

SCDI Technologies



# Présentation de Borland Delphi

Gérard Métrailler Jr.

email: [gmetrail@di.epfl.ch](mailto:gmetrail@di.epfl.ch)  
<http://diwww.epfl.ch/~gmetrail>

**Version 1.0**

20 novembre 1995

*Ce document comporte une présentation de l'outil de développement «Delphi pour Windows» de la société Borland. Il s'agit d'un compilateur RAD (Rapid Application Development) basé sur un développement visuel de l'interface graphique. Il permet la création d'applications autonomes (sous forme d'exécutable ou de DLL), l'utilisation des fonctions OLE et MCI de Windows, des VBX, ainsi que des appels directs aux API.*

*Ce langage est de plus conçu pour la création de bases de données (dBase, Paradox, SQL 92) puissantes et rapides.*

# Préface

## Remerciement

Je tiens à remercier Alessandro Vernet <[avernet@di.epfl.ch](mailto:avernet@di.epfl.ch)>, Christian Mettler, Vojto Elias <[velias@di.epfl.ch](mailto:velias@di.epfl.ch)>, mon père Gérard ainsi que tous les autres, pour la précieuse aide qu'ils m'ont apportée lors de la préparation de ce document, ainsi que pour les corrections typographiques de celui-ci.

## Mention légale

Ce document est la propriété de Gérard Métrailler Jr. (© 1995).

Sa diffusion et sa copie, totale ou partielle, est autorisée pour une utilisation personnelle ou dans le cadre d'une association à buts non lucratifs, pour autant que cette mention légale y soit reportée dans son intégralité. Dans ces cas, sa distribution est de plus largement encouragée. Toute autre utilisation, notamment commerciale, nécessite un accord préalable de l'auteur.

**La responsabilité de l'auteur n'est en aucun cas engagée en ce qui concerne la validité des informations contenues dans ce document ou pour toute autre raison.**

Delphi est une marque déposée de Borland International, Inc.

Microsoft, Microsoft Windows et Visual Basic sont des marques déposées de Microsoft Corporation.

Tous les autres produits sont des marques déposées de leur société respective.

## Obtention de ce document

Ce document est distribué sous la forme d'un fichier PostScript compressé (format Zip) qu'il vous est possible de récupérer sur l'internet (World Wide Web: <http://diwww.epfl.ch/~gmetrail>). En récupérant ce fichier, vous acceptez la licence telle qu'elle est décrite ci-dessus.

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Pourquoi Delphi . . . . .	4
1.2	Qu'est-ce que Delphi . . . . .	5
1.3	Configuration requise . . . . .	7
1.4	Delphi contre Visual Basic . . . . .	7
1.5	Le futur de Delphi . . . . .	8
1.6	Autres sources d'informations . . . . .	10
1.7	A propos de ce document . . . . .	11
<b>2</b>	<b>La conception visuelle</b>	<b>13</b>
2.1	Les fichiers de Delphi . . . . .	13
2.2	L'interface graphique de Delphi . . . . .	14
2.3	Les objets VCL . . . . .	18
2.3.1	Les composants visuels . . . . .	19
2.3.2	Les composants non-visuels . . . . .	20
2.3.3	Création de VCL . . . . .	22
2.4	Les événements . . . . .	22
<b>3</b>	<b>Le langage ObjectPascal</b>	<b>25</b>
3.1	Le Pascal . . . . .	25
3.2	Les extensions ObjectPascal . . . . .	26
3.3	Le langage ObjectPascal en action . . . . .	27
<b>4</b>	<b>Les bases de données</b>	<b>31</b>
4.1	Exemple d'application . . . . .	33
<b>5</b>	<b>Les fonctions avancées</b>	<b>35</b>
5.1	Les applications Multimedia . . . . .	35
5.2	Gestion des exceptions . . . . .	36
5.3	Les fichiers DLL . . . . .	38
	<b>Index</b>	<b>40</b>

# Table des figures

1.1	Interface de Borland Delphi (IDE) . . . . .	5
1.2	Version 32-bits de Delphi . . . . .	9
2.1	Fenêtre principale de Delphi . . . . .	14
2.2	Formulaire contenant quelques VCL . . . . .	15
2.3	Fenêtre affichant le code source . . . . .	16
2.4	Inspecteur d'objet de Delphi . . . . .	16
2.5	Gestionnaire de projets de Delphi . . . . .	17
2.6	Paramétrage des options de debugging . . . . .	18
2.7	Éditeur de texte créé avec Delphi . . . . .	19
2.8	Composants non-visuels et objet TMenu . . . . .	21
2.9	«A Propos...» de l'éditeur de texte . . . . .	23
4.1	Expert de bases de données . . . . .	32
4.2	Application de gestion d'adresses . . . . .	33
5.1	Application Multimedia . . . . .	36

# Chapitre 1

## Introduction

### 1.1 Pourquoi Delphi

La programmation a toujours été une tâche longue et fastidieuse. Même pour la création de petites applications dans un but bien précis, le développement de programmes informatiques restait réservé à une minorité des utilisateurs d'ordinateurs.

MS-Windows simplifia la gestion des périphériques. Malheureusement dans le même temps, il créa une grande complexité quand à la gestion des API. Ce que le développeur gagnait en vitesse et en facilité d'utilisation d'un côté, il le perdait en grande partie de l'autre.

Plusieurs «essais» ont été effectués afin de faciliter le développement d'applications. Mais aucune solution ne permettait un développement rapide de petits utilitaires ou de grandes applications professionnelles.

La solution la plus aboutie fut certainement le Microsoft Visual Basic qui introduisit un développement de l'interface graphique de manière visuelle. De nombreux autres langages fonctionnant sur le même principe furent commercialisés.

Delphi est lui aussi basé sur ce même concepte tout en apportant grandes améliorations par rapport à ce qui existe sur le marché. Mais le développement avec ce nouveau langage s'effectue toujours suivant une méthode qui consiste à «dessiner» l'interface graphique puis à assigner des parties de programme à des évènements précis, ceci afin de devoir écrire le moins de code possible.

Delphi dispose en outre d'une bibliothèque de classes orientée objets (les VCL ou Visual Class Library). Grâce à celle-ci, l'interface d'une application peut-être créée extrêmement rapidement, et ce sans avoir de connaissances préalables spécialement approfondies. Dès lors, le développeur n'entre pratiquement plus en contact avec le système d'exploitation, à moins qu'il n'y tienne absolument.

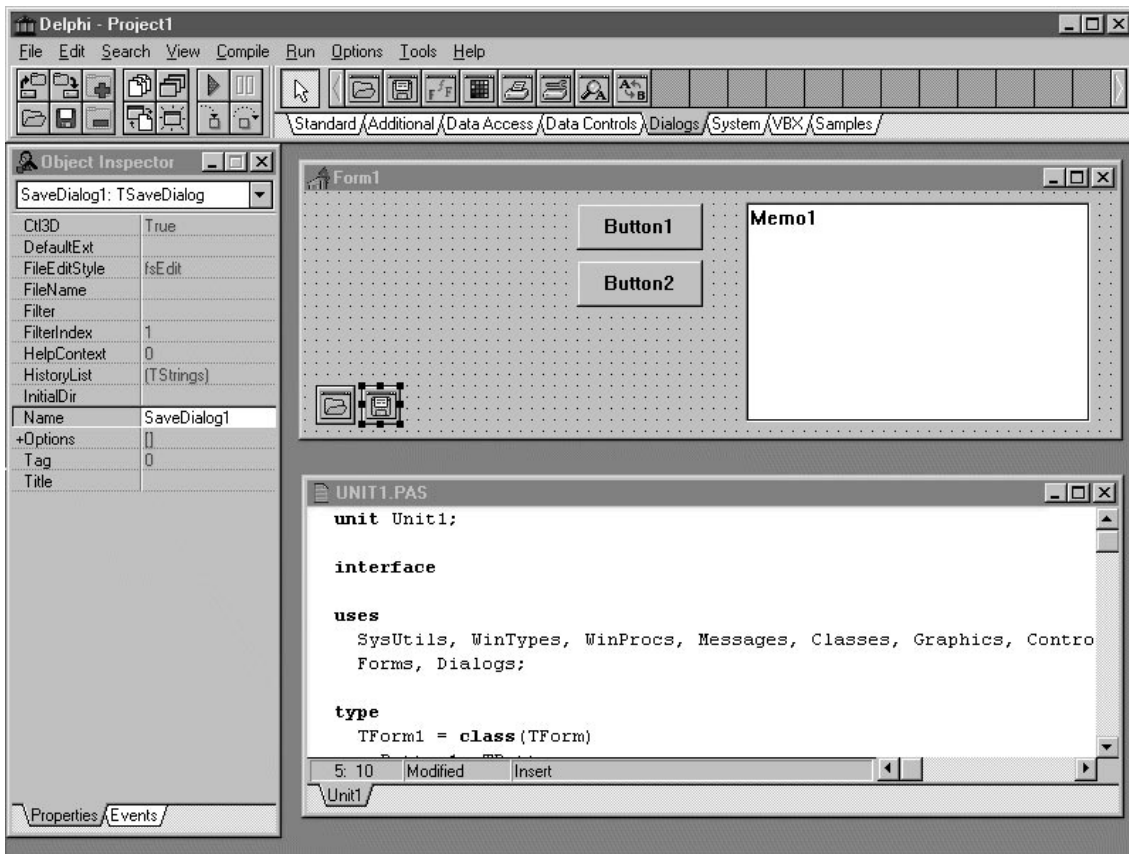


FIG. 1.1 - Interface de Borland Delphi (IDE)

## 1.2 Qu'est-ce que Delphi

Avant de commencer à vous décrire quelques unes des nombreuses possibilités de Delphi, regardons, à la figure 1.1, l'interface telle qu'elle se présente après le chargement du programme. Les fenêtres seront décrites au chapitre 2, à la page 13.

Ci-dessous sont regroupées les possibilités les plus importantes que Delphi offre aux développeurs. Elles seront en grande partie développées dans la suite de ce document.

