendre inactif, le cas échéant le composant TQuery1
   TQuery1. Active = False ;
   ou mieux :
   TQuery1.Close ;

2. Nettoyer la propriété SQL d'éventuels textes antérieurs:
   TQuery1.SQL.Clear ;

3. Ajouter les différentes lignes composant la requête à l'aide de la méthode Add.
   TQuery1.SQL.Add (' CREATE TABLE "Employe");

4. Enfin on exécute la requête avec la méthode ExecSQL :
   TQuery1.ExecSQL ;

13.4.2 Programmation de requêtes SQL

Il est aussi possible de réaliser une requête SQL directement dans le code de l'application. Pour pouvoir programmer une telle requête il faut utiliser la propriété SQL du composant TQuery.

On agit tout d'abord comme précédemment (les 3 premières phases):

```
TQuery1. Close ;
TQuery1.SQL.Clear;
TQuery1.SQL.Add (' Select * from Liste_Personnel );
```

Dans ce cas il faut réaliser autant d'appels à la méthode Add () qu'il y a de lignes dans la requête à créer.

Il faut toujours penser à "nettoyer", grâce à la méthode Clear, les chaînes contenues dans la propriété SQL avant d'en rajouter de nouvelles.

Par contre on exécute la méthode Open pour lancer la requête:

```
TQuery1.Open ;
```

- ExecSQL ou Open ? : La méthode ExecSQL est à utiliser quand l'ordre émis ne renvoie pas de résultat. Dans le cas contraire (requête) il faut utiliser la méthode Open.

13.5 Requêtes SQL dynamiques (ou "paramétrées")

Il y a lieu de distinguer :

- Les requêtes SQL statiques dont le texte de la requête peut être défini entièrement (dès la phase de conception ou pendant l'exécution).
- Les requêtes SQL dynamiques dont certaines parties ne sont définies qu'à l'exécution (en général à l'issue d'un choix fait par l'utilisateur).

Jusqu'à présent nous n'avons utilisé que des requêtes statiques. Mais la puissance de SQL ne se révèle que dans la possibilité de réaliser des requêtes dynamiques, seules requêtes permettant à l'utilisateur de réaliser un choix.

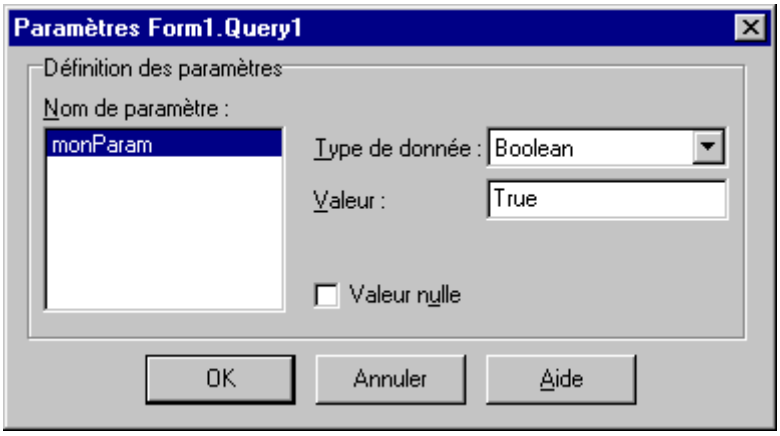
13.5.1 Réalisation d'une requête SQL dynamique

▲ **Paramétrage initial :**
Pour pouvoir programmer une requête dynamique, il faut dès la phase de conception, réaliser certains paramétrages sur le champ qui va être initialisé dynamiquement (il est possible que la requête dynamique soit réalisée sur plusieurs champs).

Pour cela il faut utiliser une boite de dialogue, appelée "éditeur de paramètres" qui apparaît lorsque l'on active la propriété Params du composant TQuery.

Cette boite permet de définir le nom et le type du, ou des, paramètre(s) qui sera utilisé dans la requête SQL dynamique. En général il s'agit de la zone de saisie d'un composant TEdit ou TDBEdit ou bien encore la sélection d'un composant TListBox, TDBListBox, TComboBox, TDBComboBox, etc....

On peut éventuellement donner une valeur initiale mais ce n'est pas une obligation.



Initialisation d'un paramètre : il s'agit ici de la zone Text d'un composant ComboBox.

▲ **Réalisation de la requête SQL :**

La requête SQL dynamique peut être générée soit via l'éditeur de requêtes soit par programmation.

Le paramètre dont la valeur est variable est signalisé par la syntaxe :

:<nom_paramètre>

Exemple :

Select Activite.Date, Activite.Marque, Activite.Modele, Activite.Piece, Activite.Nombre

From Activite

Where Activite.Entree = 'entrée' AND Activite.Jour = :CBDate.Text

{ La valeur prise en compte dans la clause Where sera celle de CBDDate.Text au moment de l'exécution de la requête}.

Delphi étudie la syntaxe de la requête SQL et propose d'emblée, dans la fenêtre 'Nom du paramètre', le ou les paramètres devant être paramétrés. Il propose aussi un type à cette variable (si le champ concerné est l'origine d'un champ objet de type TField).

S'il n'y a pas de requêtes SQL rédigée, Delphi n'est pas en mesure de faire de propositions.

▲ **Exécution de la requête :**

Une requête SQL dynamique ne peut être exécutée que par programmation. Il faut pour cela utiliser la propriété Params [i] ou la méthode ParamByName du composant TQuery.

Par exemple le gestionnaire d'événement suivant exécute la requête dynamique précédente :

```
procedure TFActivite.SBEntreeClick (Sender: TObject);
begin
    ...          { initialisations préalables }
    QEntree.Params [ 0 ].AsString := CBDate.Text ;
    QEntree.Active := TRUE ;          { Ou QEntree.Open }
    ...
end;
{ La valeur courante de CBADate.Text est celle qui est prise en compte dans la requête SQL à son exécution }
```

On aurait pu utiliser la ligne :

```
QEntree.ParamByName ('Date' ).AsString := CBDate.Text ;
```

▲ **Limitations SQL aux requêtes dynamiques :**

- Lorsque l'on réalise une requête dynamique, la syntaxe autorisée est réduite :
- On ne peut utiliser la clause ORDER BY.
- On ne peut utiliser de champ calculé dans la clause SELECT.
- La clause WHERE ne comporte que des comparaisons de noms de champs à des constantes scalaires.

13.5.2 Requêtes dynamiques liées à une table

Une autre possibilité de réaliser des requêtes dynamiques est d'utiliser la propriété DataSource du composant TQuery.

Cette propriété doit alors spécifier le composant DataSource lié à une table (par son composant TTable).

On peut alors utiliser, dans la clause WHERE, des paramètres non initialisés (selon la syntaxe : <paramètre >). Si le nom du paramètre ainsi spécifié correspond à un des champs de la table "maître" liée par DataSource, le paramètre prendra la valeur du champ correspondant de l'enregistrement actif au moment de l'exécution de la requête.

Exemple :

On crée un projet comportant :

- Une table "maître" utilisant les composants suivants :

Un composant TTable
(DataBaseName = DBDEMOS , TableName = CUSTOMERS , Name =Cust).
Un composant TDataSource (DataSet = Cust , Name = DataSource1).
Un composant TDBGrid (Name = CustGrid , DataSource = DataSource1).

- Une table utilisant les composants suivants :
  Un composant TQuery
  (DataBaseName = DBDEMOS , DataSource = DataSource1 , Name = Orders)
  Un composant TDataSource (DataSet = Orders , Name =Datasource2).
  Un composant TDBGrid (Name = OrdersGrid , DataSource =DataSource2).

- Les deux tables ont leur propriété Active = True.

- La requête SQL du composant TQuery est :
  Select Orders.CustNo, Orders.OrderNo, Orders.SaleDate
  From Orders
  Where Orders.CustNo = :CustNo

Comme les deux ensembles de données sont actifs, les requêtes sont exécutées dès le démarrage de l'application. Le paramètre :CustNo est initialisé avec la valeur du champ correspondant de l'enregistrement actif de la table Cust. A chaque fois que l'enregistrement actif est modifié, la requête est de nouveau exécutée pour mise à jour de la grille.

▲ **Les limitations SQL de Delphi :**

Dans sa version "locale" il ne fait aucun doute que les possibilités SQL de Delphi restent limitées. Cela correspond à la tendance générale qui veut que l'implémentation du langage SQL est souvent adaptée à l'environnement qui l'emploie. C'est pourquoi, il ne faut pas s'attendre à retrouver dans Delphi, un langage SQL aussi performant et aussi puissant que ceux implémentés dans les SGBDR (Serveurs SQL).

Cette limitation oblige parfois le développeur à refaire "à la main", par programmation, ses propres fonctions de tri. Et c'est là que les nombreuses possibilités offertes par Delphi dépassent celles que l'on peut trouver dans de nombreux L4G spécifiques. Comme de plus le code généré est compilé, on retrouve alors une rapidité d'exécution pouvant soutenir la comparaison avec des requêtes SQL plus puissantes mais restant interprétées.

14 Possibilités avancées

Delphi propose d'autres possibilités qui permettent de réaliser des applications orientées "données" rapides et fiables. Ces possibilités sont néanmoins - pour la plupart - réservées aux programmeurs avertis car elles nécessitent souvent de se passer des possibilités offertes par la simple manipulation des composants.

14.1 Création d'une base de données de manière dynamique

Il est possible de créer dynamiquement une base de données. Cette manière de procéder, se révèle très puissante mais nécessite la connaissance précise de la localisation de la base à construire et celle de sa structure interne.