- Delphi est un outil de développement puissant et rapide pour la programmation d'applications pour Windows 3.1x, 95 et NT.
- Delphi est un langage très facilement maîtrisable permettant de créer de petites applications, ceci en cachant au développeur tout les appels système. Cependant, s'il le désire, il peut accéder au plus profond de la programmation Windows (appels API, ...)
- La conception de l'interface graphique est conçue visuellement, un peu à la

manière des jeux de construction. On dispose pour ce faire d'une palette d'objets (les VCL) que l'on peut placer sur la fenêtre de développement (ils contiennent les barres de menus déroulants, les boutons, les boîtes de dialogue standard Windows, ...). Ces objets sont les pièces de ce jeu de construction. De plus, Delphi est capable d'utiliser les VBX (version 1.0) qui ont été, à l'origine, créés pour Visual Basic.

- Delphi est un langage de programmation de type Objet et Événementiel. C'est à dire qu'il attend un événement, tel un click de souris, pour effectuer une action précise qui aura été préalablement programmée.
- Ce langage permet de créer des applications de type SDI (Single Document Interface), c'est-à-dire utilisant une fenêtre principale et différentes boîtes de dialogue comme le Notepad de Windows 3.1, ou de type MDI (Multiple Document Interface), qui sont des programmes, comme Microsoft Word, permettant d'ouvrir simultanément plusieurs fichiers.
- Delphi permet de très facilement créer des Objets (VCL) qu'il nous sera possible soit d'incorporer plus tard dans nos propres applications ou de redistribuer afin que d'autres utilisateurs Delphi puissent les utiliser. Il permet aussi de créer des DLL (Dynamic Linked Library) qui pourront être accessible depuis de nombreuses autres applications (créés avec Delphi, Visual Basic, C++, dBase Windows, ...).
- Delphi est un véritable langage compilé, ce qui a comme principale conséquence de fonctionner beaucoup plus rapidement que des applications interprétées. Le compilateur génère, dans le cas d'une application simple, un unique exécutable, qui ne nécessite aucune librairie externe.
- Cet outil de développement permet la création rapide de programmes de bases de données locales (au format dBase, Paradox ou en utilisant les fonctions ODBC) ou distantes (SQL 92 tels que Oracle, Sybase ou Informix). Dans le cas des applications faisant appel à des serveurs SQL, il faudra disposer de la version Client/Serveur de Delphi.
- Le langage de programmation ObjectPascal, comme son nom l'indique, est un langage orienté objets (tel le C++) qui utilise donc les classes et le polymorphisme. Il gère aussi l'Exception Handling (traitement des erreurs contrôlées au niveau du programme).
- Delphi permet aussi de travailler avec des liens OLE et DDE, ainsi que l'utilisation des commandes MCI (pour les applications Multimedia). De plus, si les fonctions proposées par les VCL ne suffisent pas, il est toujours possible d'accéder directement aux API de Windows.

Delphi est donc un outil idéal pour la création de petites applications ne nécessitant pas de connaissance en programmation Windows. Mais cet outil de développement permet aussi, au programmeurs qui le désirent, un accès direct aux fonctions internes de Windows (les API) afin de pouvoir créer des applications professionnelles. Il existe par ailleurs déjà au moins un programme commercial développé en très grande partie avec Delphi. C'est Delphi lui-même. Il est donc possible de créer tout ce que l'on désire avec cet outil de Borland.

## 1.3 Configuration requise

D'après Borland, la configuration minimale afin de pouvoir utiliser Delphi consiste en:

- Ordinateur Compatible-PC 386 ou supérieur
- Microsoft Windows 3.1 ou suivant
- 6Mb de mémoire étendue ou plus
- Lecteur CD-Rom (Média sur lequel Delphi est livré)
- 30Mb de place disque.

Une installation complète de Delphi Desktop nécessite 80Mb de place disque. Il faut compter environ 10Mb supplémentaires pour la version Client/Serveur.

Il s'agit ici de la configuration requise pour l'outil de développement Delphi, et non pas de l'application créée qui fonctionne sur un ordinateur 286 ou supérieur. La quantité de mémoire requise pour les programmes créés avec Delphi varient suivant leur taille et leur codage<sup>1</sup>. 4Mb sont cependant suffisants dans la majorité des cas.

La configuration indiquée ci-dessus est le minimum «vital». Si on souhaite développer des applications assez importantes, il faut prévoir 8Mb de mémoire au minimum (16Mb sont préférable). La place disque variera en fonction du nombre de programmes développés et par leur taille (Code source et projets).

La taille disque et les besoins en mémoire peuvent paraître exagérés. A titre de comparaison, Microsoft Visual C++ 2.0 et Borland C++ 4.5 nécessitent tous environ 100Mb de place disque chacun.

## 1.4 Delphi contre Visual Basic

L'une des références en matière de développement visuel et événementiel d'applications est sans doute le Microsoft Visual Basic Pro. Ce langage en est à sa

---

1. Un programme de gestion de «grandes» listes par pointeurs nécessitera plus de mémoire qu'un petit éditeur de texte.

troisième version. La version 4 devrait être disponible très rapidement après la sortie de Windows 95 et permettra la création d'applications 32-bits (pour Windows 95 et NT). Il a été très longtemps le leader de sa catégorie. Delphi se présente donc comme son challenger le plus direct.

Tous deux servent, dans leurs versions actuelles, à développer des applications 16-bits pour Windows 3.1x. Mais la principale différence entre Delphi et Visual Basic est que le premier génère un exécutable compilé qui ne nécessite aucune librairie supplémentaire, alors que le second, comme il est «interprété», nécessite une DLL séparée (VBRUNx00.DLL). Le code généré par Visual Basic est par conséquent nettement plus lent lors de l'exécution.

Delphi utilise le langage ObjectPascal, qui est, comme son nom l'indique, un langage Orienté Objets, alors que Visual Basic est basé sur un langage structurel.

De plus, Visual Basic ne permet pas la création de DLL, et encore moins des composants qui le forment (VBX). Delphi est quand à lui, comme nous l'avons vu précédemment, capable de générer des DLL et des VCL (et même, semble-t-il, des VBX).

Visual Basic Pro gère les bases de données en utilisant le moteur de Microsoft Access. Delphi utilise le moteur BDE (Borland Database Engine). De ce point de vue, les deux systèmes sont équivalents. Par contre, Visual Basic ne permet pas d'accéder directement aux bases de données SQL distantes. Il faut pour ce faire recourir à des drivers ODBC. Cela a le désavantage de ralentir le fonctionnement de l'application. Ces deux outils de développement sont de plus proposés avec un «générateur de rapports» de base de donnée (ReportSmith 2.5 est livré avec Delphi et Cristal Reports avec Visual Basic Pro 3).

Visual Basic, tout comme Delphi, permet de travailler directement au niveau des API, d'utiliser les liens OLE, les fonctions MCI et ODBC, ainsi que d'accéder aux fichiers d'aide que nous associons à notre application...

Un des avantages de Visual Basic est qu'il est fourni avec un utilitaire permettant la création de disquettes d'installation, alors que Delphi en est dépourvu. Ce genre de programmes peut paraître secondaire, mais il devient très rapidement obligatoire lors de la création d'applications professionnelles. Il existe cependant un excellent Shareware, Chief Installer Pro, qui permet de palier à ce défaut.

Les deux langages ont donc leurs avantages et leurs inconvénients. Le choix se basera donc plus sur une question de «goût» personnel, à moins qu'une des caractéristiques proposées par l'un des deux outils de développement soit nécessaire au programmeur. Pour ma part, j'ai opté pour Delphi.

## 1.5 Le futur de Delphi

Actuellement, Delphi est un compilateur 16-bits. Cela signifie que les applications créées avec ce langage fonctionnent sur toutes les machines pouvant faire fonctionner Windows 3.1x en mode Standard (286 ou plus).

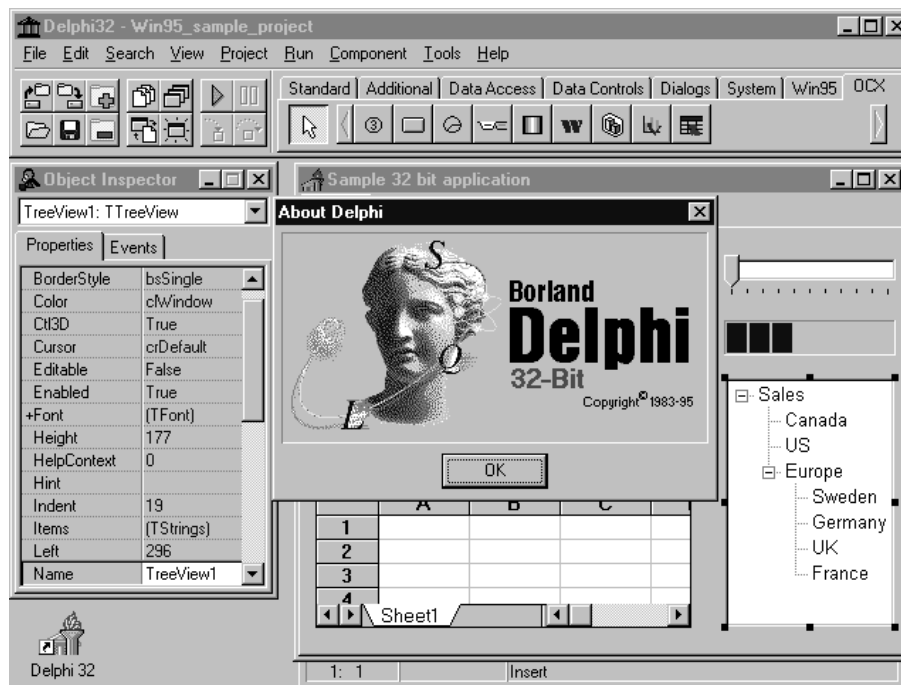


FIG. 1.2 - Version 32-bits de Delphi

Borland a cependant annoncé la disponibilité au plus tard 90 jours après la disponibilité de Windows 95 d'une version 32-bits de Delphi (cf. figure 1.2). Celle-ci permettra la création d'applications utilisant tous les systèmes d'exploitation 32-bits de Microsoft (Win95 et NT). Cette version devrait permettre:

- Des chaînes de caractères allouées dynamiquement d'une longueur illimitée au lieu des 64K actuels.
- Support des tables de caractères Unicode (pour Windows NT)
- Fonctionnement en multitâche préemptif des applications créées avec ce langage. Support des «threads». De plus, il semble qu'il sera facile de créer des applications multi-thread.
- La taille des exécutables sera réduite de 20% à 25% par rapport à Delphi 16bits. De plus, ces programmes devraient fonctionner au moins deux fois plus vite que celles créées avec la version actuelle<sup>2</sup>.
- Toutes les VCL sont 32-bits. De plus, de nouveaux composants permettent de créer des applications avec le look Windows 95.

---

2. Le travail d'optimisation du compilateur semble fantastique. Le code assembleur pourra toujours être inclus dans un programme Delphi, mais cela ne semble pas très utile si on considère les performances du compilateur

- Les OCX (successeurs des VBX) sont supportés par Delphi-32. De plus, cet outil de développement est capable de les créer simplement et rapidement.
- Les applications créées avec Delphi pourront être des «OLE Automation Controller». Cela signifie qu'elles pourront, par exemple, prendre le contrôle de Word afin de lui faire effectuer des tâches précises. Ceci toujours avec très peu de lignes de code.
- Delphi-32 utilisera le même générateur de code optimisé que le nouveau compilateur Borland C++. Cela signifie que les .OBJ créés avec Delphi pourront être linkés dans un projet en C++ et vice-versa.
- Delphi-32 pourra aussi afficher des alertes et des aides<sup>3</sup> lors de la compilation, et pas seulement en cas d'erreur comme c'est actuellement le cas.
- La prochaine version de cet outil de développement sera un produit disposant du «Windows 95 Logo», avec tout ce que cela implique. Il sera de plus très facile de créer des applications se conformant à ce logo.

Cette version devrait permettre un portage extrêmement rapide des applications développées avec la version actuelle de Delphi. Dans la majorité des cas, Borland annonce même qu'une simple recompilation avec Delphi 32-bits suffira (sauf si le programme fait des appels directs aux API de Windows 3.1x).

La compatibilité avec les version 3.1x de Windows serait cependant assurée en utilisant Win32s (module 32-bits pour Windows 3.1x en mode 386 Étendu).

Dans un futur bien plus éloigné, il est possible que Borland propose des versions OS/2 ou UNIX de Delphi, mais aucune annonce n'a été faite dans cette direction de la part de Borland. Il est certain que l'existence de ces versions dépendront du nombre d'exemplaires vendus sous Windows, ainsi que de la demande des développeurs.

## 1.6 Autres sources d'informations

Un nouveau langage a très rarement provoqué autant d'engouement que Delphi. De ce fait, les développeurs ont très rapidement voulu trouver des solutions pour s'échanger des algorithmes, VCL et programmes créés avec Delphi. Actuellement, plusieurs méthodes d'informations sont apparues sur les différents réseaux informatiques.

- **FidoNet**<sup>4</sup>

Sur ce réseau de courrier électronique, plusieurs aires de messages sont apparues concernant Delphi. Celle en langue française s'appelle **Delphi.fr**. Il

---

3. comme les variables inutilisées, boucles vides, etc.

4. Pour de plus amples informations quand aux réseaux informatiques et à FidoNet en particulier, vous pouvez vous reporter au «petit guide FidoNet» écrit par Alessandro Vernet [10].

est possible d'y poser toutes questions se rapportant à cet outil de développement.

– **Internet**<sup>5</sup>

Ce réseau des réseaux propose tout ce que nous pouvons désirer sur Delphi. Les informations principales sont accessibles depuis le World Wide Web (WWW). Voici quelques pages parmi les plus intéressantes:

- <http://www.borland.com/Product/Lang/Delphi/Delphi.html>

Il s'agit de la page officielle de Borland concernant Delphi. Vous y trouverez tout ce dont vous aurez besoin, et même plus. En particulier, c'est le meilleur endroit pour trouver les derniers patches ainsi que les annonces des nouveaux produits. De nombreux links sur d'autres pages concernant cet outil de développement y sont aussi disponibles. De plus, il y a des animations de plusieurs Mb. qu'il est possible de downloader<sup>6</sup>.

- <http://www.pennant.com>

The Delphi Connection. Il s'agit d'un journal Online réservé uniquement à Delphi. Il comporte de nombreux articles très intéressants, sur la création de composants, d'économiseurs d'écran, ...

- <http://www.teleport.com/~cwhite/wilddelphi.html>

The Delphi Station. Il s'agit d'un des sites concernant Delphi les mieux fournis en trucs et astuces, codes sources, composants freeware et shareware, ...

Il existe aussi de nombreuses informations concernant Delphi sur des réseaux comme Compuserve (tapper GO BORLAND).

De plus, il vous est toujours possible de joindre Borland directement afin qu'ils vous fassent parvenir de la documentation sur cet outil de développement.

## 1.7 A propos de ce document

Ces quelques pages ont été écrites pour vous présenter Delphi et quelques unes de ses nombreuses possibilités. Il n'a jamais été dans mon intention d'écrire un cours sur la programmation Delphi, un manuel de référence, ou tout autre livre qu'il est possible d'obtenir dans le commerce. Vous trouverez quelques références bibliographiques à la fin de ce document afin de vous permettre de choisir le

---

5. Réseau international connectant des millions d'utilisateurs à travers le monde. De type hétérogène, utilisant le protocole TCP/IP, internet permet un accès mondial à des informations.

6. Internet étant un réseau dépourvu de Bandwidth Management (c'est-à-dire en simplifiant, qu'il n'y a pas de gestion de la bande passante), la récupération d'un tel fichier peut prendre énormément de temps.

document que vous désirez acheter. Les livres publiées par Borland sont en fait les modes d'emploi livrés avec la version anglaise de «Delphi Desktop for Windows».

Je ne vais donc pas, par exemple, expliquer les bases de la programmation en ObjectPascal, mais je me contenterai de survoler rapidement les différentes possibilités offertes aux développeurs.

---

# Chapitre 2

## La conception visuelle

Ce chapitre consistera en une présentation de l'interface de développement de Delphi (l'IDE) et de quelques unes de ses fonctions. De plus vous y trouverez une explication sur les VCL. J'y aborderai l'utilisation des composants de base livrés avec Delphi.

### 2.1 Les fichiers de Delphi

Delphi travaille avec des projets. Il s'agit en fait d'un regroupement de divers fichiers (source + interface graphique) qui sont utilisés par le programme qui aura été développé. Celui-ci pourra ensuite ajouter des modules à l'intérieur du projet et Delphi s'occupera pratiquement entièrement de l'intégration de ceux-ci dans le projet. Lorsque nous débutons la création d'un nouveau projet (Menu Fichier|Nouveau Projet) Delphi génère automatiquement cinq fichiers:

- Le fichier principal du projet, qui s'appelle par défaut PROJECT1.DPR. Il s'agit d'un fichier texte ASCII. Il n'y a cependant normalement jamais besoin de le modifier, car Delphi s'en occupe (c'est par ailleurs assez déconseillé). Lors de la première sauvegarde de notre nouveau programme, Delphi va demander au programmeur un nouveau nom pour ce fichier. Le nom de l'application compilée (.EXE ou .DLL) prendra le même nom que ce fichier .DPR. A vous donc de bien le choisir...
- Le module principal s'appelle par défaut UNIT1.PAS. C'est aussi un fichier texte ASCII. C'est ici que le programmeur commencera à écrire le code source de son application. Lors de la première sauvegarde, il est conseillé de le nommer MAIN.PAS.
- Le fichier d'options, appelé PROJET1.OPT. Il s'agit aussi d'un fichier texte qui est utilisé par Delphi pour contenir tous les choix de configuration du projet. Il contient par exemple toutes les directives de compilation. Ce fichier prend le même nom que le fichier du projet principal.



FIG. 2.1 - Fenêtre principale de Delphi

- Le formulaire principal s'appelle par défaut UNIT1.DFM. Il s'agit d'un fichier « binaire » qu'il n'est possible de modifier qu'à partir de l'IDE (cf. section 2.2). Ce fichier contient toutes les informations sur la disposition du formulaire et les divers objets (VCL) qui le composent. Son nom est toujours identique au module principal.
- Le fichier bureau s'appelle par défaut PROJECT1.DSK. Ce fichier texte ASCII contient toutes les informations sur l'état de l'IDE.

Le développement d'une application avec Delphi peut se diviser en deux parties. La première consiste à construire l'interface en utilisant les VCL (cf. section 2.3), un peu comme un jeu de construction. Toute cette phase du développement simplifie grandement la tâche du programmeur, car celui-ci n'a plus besoin de gérer les divers appels aux API de Windows pour dessiner une fenêtre, un bouton ou tout autre objet de l'interface graphique. La seconde partie du développement consiste à écrire le code source correspondant aux divers événements qui peuvent se produire (click de la souris sur un bouton, édition de texte, message d'erreur de Windows, ...). Cette partie revient en grande partie à de la programmation classique en langage ObjectPascal (cf. chapitre 3, à la page 25).

## 2.2 L'interface graphique de Delphi

Une fois l'installation de Delphi terminée, il nous est enfin possible de démarrer l'application. Après quelques instants de chargement, Delphi nous apparaît (cf. figure 1.1, à la page 5). Il s'agit là de ce que Borland désigne par l'IDE (Integrated Development Environment). C'est depuis cet écran que pratiquement tout se passe. Il nous est possible de tout modifier, que ce soit l'aspect visuel de l'application ou son code source, les directives de compilation et même les différents composants (VCL) disponibles.

Cet IDE peut être séparé en plusieurs parties visibles à l'écran.

- **La fenêtre principale:** (cf. figure 2.1).  
Cette fenêtre longue et étroite se trouve en haut de l'écran. C'est ici qu'il est possible d'accéder à pratiquement toutes les fonctions de l'environnement de programmation. Il y est possible de charger ou fermer des projets, d'afficher

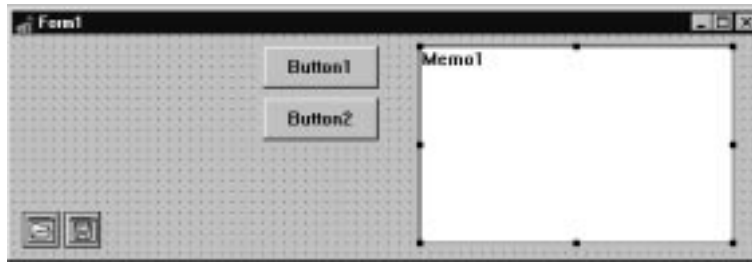


FIG. 2.2 - Formulaire contenant quelques VCL

les fenêtre décrites ci-dessous, de lancer la compilation ou encore d'accéder au système d'aide. Cette fenêtre principale se décompose en trois parties distinctes. Premièrement, une barre de menus qui permet d'accéder aux fonctions de Delphi décrites ci-dessus et bien plus. Deuxièmement, il y a une barre d'outils qui permet d'accéder plus rapidement aux options les plus importantes. Enfin, une barre des composants s'y trouve aussi. C'est depuis celle-ci que nous choisirons les objets (VCL) que nous désirons mettre sur le(s) formulaire(s).