14.1.1 Création dynamique de la base

La création dynamique d'une base de données passe par l'utilisation du composant TTable, dans la mesure où c'est lui qui permet de faire l'interface avec la base elle-même.

Il ne faut pas oublier que l'on peut aussi créer une table et, éventuellement, modifier ses caractéristiques grâce au langage SQL (en utilisant les ordres de description des données).

Certaines propriétés du composant doivent alors être renseignées, il s'agit de :

- **TableName** : Chemin complet indiquant le répertoire dans lequel les tables et autres objets nécessaires à la gestion de la base seront stockés.
- **TableType** : Type de base de données (Paradox, DBase, ASCII,etc...).

Si ce champ n'est pas renseigné, le format de la base sera le format Paradox.

▲ **Création des champs :**
Une fois indiqué où sera créée la base, il faut définir les champs qui vont composer la structure de la base. Cela est réalisé à l'aide de la propriété FieldDefs (qui est un objet à part entière), qui répertorie l'ensemble des informations relatives à chaque champ composant la base.

Il est possible de créer les champs un par un pour réaliser la base. On utilise pour cela la méthode Add (). L'appel à cette dernière se faisant de la manière suivante :

Add (NomChamp, TypeChamp, TailleChamp, ChampObligatoire) ;

Avec :

NomChamp	Nom du champ à créer
TypeChamp	Type du champ (chaîne, entier, logique, etc...)
TailleChamp	Définit la taille du champ (particulièrement utile si le champs est de type chaîne, par ex.). Ce champ vaut 0 (zéro) si le champ créé est de type entier, mémo, logique, etc...ChampObligatoire De type booléen (True : le champ est obligatoire)

Il est conseillé au préalable de s'assurer que FieldDefs est vide avant de l'initialiser. Cela peut être réalisé par la méthode Clear.

▲ **Création de l'index primaire :**

On peut ensuite définir les indexs. En réalité, seul les index primaires peuvent être créés à ce niveau. Il faut utiliser la propriété IndexDefs du composant TTable. Pour cela on utilise une nouvelle fois la méthode Add () selon la syntaxe :

Add (NomIndex, ChampsConcernés, TypeIndex);

Avec :

NomIndex	Nom de l'index
ChampsConcernés	Chaîne répertoriant l'ensemble des champs servant pour l'indexage. Si il y a plus d'un champ, il faut alors les séparer par un ' ; '

ex. : Champs1 ; Champs2 ; Champs3

TypeIndex : combinaison d'une ou plusieurs caractéristiques permettant de typer l'index. Les possibilités sont les suivantes : ixPrimary, ixUnique, ixDescending et ixNonMaintained. Pour un index primaire à clé unique, nous aurons : [ixPrimary, ixUnique].

Pour la même raison que précédemment il faut nettoyer la liste des indexes à l'aide de la méthode Clear.

▲ **Création de la table :**

Les champs et l'index primaire étant définis, il reste à créer la base proprement dit. Pour pouvoir faire cela, il faut utiliser la méthode CreateTable du composant TTable.

Exemple :

Supposons que nous voulions créer une base composée des champs suivants :

Nom	String [20]	Champ obligatoire
Prénom	String [20]	Champ obligatoire
Age	Entier	

L'index primaire est basé sur le couple (Nom, Prénom). Ce dernier devant être une clé unique.

Le code Delphi correspondant à la création d'une telle base se fera de la manière suivante :

```
Procedure TMainForm . CreerTable (NomTable : String) ;
Var
    Table : TTable ;            { Table dynamique }
Begin
    Table := TTable . Create (Self);
    { Création d'une instance de la table TTable }
    With Table Do
    Begin
        TableName := 'C:\MaBase' ;
        { Chemin d'accès à la future base }
        TableType := ttParadox ; {Structure de type PARADOX}
        With FieldDefs Do
        Begin
            Clear ; { Nettoyage de FieldDefs }
            Add ('Nom', ftString, 20, True) ;
            { Création du champ 'Nom' }
            Add ('Prénom', ftString, 15, True);
            { Création du champ 'Prénom'}
            Add ('Age', ftDate, 0, False) ;
            { Création du champ 'Age' }
        End ;
        With IndexDefs Do { Création de l'index }
        Begin
            Clear ;
            Add ('Index', 'Nom ; Prénom', [ ixPrimary, ixUnique]);
            { Création de l'index principal }
        End ;
        CreateTable ;
    End ;     { Création physique de la base }
End;
```

14.1.2 Création d'index secondaires

La création d'un index secondaire n'est pas ce qu'il y a de plus facile. Il n'est en effet plus possible d'utiliser la méthode Add () de IndexRefs.

Il faut faire appel à une commande SQL appelée CreateIndex, et pour cela utiliser un composant TQuery spécialement pour cela.

On ne peut définir un index secondaire que si un index primaire a été créé au préalable.

Exemple :

Supposons que nous voulions créer un index secondaire sur l'âge de la personne, de manière à pouvoir trier les personnes par leur âge (pourquoi pas, après tout...). Quoiqu'il en soit, cela se fera ainsi :

```
Procedure TMainForm . CreerIndexSecondaire (NomTable :String) ;
Var
    Query : TQuery ;
    { Création d'un composant Requête dynamiquement }
Begin
    Query := TQuery . Create (Application);
    With Query Do
        Begin
            SQL.Clear ;          { Nettoyage initial }
            SQL.Add (Format ('CREATE INDEX AutreIndex ON "%S"
            ("%S"."Date de Naissance")', [NomTable, NomTable ] )) ;
            ExecSQL ;
        End ;
    Query.Free ;
End ;
```

▲ **Libération de la base :**
Il est possible de supprimer une base si celle-ci était une base temporaire.

Pour cela il suffit d'appeler la méthode Free :

```
Table.Free ;
```

14.2 Possibilités avancées du composant TTable

14.2.1 Champs calculés

Il est possible de définir des "champs calculés". Il s'agit de champs spéciaux qui n'ont pas d'existence réelles (ils ne correspondant pas à des données stockées sur le disque) mais qui correspondant à des données calculées en mémoire à partir de données présentes dans d'autres champs.

Ces champs apparaissent néanmoins dans l'interface utilisateur (soit en tant que zone d'édition soit comme colonne au sein d'une grille).

▲ **Définition d'un champ calculé :**
Pour utiliser un champ calculé, il faut d'abord le définir. Pour cela il faut utiliser l'éditeur de champs attaché à l'ensemble de données utilisé (un composant TTable ou TQuery).

En double-cliquant sur le composant, l'éditeur de champs apparaît. Il faut alors utiliser le bouton 'Définir'. Une nouvelle boite de dialogue apparaît alors. Elle permet de définir le champ calculé souhaité en lui donnant un nom (ce qui implique le nom de l'objet TField associé) et le type du champ.

- Bien prendre en compte le fait que le champ calculé est alors manipulable comme tous les objets TField attachés au composant. On peut donc configurer les conditions d'affichage du champ.

▲ **Calcul du champ :**
Il faut ensuite indiquer la formule (instruction ou ensemble d'instructions) permettant le calcul du champ ainsi créé.

Il faut pour cela créer un gestionnaire d'événement attaché à l'événement OnCalcFields du composant.

Ce gestionnaire est appelé à chaque fois qu'un rafraîchissement se produit dans l'ensemble de données.

Exemple d'utilisation :

Soit une table PRODUITS comprenant les champs suivants :
Produit (String), PrixHT(Currency), TVA (Numeric), Remise (Numeric).
(Soit des enregistrements comprenant 4 champs de données présentes physiquement sur le disque).

On peut créer deux champs calculés à partir de cela :
Le champ PrixTTC (Currency) et le champ PrixVente (Currency).
Ces deux champs sont définis à l'aide de l'éditeur de champs.

On crée une interface utilisateur avec un composant TTableconnecté à la table PRODUITS, un TDataSource et une TGrid.

On crée ensuite le gestionnaire d'événement suivant :

```
procedure TForm1.TProduitsCalcFields (DataSet: TDataset );
```

```
begin
      TProduitsPrixTTC.Value := TProduitsPrixHT.Value * (1 + (TProduitsTVA.Value / 100 )) ;
      TProduitsPrixVente.Value := TProduitsPrixTTC.Value *(1 - (TProduitsRemise.Value / 100 )) ;
end;

{ A l'exécution 6 champs apparaissent dans la grille : les 4 champs normaux suivis par les deux
champs calculés }

{ On peut constater sur l'exemple qu'un champ calculé peut servir de base de calcul pour un autre
champ calculé }
```

14.2.2 Pose de marques

Il est parfois nécessaire de pouvoir marquer un enregistrement courant dans une table, puis de se déplacer dans celle-ci tout en conservant la possibilité de revenir à l'enregistrement de départ rapidement.

Pour ce faire on utilise les possibilités de marquages offertes par Delphi en appelant les méthodes GetBookmark, GoToBookmark () et FreeBookMark () du composant TTable.

- GetBookMark () est utilisé pour poser une marque sur un enregistrement particulier. Elle renvoie une donnée (de type TBookMark) qui est un pointeur sur un enregistrement. La fonction renvoie nil si l'appel échoue.
- GoToBookMark () permet de se déplacer vers une marque préalablement posée.
- FreeBookMark () libère une marque posée.

- Il est possible de poser plusieurs marques dans une même table. Mais il faudra penser ultérieurement à les libérer une à une.