– **Le formulaire:** (cf. figure 2.2).

C'est dans cette fenêtre que nous allons construire notre interface graphique, en utilisant les objets se trouvant dans la barre des composants. Pour créer cette interface, il nous suffit de «prendre» les composants dans la fenêtre principale pour les placer à l'endroit désiré sur le formulaire. On peut par exemple y déposer des boutons, des boîtes de saisie ou des zones de texte. De plus, il est possible de réarranger leurs dispositions respectives à n'importe quel moment du développement en utilisant la méthode du «Glisser et Déplacer». Le formulaire est par ailleurs lui-même un objet VCL, ce qui permet sa configuration à l'aide de l'inspecteur d'objets (cf. 2.2). C'est ici que Delphi simplifie et accélère réellement le développement d'une application Windows. Avec Delphi, il suffit de «dessiner» l'interface.

– **Le code source:** (cf. figure 2.3).

Cette fenêtre comporte le programme que vous avez écrit ou que Delphi a généré automatiquement. Il s'agit d'un éditeur de texte ayant des fonctions évoluées conçues pour faciliter le développement, comme la mise en évidence de la syntaxe ObjectPascal. Les options d'affichage du texte et de la syntaxe sont par ailleurs paramétrables (changement de police de caractères, de couleur, de forme, ...).

– **L'inspecteur d'objets:** (cf. figure 2.4).

Cette fenêtre sert à modifier les propriétés de la VCL sélectionnée (onglet «propriétés» au bas de la fenêtre). Elle les affiche dans leur ordre alpha-



```
UNIT1.PAS
unit Unit1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Contro
  Forms, Dialogs;

type
  TForm1 = class(TForm)
```

FIG. 2.3 - Fenêtre affichant le code source

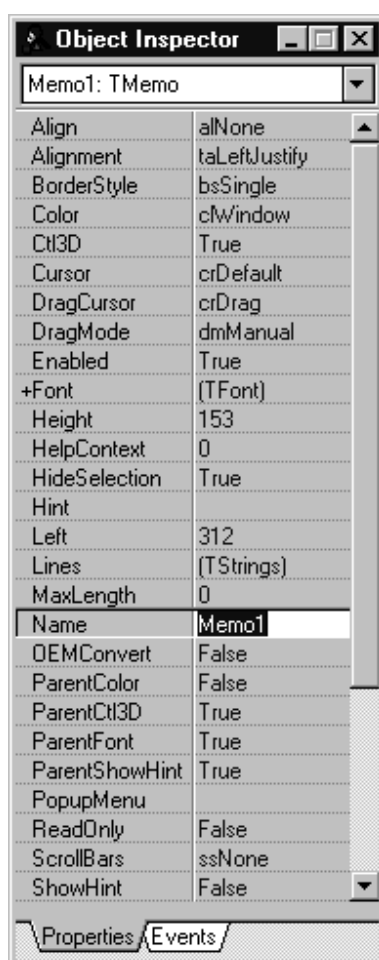


FIG. 2.4 - Inspecteur d'objet de Delphi

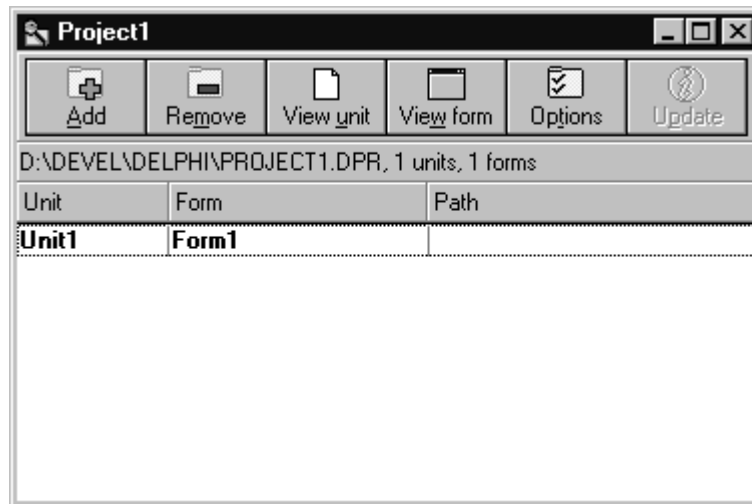


FIG. 2.5 - *Gestionnaire de projets de Delphi*

bétique. Chaque VCL pouvant avoir des propriétés différentes, l'inspecteur d'objets adapte automatiquement son contenu au composant sélectionné. Il existe de très nombreuses propriétés. Par exemple, en modifiant la propriété «Caption» d'un formulaire, nous changeons le texte de la barre de titre du formulaire sélectionné. L'inspecteur d'objets sert aussi à assigner un bout de code à un événement (onglet Événements au bas de la fenêtre). Par exemple, si nous assignons une procédure à l'événement OnClick d'un bouton, cette procédure sera exécutée à chaque fois qu'un utilisateur de votre application cliquera sur ce bouton. Il s'agit d'une des parties les plus utilisées lors de la création de l'interface graphique (cf. section 2.3, à la page 18).

– **Le gestionnaire de projets:** (cf. figure 2.5).

Cette fenêtre permet d'avoir une vue d'ensemble d'un projet. Son utilisation n'est pas très utile pour de petits projets n'utilisant qu'un seul formulaire, mais devient très vite nécessaire dès que notre projet prend plusieurs modules et formulaires. A l'aide des boutons placés au haut de la fenêtre, il nous est possible d'ajouter, de retirer ou de visualiser un module ou un formulaire. De plus, nous pouvons accéder aux options du projet (tel que les options de debugging de la figure 2.6).

– **Les options de Debugging:** (cf. figure 2.6).

Il s'agit d'une des nombreuses fenêtres de configuration du projet en cours. Elle permet de spécifier les options de debugging, ainsi que les options de compilations comme une vérification et la correction de l'erreur de calcul des Pentium dans les programmes créés. Lors de la création d'applications, il est recommandé de mettre toutes les options de debugging, ceci afin de

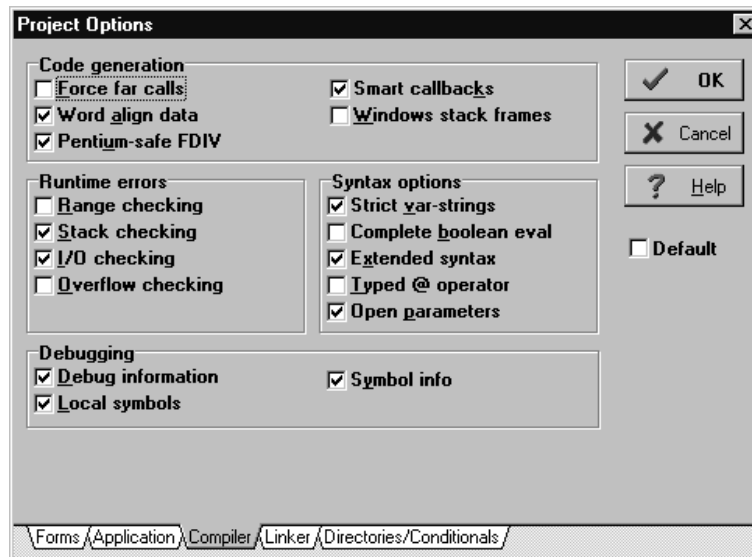


FIG. 2.6 - Paramétrage des options de debugging

faciliter la correction des erreurs. Ces options agrandissent quelques peut la taille de l'application compilée. Pour les versions finales, il est tout à fait possible d'enlever ces options.

C'est ainsi que se termine la visite guidée de l'interface graphique (IDE). Je vais maintenant regarder certaines des VCL de base, ainsi que le principe général de fonctionnement.

## 2.3 Les objets VCL

Comme je l'ai déjà dit précédemment, l'interface graphique de l'application se construit par de simples déplacements d'objets depuis la barre de composants sur le formulaire de travail, un peu comme un jeu de construction. C'est principalement cette partie du développement qui est effectuée bien plus rapidement avec Delphi qu'avec un langage de programmation traditionnel.

Il est à noter que les termes composants et VCL ont une signification identique dans ce document. Ils seront donc utilisés indifféremment pour désigner les mêmes objets.

Afin de pouvoir démontrer la facilité de création d'une application sous Windows, la meilleure solution c'est très certainement de passer à la pratique par un exemple. C'est pourquoi nous allons suivre, au cours de ce chapitre et du suivant, la construction d'un petit éditeur de texte (cf. figure 2.7). Celui-ci est une version légèrement simplifiée de l'exemple proposé par Borland dans le mode d'emploi principal de Delphi [2]. La version proposée ici ne permet que d'afficher un do-

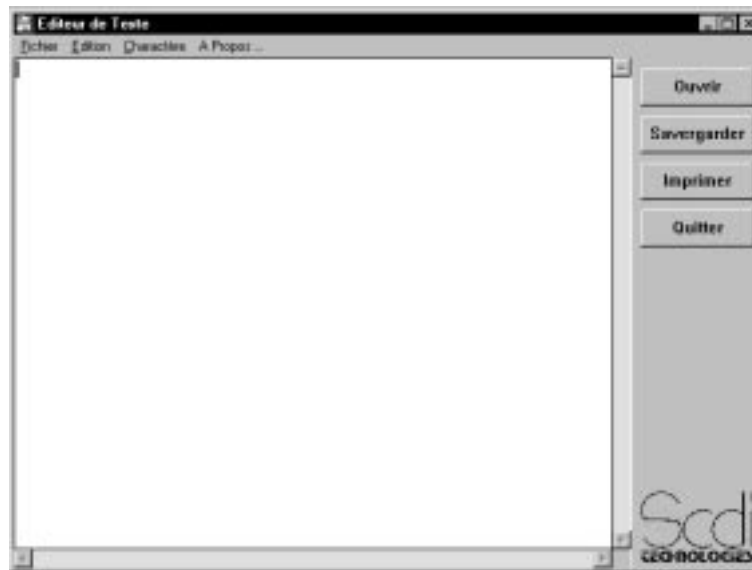


FIG. 2.7 - Éditeur de texte créé avec Delphi

cument à la fois (c'est une application SDI), alors que celle se trouvant dans le mode d'emploi peut afficher plusieurs fichiers texte simultanément (application MDI). La présentation visuelle a aussi quelque peu changé.