Exemple :

```
Bookmark : TBookMark ;
      { Déclaration d'une marque (variable globale)}
if BookMark = nil then
      { vérification que la marque n'est pas utilisée }
      Bookmark := Table1.GetbookMark ;        { Pose d'une marque}

< suite du programme >

if BookMark <> nil then { Va à la marque si elle existe }
begin
      Table1.GoToBookMark (BookMark);
      { Déplacement vers la marque }
      < Suite du programme >

      Table1.FreeBookMark (BookMark);
      BookMark := Nil ;
end ;
```

14.2.3 Définition de filtres

Il est possible d'appliquer des filtres à une table de manière à ce que seuls certains enregistrements soient affichés. Cela sans avoir à utiliser une requête SQL.

Pour cela il faut utiliser les méthodes SetRangeStart, SetRangeEnd (ou SetRange ()), ApplyRange et CancelRange du composant TTable.

Un filtre peut être constitué de deux valeurs (une maximale et une minimale) indiquant les limites des valeurs affichées pour un champ donné.

Exemple :

```
begin
    with TTable1 do
    Begin
        SetRangeStart ;
        FieldByName ('prix ' ). AsInteger := 100 ;
        SetRangeEnd ;
        FieldByName ('prix ' ). AsInteger := 1000 ;
        ApplyRange ;
    end ;
end ;
```

On aurait pu aussi faire :

```
    begin
        TTable1.SetRange ([100],[1000]);
    end ;
```

{Dans cet exemple seuls les enregistrements dont la valeur du champ 'prix' est comprise entre 100 et 1000 seront affichés}.

- La méthode CancelRange supprime toutes les contraintes d'étendue appliquées préalablement avec les méthodes SetRange…

14.3 Contrôle de validité des données

On a vu dans les chapitres précédents qu'il était possible de contrôler, de différentes manières, les données entrées dans une zone de saisie (chapitre 33).

Les composants "orientés données" permettent de réaliser divers contrôles de saisie, sans programmation, qui se révèlent suffisant dans bien des cas. Cependant, pour un contrôle accru, il faudra utiliser les méthodes définies précédemment.

- L'utilitaire DBD permet, lors de la phase de conception des tables de créer des marques de validité des divers champs. Malheureusement les masques ainsi définis ne sont pas pris en compte par Delphi. Ils ne sont donc pas utilisables.

14.3.1 Contrôles de base

Avant toute configuration des différents composants, des contrôles automatiques sont réalisés sur le type de donnée entrée. De ce fait :

- Il n'est pas possible d'entrer un caractère alphabétique dans un TDBEdit connecté sur un champ numérique (entier, numérique ou monétaire).
- Le format "monétaire" est affiché, pour un champ défini à ce type, dès que l'enregistrement est validé.
- Il n'est pas possible d'entrer un nombre réel dans un champ de type entier.

- Les différents séparateurs et caractères spécifiques permettant de spécifier les formats décimaux, les dates et heures et les format monétaires sont ceux spécifiés dans le ' Panneau de configuration | International ' de Windows.

Pour les champs de type date ou heure, aucun contrôle n'est fait à la saisie mais une exception est générée lors de la validation (qu'il s'agit alors de gérer). Les champs corrects sont validés.

14.3.2 Limitation de la longueur de la zone de saisie

La propriété MaxLenght, des composants TDBEdit utilisés, permet de définir le nombre de caractères maximal pouvant être entré dans la zone de saisie.

- Cette propriété n'est active que pour des champs de type chaîne de caractères.

Utilisation de la méthode IsValidChar ()

Les objets de type TField comportent la méthode IsValidChar () qui permet de tester si un caractère entré est valide.

Son prototype est :

```
function IsValidChar (InputChar: Char): Boolean; virtual;
```

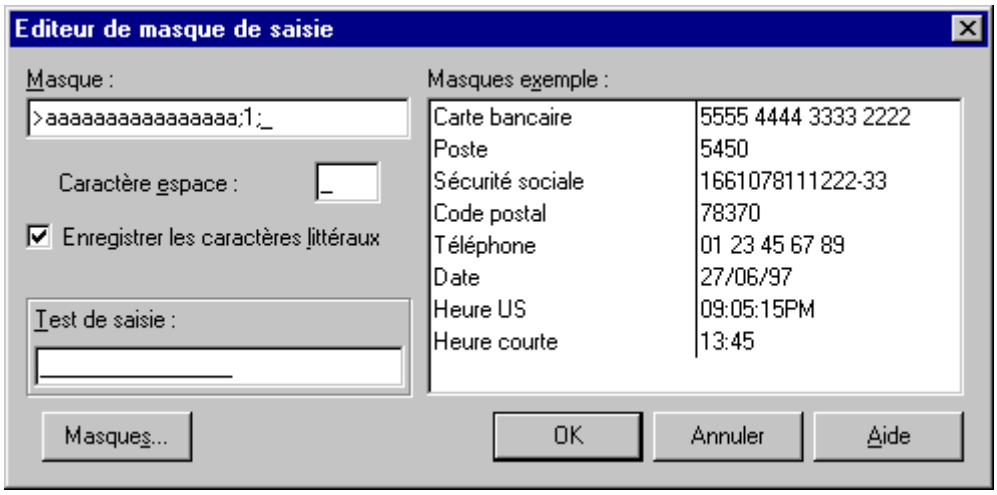
La méthode est surtout utilisée pour vérifier que des caractères numériques sont bien entrés dans la zone de saisie. Si l'objet TField est de type :

Type TField	Caractères autorisés
TIntegerField	Caractères ' 0' à ' 9' ainsi que ' +' et '-'
TSmallintField	
TWordField	
TBCDField	Idem plus 'E' , 'e' et le caractère de séparation décimale
TFloatField	
TStringField	Tous les caractères

14.3.3 Utilisation de la propriété EditMask

Les différents objets TField, accessibles via l'éditeur de champ, que l'on appelle en double-cliquant sur le composant TTable, disposent d'une propriété EditMask qui permet de créer des masques de saisies.

Le fait d'activer cette propriété permet d'accéder à un éditeur de masque. Celui-ci permet de définir, en utilisant l'aide appropriée, des masques de saisie propres à chaque champ. Des modèles de masque sont proposés. Outre le fait qu'ils correspondent aux masques les plus fréquents, ils permettent de mieux comprendre la syntaxe un peu particulière de création de ces masques.



L'éditeur de masque accessible via la propriété EditMask

Exemples :

- Pour certifier que les caractères entrés seront mis en majuscules :
  >aaaaaaaaaa (autant de 'a ' que la longueur maximale souhaitée)

  Pour certifier que les caractères entrés seront mis en minuscules :
  <aaaaaaaaaa

Ces deux exemples sont intéressants pour les chaînes de caractères mais les espaces ne sont pas acceptés.

Pour des champs de type date, heure, ou téléphone il suffit d'utiliser les modèles proposés :

date : !99/99/00 ; 1 ; _

heure : !90:00 ; 1 ; _
téléphone : !00\-00\-00\-00 ; 1 ; _

Les délimiteurs spéciaux ' / ' , ' : ' ou ' - ' sont générés automatiquement.

- Pour les nombres : 00000 ; 1 ; _
  (autant de ' 0 ' qu'il y a de chiffres à entrer)

Malgré le masque ainsi réalisé, il y a génération d'une exception lors de la validation, si le format de la date ou de l'heure n'est pas valide (13° mois, 25° heure, etc ...).

14.3.4 Utilisation des propriétés Editformat et DisplayFormat

Pour les champs numériques :

- La propriété DisplayFormat est utilisée pour formater la valeur du champ à l'affichage.
- La propriété EditFormat est utilisée pour formater la valeur du champ à stocker.

On peut utiliser aussi ces deux propriétés pour les champs de type date, time et monétaire lorsque l'on ne souhaite pas se conformer aux indications du panneau de configuration.

Ces propriétés sont très mal documentées.

14.3.5 Gestion d'une erreur due à une violation de clé

Lorsque l'on manipule des bases de données il est particulièrement nécessaire de veiller à ne pas violer l'intégrité des clés primaires mises en place car une telle erreur génère une exception plantant l'application.

Une des méthodes que l'on peut utiliser pour éviter ce problème est de ne pas laisser l'utilisateur entrer lui même les champs correspondant aux différentes clés (ces champs sont remplis automatiquement par programmation avec des routines évitant toute erreur). Mais cela n'est pas toujours possible.

Il faut donc être en mesure de gérer l'exception qui peut être générée lors de la violation de la clé.

Exemple :

```
procedure TForm1.BOKClick(Sender: TObject);
begin
      try
            { entrée dans le gestionnaire d'exception }
            TJournees.Post ;

      except     { Une violation de clé s'est produite }
            ShowMessage (' Ce numéro d''index est déjà utilisé');
            TJournees.Cancel ;
            { Efface l'enregistrement en cours de validation }
            DBIndex.SetFocus ;
            { Le focus est placé de nouveau dans le champ de saisie de l'index }
      end ;
      { Fin du bloc de gestion de l'exception }

      TJournees.Append ;       { Préparation d'un nouvel  enregistrement }
      DBIndex.SetFocus ;
end;
```

14.4 Utilisation optimisée des composants

14.4.1 Accès rapide à un enregistrement

Lorsqu'une table contient un grand nombre d'enregistrements, les méthodes standards fournies par Delphi pour accéder à un enregistrement particulier se révèlent insuffisantes : le composant navigateur ne permet que le déplacement enregistrement par enregistrement et l'utilisation des ascenseurs d'une grille se révèle laborieuse et aléatoire.

On peut alors être tenté d'utiliser un composant de type TDBListBox ou TDBComboBox pour accéder rapidement à un enregistrement. L'idée de départ est la suivante : le composant affiche (sous forme d'une liste ouverte ou déroulante) les différentes valeurs d'un champ de la table (par exemple tous les champs 'Nom' de la table). Il suffit alors, pour l'utilisateur de sélectionner le nom recherché pour que l'index dans la table se déplace et se positionne sur l'enregistrement souhaité.