Les composants qui forment une applications peuvent être séparés en deux types principaux qui sont détaillés aux sections 2.3.1 et 2.3.2. Ceux-ci peuvent être paramétrés soit lors du développement en utilisant l'inspecteur d'objets, soit lors de l'exécution, en les modifiant par du code.

### 2.3.1 Les composants visuels

Nous désignons par «composants visuels» ceux qui sont visibles non seulement lors de la conception du programme, mais aussi lors de son exécution.

Dans la fenêtre principale de notre petit éditeur de texte, nous allons utiliser quatre types de composants visuels.

#### 1. TMenu

Il s'agit du composant qui gère les menus de la fenêtre sur laquelle il est placé. Une fois que nous avons posés cet objet sur le formulaire, un petit carré avec l'icône d'un menu apparaît. Afin de commencer réellement la création du menu, un double-click sur ce composant suffit. Une fenêtre apparaît. Il ne nous reste plus qu'à entrer les différents choix offerts à l'utilisateur du programme, de façon très intuitive. Au fur et à mesure de la création du menu, celui-ci apparaît sur le formulaire principal. Une fois terminé la «construction» du menu, il ne nous reste plus qu'à assigner les

événements relatifs aux différents choix de l'utilisateur (cf. section 2.4, à la page 22).

## 2. **TButton**

C'est avec ce composant que nous créons des boutons à cliquer. En déposant cette VCL, Delphi crée le bouton en lui assignant un nom générique. Il nous faut encore renommer ce composant à l'aide de l'inspecteur d'objets (propriété Caption et Name). Dans la fenêtre de l'éditeur de texte, il y a 4 boutons qui peuvent servir de raccourci aux options du menu Fichier. Il nous faudra encore écrire la partie du code correspondant aux clics de ces boutons (dans le cas de l'éditeur de texte, le code est identique à celui qui est exécuté en passant par le menu Fichier).

## 3. **TMemo**

Ce composant prend le plus de place sur notre formulaire. Il s'agit en effet de l'endroit où nous allons visualiser et taper du texte. Sa création est tout aussi aisée que celle des boutons. Quelques propriétés peuvent encore être modifiées à l'aide de l'inspecteur d'objets. Plusieurs de ces propriétés seront modifiées lors de l'exécution, par exemple lorsque l'utilisateur choisit une option du menu.

## 4. **TImage**

Il s'agit de l'image affichée dans le coin inférieur droit de la fenêtre principale. Cela peut paraître moins évident que ce logo soit aussi une VCL. Il s'agit en fait d'une «boîte» capable d'afficher un graphique. Le fait que celle-ci soit aussi un composant permet de faciliter grandement sa construction. Il nous suffit donc, comme pour toutes les autres VCL, de la placer à l'endroit désiré sur le formulaire et de lui modifier quelques propriétés afin de disposer d'une image visible lors de l'exécution.

Delphi est livré avec un très grand nombre de composants, dont la majeure partie est visuelle. Les autres forment ce que Borland appelle les composants non-visuels.

### 2.3.2 Les composants non-visuels

Ceux-ci ne sont pas visibles (tout au moins pas directement) lorsque l'application est exécutée. Le développeur ne voit qu'une petite icône représentant l'objet qu'il a placé sur le formulaire. Ces VCL gardent cependant toutes les possibilités de paramétrage qu'offrent les composants visuels, toujours en utilisant l'inspecteur d'objets ou en les utilisant dans le code source. Notre éditeur de texte utilise cinq composants non-visuels. Il s'agit de composants qui effectuent la transition entre les boîtes de dialogue standard de MS-Windows et Delphi, afin que l'utilisateur n'ait pas à accéder directement à la programmation système. Tous les



FIG. 2.8 - Composants non-visuels et objet TMenu

composants énumérés ci-après sont toujours placés sur le formulaire (cf. figure 2.8).

1. **TOpenDialog**

Ce composant permet d'afficher la boîte de dialogue standard de Windows afin d'ouvrir un fichier. Une fois que l'utilisateur a sélectionné ce fichier, cet objet garde le nom et le chemin du fichier sélectionné. Il ne reste plus qu'à l'afficher dans le composant TMemo (cf. section 2.3.1). Cette opération est par ailleurs gérée par la VCL TMemo.

2. **TSaveDialog**

Cette VCL affiche la boîte de dialogue de Windows pour sauvegarder le fichier. Son fonctionnement est identique à celui de l'objet TOpenDialog.

3. **TFontDialog**

Ce composant sert à changer la police de caractère de l'application en cours d'utilisation. Pour ce faire, elle affiche la boîte de dialogue de Windows qui gère les «fonts». Une fois que l'utilisateur a choisi la nouvelle police et ses caractéristiques, le programme codé par nos soins effectue les modifications.

4. **TPrintDialog**

Cette boîte de dialogue permet d'imprimer le texte en cours d'édition sur le périphérique choisi par l'utilisateur. La partie du code qui gère l'impression est cependant un peu plus «compliquée» que les autres opérations mentionnées ci-dessus. C'est un point que Borland améliorera certainement lors de la prochaine version de Delphi.

5. **TPrinterSetupDialog**

Ici, nous pouvons configurer l'imprimante en utilisant les possibilités offertes par Windows. Il suffit de faire appel à cette VCL pour que Windows fasse apparaître la boîte de dialogue et qu'il effectue toutes les modifications nécessaires.

L'utilisation de ces composants non-visuels peut paraître «inutile» puisqu'il est aussi possible d'accéder directement à ces boîtes de dialogue en faisant appel aux API de Windows. Ils simplifient cependant grandement l'utilisation de ces fonctions standards de Windows. De plus, lorsque la nouvelle version de Delphi 32-bits sortira sur le marché, les applications utilisant exclusivement des VCL ne nécessiteront qu'une recompilation afin de pouvoir accéder à toutes les fonctions de Windows 95 et NT alors que les programmes effectuant de nombreux appels API devront être en grande partie modifiés pour pouvoir accéder aux nouvelles fonctionnalités du système d'exploitation.

Il est donc vivement recommandé d'utiliser le plus possible les VCL fournis par Borland ou par des programmeurs indépendants plutôt que d'accéder directement aux API (ce qui est toujours possible si cela s'avère nécessaire).

### 2.3.3 Création de VCL

L'utilisation d'objets tels les VCL plutôt que des VBX réside dans le fait qu'il est très facile de créer ses propres composants. Delphi étant lui-même créé avec ces VCL, cela permet de se faire une idée des possibilités d'application.

De nombreux utilisateurs de Delphi ont déjà créé des composants forts utiles qu'ils proposent gratuitement ou contre une assez faible rémunération. Comme leur développement peut être assez long et fastidieux, il est très fortement recommandé de vérifier si une VCL effectuant une tâche précise n'existe pas déjà. Ceci partant du principe qu'*il ne faut pas réinventer la roue à chaque fois*<sup>1</sup>.

L'objectif de cette présentation n'étant pas d'aborder tous les sujets, vous pouvez, si cela vous intéresse, vous reporter à la documentation de Delphi [1] ou à des livres traitant le sujet [5][9].

## 2.4 Les événements

Les programmes fonctionnant sous Windows sont dans la grande majorité des cas des applications qui attendent un ordre de l'utilisateur afin d'effectuer une action prédéterminée (une partie du code). Le programme devra par exemple réagir différemment à un click de souris sur un bouton ou dans le menu. L'application attend donc un événement bien précis et réagira en conséquence.

Delphi résout le problème de la gestion des événements en assignant une partie du code (une procédure ou fonction) à un événement d'un composant. Le programme fera appel automatiquement à cette partie lorsque l'événement s'est produit.

---

1. Conseil donné dans tous cours de programmation: si un algorithme existe déjà, il vaut mieux l'utiliser plutôt que d'essayer de le réinventer. De plus, cela prendra beaucoup de temps et il sera difficile de créer un algorithme meilleur que celui qui aura déjà été optimisé durant des années.



FIG. 2.9 - «A Propos...» de l'éditeur de texte

Par exemple, lorsque l'utilisateur click une fois sur le bouton «Ouvrir», Delphi va exécuter la procédure qui est assignée à l'événement «OnClick» de ce composant. Dans notre cas, l'application exécutera la procédure:

```
PROCEDURE TTextForm1.Ouvrir1Click(Sender: TObject);
BEGIN
    ...
END;
```

Toute cette partie de code (sans ce qui est symbolisé par les ...) est générée automatiquement par Delphi lorsque nous assignons une nouvelle procédure à un événement donné. Cette procédure reçoit en paramètre une variable Sender qui permettra au programme de réagir de diverses manières en fonction de l'origine de l'appel (par exemple depuis le menu Fichier ou depuis le bouton). Il nous faut uniquement compléter les «...» afin d'effectuer une tâche particulière (cf. chapitre 3, à la page 25).

Un autre événement se produit lorsque l'utilisateur click sur le menu «A Propos ...». Dans ce cas, Delphi va afficher une nouvelle fenêtre, que nous aurons construit de la même manière que le formulaire principal, contenant les informations quand au programme utilisé (cf. figure 2.9). En fait, le programme exécute la procédure:

```
PROCEDURE TTextForm1.APropos1Click(Sender: TObject);
BEGIN
    AboutBox.ShowModal;
END;
```

à chaque fois que l'utilisateur click sur cette partie du menu. C'est ce bout de programme qui affiche la fenêtre «A Propos».

La conception événementielle de Delphi simplifie grandement le développement des applications, car il n'y a pas besoin de gérer les événements eux-même.

Cela signifie principalement que le programmeur devra écrire bien moins de code qu'avec un langage traditionnel.

Pour continuer l'analogie avec le jeu de construction, Delphi agit comme si nous accrochions une petite étiquette au composant placé sur le formulaire. A chaque fois que l'utilisateur utilise cet objet (par exemple il click dessus), le programme va regarder l'étiquette et agir en conséquence.