Pour mettre en œuvre cette idée séduisante il faut en fait réaliser diverses opérations et surtout se rendre compte que l'idée de départ est dangereuse. En effet si l'on utilise tels quels des composants de type "liste orientés données" (TDBListBox ou TDBComboBox) on va corrompre la base car la valeur sélectionnée dans la liste devient la valeur correspondante du champ dans l'enregistrement courant (sans qu'il y ait par ailleurs déplacement).

Il est donc préférable d'utiliser un composant de type liste (TListBox ou de préférence TComboBox) sans que celui-ci soit connecté à une base de données.

Exemple :

Pour ce genre de réalisation on aura besoin des composants suivants :

- Une table de données contenant un champ permettant la recherche rapide. Ce champ doit ne pas contenir de doublon (dans la mesure du possible il est souhaitable qu'il serve de clé).

  Dans l'exemple on utilisera une table IDENTITE contenant un champ 'Nom' .

- Un composant TTable nommé TIdentite connecté à IDENTITE et un composant TDataSource, nommé DSIdentite. Un objet de type TField est créé sur le champ nom (son nom est donc : TIdentiteNom).
- Un composant TDBGrid permettant d'afficher les enregistrements. Ce composant est nommé DBGIdentite et est connecté à la DataSource.
- Un composant de type ComboBox, appelé CBNom, utilisé pour accéder rapidement à un enregistrement.

▲ **Initialisation de la liste :**
A la création de la fenêtre contenant le composant CBNom il faut initialiser la liste déroulante de ce dernier.

Pour cela il suffit de parcourir la table :

```
TIdentite.Open ;
while not TIdentite.EOF do
begin
      CBNom.Items.Add (TIdentiteNom.Value);
      TIdentite.Next ;
end ;
TIdentite.First ;
```

Lorsque la ComboBox sera déroulée, elle affichera toutes les valeurs des champs noms contenus dans la table.

Ultérieurement, si la table est modifiée, par ajout ou suppression d'enregistrements, il faut penser à actualiser le contenu de la liste déroulante.

▲ **Déplacement dans la table :**
Une fois la liste initialisée, il faut gérer le déplacement dans la table après que l'utilisateur ait sélectionné une valeur. Pour cela il faut créer un gestionnaire d'événement associé à l'événement OnClick de la ComboBox.

On a alors le code suivant :

```
procedure TForm1.CBNomClick (Sender: TObject );
begin
      with TMembres do
      begin
            First ;
            SetKey;
            FieldByName('NOM').AsString:=CBNom.Items[CBNom.ItemIndex];
            GotoKey;
      end;
end;
```

{ Lorsque l'utilisateur clique sur un nom dans la liste déroulante, le curseur de la grille se positionne sur l'enregistrement correspondant }.

L'instruction ' First' est impérative car si l'on réalise plusieurs recherches successives, la recherche ne se fera qu'en "aval". Il faut donc se repositionner à chaque fois en tête de table.

Il n'est pas besoin de vérifier la valeur de retour de GotoKey car la liste a été initialisée avec les éléments mêmes de la table : les items proposés existent donc.

On aurait pu réaliser le même type de recherche au sein d'une interface de type formulaire : la grille est alors remplacée par un ensemble de composants de type TDBEdit (un par champ). Le résultat aurait été le même. Il se pose quand même le problème de l'affichage du champ 'Nom' puisque la ComboBox n'est pas connecté à ce champ. L'astuce consiste alors à placer un TDBEdit au dessus de la ComboBox (de manière à ce qu'il occupe la place de la zone édition de ce composant) et à le connecter sur le champ 'Nom' de la table. Lorsque le choix sera validé par l'utilisateur, la valeur qui apparaîtra alors

sera alors celle issue de l'enregistrement courant et non celle dupliquée à partir de la liste déroulante. Elle pourra éventuellement être modifiée.

Ce procédé est conforme à la philosophie "Windows" qui veut que l'utilisateur n'ait pas à mémoriser de données mais que celles-ci lui soit toutes proposées de manière à ce qu'il n'ait plus qu'à faire un choix.

14.4.2 Utilisation du composant TBatchMove

Ce composant permet de copier la structure d'une table (ou les données qu'elle contient) dans une autre table.

Ses principales propriétés sont :

Source	Spécifie l'ensemble de données source (un composant TQuery ou TTable) correspondant à une table source existante.
Destination	Spécifie un composant TTable correspondant à une table de base de données (la table cible peut déjà exister ou non).
Mode	Opérations à accomplir.
Mappings	A utiliser si les noms de colonne sont différents dans Source et dans Destination afin de contrôler le transfert de ces champs.
AbortOnKeyViol	Demande l'interruption de l'opération lorsqu'une violation d'intégrité (de clé) se produit.
AbortOnProblem	Demande l'interruption de l'opération si une erreur de conversion de type se produit.

La méthode Execute effectue le déplacement par lot, défini par Mode, de la table Source vers la table Destination.

Les différents modes de transfert possibles sont indiqués par la propriété Mode :

Valeur	**Action**
batAppend	Ajoute des enregistrements dans la table de destination. La table de destination doit exister. Il s'agit du mode par défaut.
batUpdate	Met à jour les enregistrements de la table de destination avec les enregistrements correspondants de la table source. La table de destination doit exister et disposer d'un index pour établir les correspondances entre les enregistrements.
batAppendUpdate	Si un enregistrement correspondant existe dans la table de destination, sa mise à jour est faite. Sinon, il est ajouté dans la table de destination. Cette dernière doit exister et disposer d'un index pour établir les correspondances entre les enregistrements.
batCopy	Crée la table de destination à partir de la structure de la table source. La table de destination ne doit pas exister. Dans le cas contraire, elle est supprimée.
batDelete	Supprime dans la table de destination les enregistrements correspondant à ceux de la table source. La table de destination doit exister et disposer d'un index.

Exemple :

```
BatchMove1.Source := Table1;
BatchMove1.Destination := Table2;
BatchMove1.Mode := batAppendUpdate;
BatchMove1.Execute;
```

Annexe1 : Codes clavier

CONST. SYMB	Val (hex)	Equivalent	Const. Symb	Val (hex)	Equivalent
VK_LBUTTON	1	Bouton gauche de la souris	VK_I	49	I
VK_RBUTTON	2	Bouton droit de la souris	VK_J	4A	J
VK_CANCEL	3	Ctrl + Break	VK_K	4B	K
VK_MBUTTON	4	Bouton milieu de la souris	VK_L	4D	L
VK_BACK	8	BACKSPACE	VK_M	4E	M
VK_TAB	9	TAB	VK_N	4F	N
VK_CLEAR	0C	Suppr	VK_O	4F	O
VK_RETURN	0D	ENTER	VK_P	50	P
VK_SHIFT	10	SHIFT	VK_Q	51	Q
VK_CONTROL	11	CTRL	VK_R	52	R
VK_MENU	12	ALT	VK_S	53	S
VK_PAUSE	13	PAUSE	VK_T	54	T
VK_CAPITAL	14	CAPS LOCK	VK_U	55	U
VK_ESCAPE	1B	ESC	VK_V	56	V
VK_SPACE	20	Espace	VK_W	57	W
VK_PRIOR	21	PAGE UP	VK_X	58	X
VK_NEXT	22	PAGE DOWN	VK_Y	59	Y
VK_END	23	END	VK_Z	5A	Z
VK_HOME	24	HOME	VK_NUMPAD0	60	0 (clavier num)
VK_LEFT	25	Flèche gauche	VK_NUMPAD1	61	1 (clavier num)
VK_UP	26	Flèche haut	VK_NUMPAD2	62	2 (clavier num)
VK_RIGHT	27	Flèche droite	VK_NUMPAD3	63	3 (clavier num)
VK_DOWN	28	Flèche bas	VK_NUMPAD4	64	4 (clavier num)
VK_SELECT	29	SELECT	VK_NUMPAD5	65	5 (clavier num)
VK_EXECUTE	2B	EXECUTE	VK_NUMPAD6	66	6 (clavier num)
VK_SNAPSHOT	2C	Impression d'écran	VK_NUMPAD7	67	7 (clavier num)
VK_INSERT	2D	Insert	VK_NUMPAD8	68	8 (clavier num)
VK_DELETE	2E	Suppr	VK_NUMPAD9	69	9 (clavier num)
VK_HELP	2F	HELP	VK_MULTIPLY	6A	*
VK_0	30	0	VK_ADD	6B	+
VK_1	31	1	VK_SEPARATOR	6C	
VK_2	32	2	VK_SUBSTRACT	6D	-
VK_3	33	3	VK_DECIMAL	6E	.
VK_4	34	4	VK_DIVIDE	6F	/
VK_5	35	5	VK_F1	70	F1
VK_6	36	6	VK_F2	71	F2
VK_7	37	7	VK_F3	72	F3
VK_8	38	8	VK_F4	73	F4
VK_9	39	9	VK_F5	74	F5
VK_A	41	A	VK_F6	75	F6
VK_B	42	B	VK_F7	76	F7
VK_C	43	C	VK_F8	77	F8
VK_D	44	D	VK_F9	78	F9
VK_E	45	E	VK_F10	79	F10
VK_F	46	F	VK_F11	7A	F11
VK_G	47	G	VK_F12	7B	F12
VK_H	48	H	VK_NUMLOCK	90	Verr. num
			VK_SCROLL	91	Arrêt defil