# Chapitre 3

## Le langage ObjectPascal

Une fois la création de l'interface graphique terminée, il nous faut encore écrire les procédures correspondant aux différents événements. Pour ce faire, Delphi propose d'utiliser le langage ObjectPascal. Il s'agit d'une version évoluée du Pascal, qui apporte la possibilité d'une programmation orientée objet (OOP). Je vais ici parcourir quelques unes des fonctions mises à disposition par ce langage. Pour ce faire, je vais continuer l'exemple de l'éditeur de texte (cf. figure 2.7, à la page 19) dont la fenêtre graphique a été décrite au chapitre 2.

### 3.1 Le Pascal

Le Pascal fut inventé au début des années 70 par Nicklaus Wirth, professeur à l'ETHZ. Il s'agissait de l'un des premiers langages à prendre en compte de façon cohérente les concepts de programmation structurée. Sa grande force réside dans la facilité d'apprentissage de la syntaxe qui se rapproche de la langue anglaise et donc de la compréhension du code. C'est la raison pour laquelle il est actuellement très utilisé comme outil d'enseignement et dans l'industrie. Il existe de nos jours de très nombreux compilateurs Pascal pour presque tous les systèmes d'exploitation courants.<sup>1</sup>

Ce petit document n'étant pas un cours de programmation, nous n'allons pas étudier la syntaxe du Pascal. Il existe cependant plusieurs livres qui traitent le langage Pascal [7] de manière spécifique.

Si vous souhaitez programmer avec Delphi, ceux-ci ne vous seront pas nécessaires, car le mode d'emploi [2] fourni par Borland, l'aide en ligne et les livres concernant Delphi [5][6] traitent suffisamment bien le sujet.

---

1. Borland Pascal pour Dos et Windows, Symantec Think Pascal pour Macintosh, GNU Pascal Compiler pour systèmes Unix, ...

## 3.2 Les extensions ObjectPascal

La programmation orientée objet (OOP: Object Oriented Programming) peut-être comparée aux «objets» de la vie courante. L'un des exemples les plus utilisés est celui du vélo (ou parfois de la voiture). Le principe est assez simple. Un vélo n'est à la base formé que d'un cadre, d'un guidon, d'une selle et de deux roues. Il s'agit donc ici de la définition d'un objet de type vélo. En ObjectPascal, les objets sont appelés «Class». Si nous désirions «programmer» un vélo, la syntaxe proposée par Borland ressemblerait<sup>2</sup> à:

```
TYPE
  TVelo = CLASS
    Cadre: Metal;
    Guidon: Metal;
    Selle: Mousse;
    Roue1: Pneu;
    Roue2: Pneu;
  END;
```

Or il existe sur le marché, des vélos contenant de nombreuses options. Mais même dans le cas où toutes les parties supplémentaires ont été ajoutées, la base est toujours formée d'un vélo tel qu'il a été défini ci-dessus. Il nous faut donc pouvoir utiliser l'objet de base qu'est TVelo afin de pouvoir lui ajouter des pièces à volonté. En ObjectPascal, le nouveau vélo se définirait donc de la manière suivante:

```
TYPE
  TVeloAvecOptions = CLASS(TVelo)
    PhareAvant: ampoule;
    PhareArrière: ampoule;
    PorteBaggage: Metal;
    Procedure ChangerLampe(MonVelo: TVeloAvecOptions);
  END;
```

Toutes les propriétés dont disposait le type TVelo sont toujours accessibles dans les objets de type TVeloAvecOptions. Comme vous pouvez le constater, les classes peuvent non seulement comporter des «variables», mais des procédures.

Par la suite, si le cycliste désire changer l'ampoule du vélo qu'il possède, il pourra le faire librement et facilement en effectuant l'opération :

---

2. Les types Metal, Mousse et Pneu ont déjà été définis auparavant

```
PROCEDURE TVeloAvecOptions.ChangerLampe(MonVelo: TVeloAvecOptions);
BEGIN
  MonVelo.PhareAvant := NouvelleAmpoule;
END;
```

La programmation orientée objet apporte donc une certaine discipline et structure à un projet. Cependant, ce type de programmation n'est pas toujours très facile à comprendre. C'est pourquoi Delphi s'occupe de toute la partie de création des classes correspondant aux formulaires construits de façon visuelle. Le programmeur n'a donc pas besoin de connaissances en OOP pour utiliser Delphi. Il s'agit une fois de plus de fonctions qui existent, mais dont le développeur peut, en grande partie, faire abstraction (en tout cas pour de petites applications). C'est Delphi qui s'en occupe.

De plus, le langage ObjectPascal dispose de fonctions de polymorphisme, ce qui permet de redéfinir certaines fonctions que la classe à reçue de l'un de ces «ancêtres». Les possibilités sont dès lors quasi illimitées.

### 3.3 Le langage ObjectPascal en action

Revenons donc à l'exemple de l'éditeur de texte afin de voir l'ObjectPascal en pratique. Comme signalé précédemment, Delphi génère une partie du code source automatiquement (cf paragraphe 2.4, à la page 22). Lorsque nous débutons un nouveau projet, Delphi va créer un module associé au formulaire principal qui contient déjà quelques lignes de code source.

```
UNIT unit1;

INTERFACE
USES
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
  Controls, Forms, Dialogs;
TYPE
  TForm1 = CLASS(TForm)
  private { Private declarations }
  public { Public declarations }
  END;
VAR
  Form1: TForm1;

IMPLEMENTATION
{$R *.DFM}
END.
```

Nous remarquons que de très nombreux autres modules sont déjà inclus. Ils permettent d'accéder à toutes les fonctions standard de Windows. De plus, nous constatons qu'il existe déjà un objet `TForm`. Il s'agit du formulaire principal. Celui-ci est encore vide. La partie de l'implémentation du code ne contient qu'une directive de compilation (`{$ R *.DFM}` signifie que Delphi doit inclure la fenêtre principale construite de façon visuelle).

En construisant l'interface graphique de notre application, Delphi va automatiquement compléter la classe `TForm1` en ajoutant les objets que nous aurons déposés sur le formulaire. Voici par exemple ce qui aura été ajouté automatiquement après avoir déposé les divers composants sur notre formulaire, sans pour autant encore avoir créé les différents menus :

```
TYPE
  TTextForm1 = CLASS(TForm)
    MainMenu1: TMainMenu;
    Memo1: TMemo;
    BtnOuvrir1: TButton;
    BtnSauve1: TButton;
    BtnQuitter1: TButton;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    FontDialog1: TFontDialog;
    PrintDialog1: TPrintDialog;
    BtnImprimer1: TButton;
    PrinterSetupDialog1: TPrinterSetupDialog;
    SCDIIcon: TImage;
  PRIVATE
    { Private declarations }
  PUBLIC
    { Public declarations }
  END;
```

Chaque ligne de cette classe est formée du nom de l'objet (propriété `Name` dans l'inspecteur d'objets) et de son type. Les menus viennent s'insérer dans cette classe au fur et à mesure de leur création. Chaque choix possible dans les différents menus prend une ligne et est déclarée de type `TMenuItem`.

Jusqu'à maintenant, tout le code source a été généré automatiquement par Delphi, en prenant en compte les modifications du formulaire et de ses divers composants dans l'inspecteur d'objet. Mais aucun événement n'est géré, et donc, si nous compilons et exécutons le programme, rien ne se produira.

Comme signalé au paragraphe 2.4, à la page 22, lorsque nous assignons un événement particulier à une `VCL`, Delphi génère automatiquement la partie du code définissant une procédure. Il ne nous restera plus qu'à compléter celle-ci afin d'effectuer la tâche souhaitée. Pour continuer l'exemple de l'éditeur de texte,

regardons donc la procédure gérant l'ouverture d'un fichier (qui est exécutée soit lorsque l'utilisateur sélectionne Ouvrir depuis le menu Fichiers, soit qu'il click sur le bouton «Ouvrir»).

```
PROCEDURE TTextForm1.Ouvrir1Click(Sender: TObject);
BEGIN
  IF OpenFileDialog1.Execute THEN BEGIN
    Filename := OpenFileDialog1.Filename;
    Memo1.Lines.LoadFromFile(Filename);
    Memo1.SelStart := 0;
    Caption := 'Editeur de Texte - ' + ExtractFileName(Filename);
    Memo1.Modified := False;
  END;
END;
```

Etudions donc un peu plus en détail cette procédure. Premièrement, nous faisons appel à la fonction booléenne «Execute» du composant `TOpenDialog`. C'est par cette commande que Delphi va afficher la boîte de dialogue standard pour ouvrir (ou plutôt charger) un fichier. Une fois que l'utilisateur a sélectionné le texte qu'il désire éditer, nous allons charger ce fichier (variable `OpenDialog1.Filename`) dans la boîte de texte. Pour ce faire, nous utilisons une fonction de la VCL `TMemo`. Comme nous désirions modifier le contenu des ses «lignes», il nous faut utiliser la commande `Memo1.Lines.LoadFromFile(...)`. Nous allons finalement modifier le texte qui apparait dans la barre de titre de l'application afin qu'elle reflète les changements effectués. Pour ce faire nous accédons à la propriété `Caption` de la VCL `TTextForm1`<sup>3</sup>. Le principe est à peu près identique pour la sauvegarde de fichiers, en utilisant la VCL `TSaveDialog`.

D'autres fonctions de notre éditeur de texte nécessitent encore moins de lignes de code. C'est par exemple le cas de l'utilisation du presse-papier pour couper - coller du texte. Voici les deux procédures, qui se passent de commentaires, correspondant aux clicks dans le menu «Edition».

```
PROCEDURE TTextForm1.Couper1Click(Sender: TObject);
BEGIN
  Memo1.CutToClipboard;
END;
PROCEDURE TTextForm1.Coller1Click(Sender: TObject);
BEGIN
  Memo1.PasteFromClipboard;
END;
```

---

3. Il est aussi possible d'écrire `TextForm1.Caption := '...'` à la place de `Caption := '...'`. Cela n'est cependant pas nécessaire, car Delphi part du principe qu'il s'agit, si nous ne signalons pas l'objet auquel appartient la propriété, de l'objet qui contient cette procédure (dans notre cas, le formulaire principal).

L'exemple de l'affichage de la boîte de dialogue va sembler tout de suite bien moins obscure. Pour mémoire, voici la procédure correspondante au click dans le menu.

```
PROCEDURE TTextForm1.APropos1Click(Sender: TObject);
BEGIN
  AboutBox.ShowModal;
END;
```

Cette procédure donne simplement l'ordre d'afficher la fenêtre **AboutBox**. La fonction **ShowModal** interrompt l'exécution de l'application tant que la fenêtre correspondante n'aura pas été fermée<sup>4</sup>.

Finalement, il nous faut aussi pouvoir quitter le programme, sans avoir à effectuer un double-click dans le menu **Système**. Pour ce faire, une simple ligne de code suffit.

```
PROCEDURE TTextForm1.Quitter1Click(Sender: TObject);
BEGIN
  Close;
END;
```

Cette commande ferme la fenêtre actuelle. Comme celle-ci est aussi l'écran principal de l'application, une exécution de cette procédure aura pour effet de quitter le programme.

---

4. Si nous voulons pouvoir afficher la boîte «A Propos...» en parallèle à l'exécution du programme, il nous faut utiliser la fonction **AboutBox.Show**.

# Chapitre 4

## Les bases de données

L'un des objectifs de Borland, lors de la conception de Delphi, a été de créer un outil de développement capable de concevoir simplement et rapidement des applications de bases de données puissantes. Il s'agit en effet de l'un des domaines de développement les plus importants pour la majorité des sociétés. Pour ce faire, de très nombreux moyens ont été mis à la disposition des programmeurs.

Delphi intègre un expert<sup>1</sup> permettant la création rapide d'applications utilisant des bases de données.

Afin de pouvoir les gérer, un programme doit aussi utiliser un «moteur» permettant d'effectuer la gestion des données. Celui fourni avec Delphi, se nomme BDE ou Borland Database Engine<sup>2</sup>. C'est le même que celui qui accompagne tous les produits de base de données vendus par Borland (dBase, Paradox, ...). Cela permet d'affirmer que s'il est possible de faire quelque chose avec dBase, vous pourrez aussi concevoir une application avec Delphi qui parviendra au même résultat. Ce gestionnaire permet d'utiliser plusieurs formats de syntaxe sans avoir à se soucier de leur fonctionnement interne. Les principaux sont:

### – dBase / Paradox

Il s'agit ici des deux bases de données vendues par Borland. Il est donc normal qu'ils soient supportés par Delphi. Ces bases sont fort utiles pour des données stockées localement. De plus, le format dBase est de nos jours encore un standard dans le monde du PC. Tous les systèmes de base de données sont capables d'utiliser ce format de fichiers (en importation et en exportation). Le format Paradox est cependant bien plus approprié dans la majorité des cas, car il permet une plus grande flexibilité<sup>3</sup>. De plus, l'exécution d'applications utilisant ces formats est très rapide et la conception

---

1. identique aux Assistants de Microsoft

2. anciennement appelé IDAPI ou Integrated Database Application Programming Interface

3. Le format Paradox autorise une description des champs de données allant jusqu'à 25 caractères, alors que dBase ne permet pas d'en avoir plus de 10. De plus ce premier format permet d'utiliser des espaces dans les définitions de champs, alors que cela est impossible avec dBase.

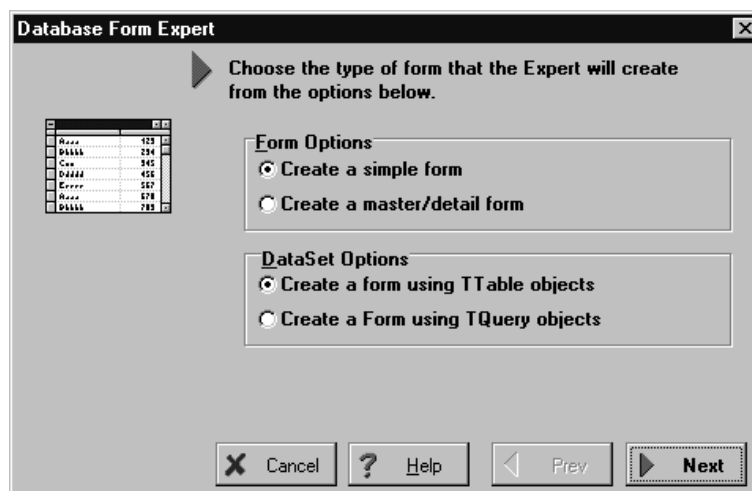


FIG. 4.1 - *Expert de bases de données*

de celles-ci est grandement simplifiée avec Delphi. Il s'agit donc des formats de prédilection pour l'accès aux bases de données locales.

#### – ODBC

Il peut arriver que vous désiriez accéder à une base de données utilisant un format qui n'est pas géré par le BDE. Pour ce faire, Delphi permet l'utilisation du format ODBC (Open Database Connection) de Microsoft. Il s'agit de l'équivalent du BDE de Borland, à cela près que le premier est un standard et permet de supporter à peut-être tous les formats connus sous Windows. Il s'agit en fait d'un jeu de commandes d'accès aux bases de données identique pour tous les formats auxquels nous désirons accéder. C'est le gestionnaire ODBC, ainsi que les drivers spécifiques, qui effectuera la conversion entre les commandes de la base de données et celles du format ODBC. Cela permet une très grande souplesse et facilite la programmation. De très nombreux drivers existent (notamment Access, Fox Pro, Informix, Oracle, ...). Malheureusement, aucun driver ODBC n'est fourni avec Delphi. Il vous faudra pour par exemple accéder à un fichier au format Access, acheter ce produit, afin de disposer du driver ODBC correspondant. Il faut cependant signaler que le gestionnaire ODBC a une exécution plutôt lente, comparé au BDE.

#### – SQL 92

L'accès aux bases de données locales, en utilisant les «formats» ci-dessus, est appropriée dans certains cas. L'intégration d'une base de données dans un réseau n'est cependant pas possible avec ceux-ci<sup>4</sup>. Pour ce faire, Delphi

4. sauf en utilisant des drivers ODBC pour des formats SQL tels Oracle, Informix, ...

FIG. 4.2 - *Application de gestion d'adresses*

permet d'accéder aux fichiers se trouvant sur un réseau en utilisant le format SQL (Structured Query Language). Il s'agit, comme son nom l'indique, d'un langage permettant d'effectuer des requêtes sur une ou plusieurs bases de données se trouvant sur un serveur. Ce format utilise une architecture dite Client/Serveur. Afin de pouvoir utiliser pleinement ce langage depuis Delphi, Borland commercialise une version spécifique permettant l'accès aux fichiers se trouvant sur le réseau (elle se nomme «Delphi Client/Server»). La version standard de Delphi ne permet que d'accéder à des bases de données SQL se trouvant localement sur la machine<sup>5</sup>.

De plus, Delphi est livré avec un générateur de rapports appelé «Report Smith for Windows» qui permet de présenter et d'imprimer de façon structurée et cohérente des bases de données. Ce programme est très facilement utilisable à partir d'applications créées avec Delphi.

Borland permet aux développeurs Delphi de fournir, sans frais supplémentaires, des versions redistribuables du Borland Database Engine (BDE) et de «Report Smith» afin de les utiliser depuis les applications développées.

## 4.1 Exemple d'application

Comme nous l'avons vu précédemment, la création d'applications utilisant des bases de données a été très simplifiée par les développeurs de Delphi. Afin de

---

5. La version Desktop de Delphi permet de concevoir des applications effectuant des appels SQL sur des bases de données en local. Elle est pour ce faire livrée avec le «Local Interbase Server». Il suffira par la suite de recompiler notre programme avec la version Client/Server de Delphi afin de pouvoir accéder aux bases de données se trouvant sur un réseau.

démontrer la facilité avec laquelle il est possible de créer ce genre d'applications, nous allons étudier brièvement la création d'un petit gestionnaire de fichiers permettant de visualiser, d'ajouter, d'effacer et d'éditer le nom, prénom et adresse d'une personne.

Pour ce faire nous allons utiliser un fichier au format Paradox préalablement créé<sup>6</sup>. Dans un premier temps, nous allons débiter un nouveau projet. Afin de nous simplifier la conception d'une petite application gérant les bases de données, Borland nous propose un Expert qui crée automatiquement l'interface graphique d'un tel programme (cf. figure 4.1). Il nous suffit, pour l'utiliser, de créer un nouveau formulaire. A ce moment là, au lieu de créer une page vide, nous choisissons d'utiliser l'expert. Celui-ci va nous poser quelques questions sur l'application que nous désirons créer.

Quelques clics de souris plus tard, nous obtenons une application prête à être utilisée pour gérer un petit carnet d'adresses (cf. figure 4.2).

Il est bien sûr possible d'ajouter de nombreuses options à notre gestionnaire d'adresses, il nous faudra cependant écrire quelques lignes de code afin d'y parvenir.

---

6. Delphi est livré avec une application capable de créer les fichiers aux formats énumérés ci-dessus. Celle-ci se nomme «Database Desktop»

---

# Chapitre 5

## Les fonctions avancées

### 5.1 Les applications Multimedia

L'un des sujets les plus à la mode actuellement, c'est le «Multimedia». Delphi ne déroge pas à la règle. Il dispose pour ce faire de très nombreuses possibilités pour simplifier la conception d'une application utilisant à la fois son, image et texte.

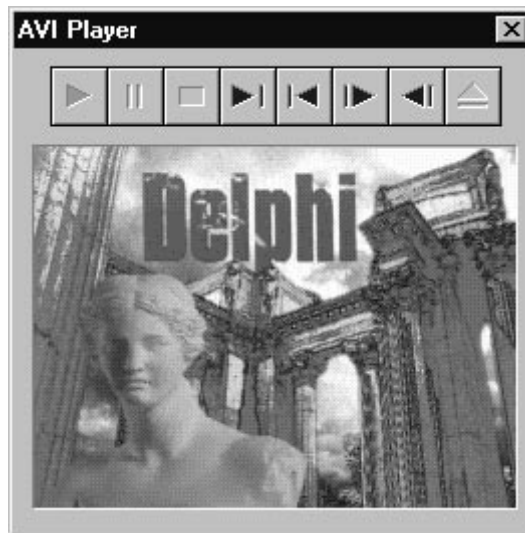
Pour ce faire, le développeur peut accéder à toutes les fonctions offertes par Windows comme les commandes MCI<sup>1</sup> ou Media Control Interface (il s'agit de commandes standardisées permettant la gestion de la carte son et des films vidéo). Delphi simplifie en outre l'accès à ces fonctions en proposant un composant du nom de «TMediaPlayer». Celui-ci se présente sous la forme d'une télécommande et permet d'utiliser toutes les commandes MCI de façon transparente pour le programmeur. Il suffit d'assigner un type de fichier soit à la conception, soit lors de l'exécution de l'application à cette VCL afin que Delphi puisse utiliser les fichiers sons ou images.

Ce composant simplifie à un tel point la conception d'applications Multimedia qu'il est par exemple possible de créer un programme permettant d'afficher un film au format AVI sans avoir à écrire le moindre code source (cf. figure 5.1). Pour ce faire, il nous suffit de déposer deux VCL sur le formulaire. Le premier est un composant de type TMediaPlayer. Il s'agit de la partie qui effectuera la gestion de l'animation.

Il nous faut encore définir une surface d'affichage du film vidéo. Pour ce faire, nous allons utiliser un composant du type TMemo. Il peut paraître quelques peu étrange d'afficher une animation dans une fenêtre servant, comme nous l'avons vu aux chapitres 2 et 3, à l'édition de fichiers textes. Ce choix s'explique cependant assez facilement. Le composant TMemo dispose en effet de nombreuses

---

1. Avant de pouvoir utiliser un périphérique Multimedia, il faut s'assurer que les drivers MCI soit installés sur le Système.

FIG. 5.1 - *Application Multimedia*

fonctions permettant l'affichage de graphismes<sup>2</sup> en plus du texte. Plusieurs autres composants peuvent aussi être utilisés pour l'affichage des animations. Il ne nous reste plus qu'à modifier quelques propriétés des divers objets qui composent notre application pour que celle-ci soit entièrement fonctionnelle.

## 5.2 Gestion des exceptions

Le problème principal que rencontre un programmeur lorsqu'il développe une application, est de découvrir et de corriger les erreurs de son application. Celles-ci se classent en trois catégories:

– **Les erreurs de compilation**

Elles sont facilement détectables car le compilateur nous les signale lors de la compilation. Il s'agit très souvent d'une erreur de syntaxe qui résulte d'une faute de frappe. Leur correction est généralement très aisée.

– **Les erreurs d'exécution**

Ce sont des erreurs assez gênantes. En effet, elles ne sont pas détectées lors de la compilation. Cependant, lors de l'exécution, le programme arrêtera son exécution sans prévenir. Delphi dispose, pour les «supprimer», d'une gestion d'erreurs (cf. ci-dessous).

---

2. Ces propriétés proviennent de l'un des «ancêtres» de la VCL TMemo qui comporte des fonctions graphiques. Comme les descendants héritent de toutes les fonctions de leurs ancêtres, ce composant en dispose aussi. (cf chapitre 3, à la page 25)

– **Les erreurs logiques**

Il s'agit des erreurs les plus difficiles à corriger, car elles résultent très souvent d'une faute de réflexion et de conception du programme. Leur correction consiste à étudier la logique des algorithmes utilisés.

Tous les composants de Delphi disposent de fonctions permettant le traitement des exceptions (exception handling). Normalement, chaque exception, c'est-à-dire chaque faute survenue lors de l'exécution, a pour effet d'arrêter immédiatement le programme en affichant un message barbare. En s'aidant de cette gestion des exception, le programme peut réagir de façon adéquate lorsque ces erreurs se produisent.

Pour ce faire, le programmeur doit «protéger» les parties du code sensibles en utilisant la syntaxe:

```
TRY
  {Code a protéger}
EXCEPT
  ON Exception DO {Réaction en cas d'erreur}
END;
```

Lors de l'exécution du programme, si une erreur survient dans la partie protégée par le le gestionnaire d'exceptions, celui-ci exécutera automatiquement la réaction appropriée d'erreur. Ici, la réaction sera effectuée pour toutes les erreurs rencontrées. Il est cependant possible d'agir plus précisément suivant les fautes. Par exemple, lors d'une opération arithmétique, il peut arriver que le programme effectue une division par zéro. Afin de protéger notre application d'un tel problème, il nous faudrait écrire:

```
TRY
  i := 42 / 0;
EXCEPT
  ON EDivByZero DO i := 0;
END;
```

Il existe de très nombreuses exceptions gérées par Delphi. Cela va de `EGPFault` à `EOutOfMemory`, en passant par `EInvalidPointer`.

Il existe une autre protection possible:

```
TRY
  {Code a protéger}
FINALLY
  {action a effectuer de tout façon}
END;
```

La différence entre la première méthode et celle-ci c'est que ici, le code suivant le mot clé `FINALLY` sera exécuté de toute façon à la fin de la partie protégée. Dans le cas d'une exception, le programme exécutera directement cette instruction. Cette méthode est très utile lors de l'utilisation de pointeurs afin de pouvoir libérer la mémoire de toute façon, ...

## 5.3 Les fichiers DLL

Lors du développement de plusieurs applications utilisant des boîtes de dialogues communes, il peut-être préférable de regrouper ces fenêtres graphiques dans une DLL<sup>3</sup> (ou Dynamic Link Library). Il s'agit d'une application (fichier unique) pouvant être exécutée par d'autres programmes afin d'accéder à ces fonctions. Une telle librairie peut non seulement être utilisée avec des applications écrites en Delphi, mais aussi en C++, Visual Basic, dBase et avec pratiquement tous les autres «langages» de programmation sous Windows. De plus, Delphi permet facilement la création de tel DLL. Leurs développement fonctionne sur le même principe que les exécutables (.EXE) Toute la création de la boîte de dialogues s'effectue de manière identique. Le fichier projet devra par contre subir quelques modifications. Il faut le transformer en un fichier ressemblant, pour le cas d'une boîte de dialogue, à:

```
LIBRARY Proj_dll;
USES
  unit1 IN 'unit1.pas' { module contenant le form. MyDialog }
EXPORTS
  MyDialog;           { Procedure affichant la boîte de dialogue }

{$R *.RES}
BEGIN
END.
```

Par la suite, lorsque nous voudrons faire appel à «MyDialog», il nous faudra ajouter:

```
PROCEDURE MyDialog(...); FAR;
EXTERNAL 'PROJ_DLL';
```

dans notre projet afin de pouvoir afficher celle-ci.

Il est aussi possible de créer des DLL n'utilisant pas de formulaires. Elles peuvent en effet contenir des algorithmes utilisés très souvent par de nombreuses applications, ...

---

3. Extension de fichiers .DLL

# Bibliographie

- [1] Borland International, 100 Borland Way, Scotts Valley, CA 95066-3249. *Borland Delphi for Windows Component Writer's Guide*, 1995. Part HDB1110WW21772 - BOR 8142.
- [2] Borland International, 100 Borland Way, Scotts Valley, CA 95066-3249. *Borland Delphi for Windows User's Guide*, 1995. Part HDB1110WW21770 - BOR 8141.
- [3] Borland International, 100 Borland Way, Scotts Valley, CA 95066-3249. *Borland Local InterBase Server for Windows User's Guide*, 1995. Part HDB1110WW21776 - BOR 8123.
- [4] Borland International, 100 Borland Way, Scotts Valley, CA 95066-3249. *Borland ReportSmith for Windows: Creating Reports*, 1995. Part HDB1110WW21773 - BOR 8125.
- [5] Charles Calvert. *Delphi Unleashed*. SAMS Publishing, 201 West 103rd Street, Indianapolis, IN 46290, 1995. ISBN: 0-672-30499-6.
- [6] Arthur Burda et Günther Fürber. *Le Grand Livre de Borland Delphi*. Micro Application, 20-22 rue des Petits-Hôtels 75010 PARIS, 1995. ISBN: 2-7429-0460-3.
- [7] Peter Grogono. *La programmation en Pascal*. InterEditions, Addison-Wesley Europe, 1992. ISBN 2-7296-0128-7.
- [8] Jon Matcho and David Faulkner. *Special Edition Using Delphi*. QUE Corporation, 201 W. 103rd Street, Indianapolis, IN 46290, 1995. ISBN: 1-56529-832-3.
- [9] Xavier Pacheco & Steve Teixeira. *Delphi Developer's Guide*. SAMS Publishing, 201 West 103rd Street, Indianapolis, IN 46290, 1995. ISBN: 0-672-30704-9.
- [10] Alessandro Vernet. *Le «petit» guide FidoNet*. 1995. Disponible sur Internet à l'adresse <http://diwww.epfl.ch/~avernet> (formats DVI et Postscript).

# Index

- Barre de composants ..... 15
- Bases de données ..... 31
  - Application ..... 33
  - dBase ..... 31
  - Expert ..... 31
  - ODBC ..... 32
  - Paradox ..... 31
  - SQL ..... 33
- BDE ..... 31
- Code Source ..... 15, 27
  - AboutBox ..... 30
  - Couper-Coller ..... 29
  - Ouvrir Fichier ..... 29
  - Quitter ..... 30
- Configuration requise ..... 7
- Copyright ..... 1
- Delphi ..... 5
  - Compilateur ..... 6
  - Conception ..... 6
  - Possibilités ..... 5
- Delphi-32 ..... 9
- Directives de compilation ..... 28
- DLL ..... 38
- Editeur de texte ..... 18
- Événements ..... 22
- Fenêtre principale ..... 14
- Fichiers ..... 13
- FidoNet ..... 10
- Formulaire ..... 14, 15
- Gestion des exceptions ..... 36
- Gestionnaire de projets ..... 17
- IDE ..... 14
- Internet ..... 11
- L'inspecteur d'objets ..... 15
- MCI ..... 35
- MDI ..... 19
- Module ..... 13
- Multimedia ..... 35
- ObjectPascal ..... 26
- ObjectPascal en action ..... 27
- Objet TTextForm ..... 28
- OnClick ..... 23
- Options de Debugging ..... 17
- Pascal ..... 25
- Projet ..... 13
- SDI ..... 19
- Show ..... 30
- ShowModal ..... 30
- VCL ..... 18
  - TButton ..... 20
  - TFontDialog ..... 21
  - TImage ..... 20
  - TMediaPlayer ..... 35
  - TMemo ..... 20
  - TMenu ..... 19
  - TOpenDialog ..... 21
  - TPrintDialog ..... 21
  - TPrinterSetupDialog ..... 21
  - TSaveDialog ..... 21
- Visual Basic ..... 4, 7