Interfacer DyalogAPL/W et Borland Delphi

Eric Lescasse

Note préliminaire

Cet article est extrait d'un texte anglais que j'ai déjà écrit, et faute de temps, je n'ai pas pu traduire les quelques exemples qui parfois sont en anglais. Je prie les lecteurs des `Nouvelles d'APL` de m'en excuser.

Introduction

Dyalog APL/W est un produit très complet, permettant de développer facilement l'interface Windows de son application, tout aussi bien que d'écrire les programmes de calculs et de traitements de données indispensables au développement d'une application Windows complète.

Pourtant, il y a de nombreuses raisons pour lesquelles il peut être intéressant d'interfacer Dyalog APL/W et Delphi.

Ces raisons sont :

· Delphi peut vous aider à développer mieux et plus vite l'interface Windows de votre application grâce à son très puissant éditeur de ressources;

· Delphi vous permet de développer aisément des DLLs dans lesquelles vous pouvez stocker des fonctions compilées rapides, que vous pouvez ensuite exécuter depuis Dyalog APL/W;

· Delphi vous permet de créer facilement des DLLs dans lesquelles vous pouvez installer des fenêtres et même des applications Delphi complètes; vous pouvez ensuite appeler ces fenêtres depuis Dyalog APL/W, presque comme s'il s'agissait de fenêtres Dyalog APL/W;

· finalement, Delphi vous permet parfois, grâce à sa très riche bibliothèque de composants, de développer des parties de votre application qui seraient difficiles à réaliser en Dyalog APL/W.

Cet article d'introduction va montrer comment :

- créer une DLL en Delphi et installer une fenêtre Delphi dans une DLL,
- appeler cette DLL depuis APL,
- passer des arguments d'APL à Delphi,
- retourner des résultats d'une DLL Delphi à APL,
- utiliser l'éditeur de ressources Delphi pour créer une boîte de dialogue et auto-générer un programme Dyalog APL/W qui utilise cette boîte de dialogue.

Les techniques présentées ici, sont certainement utilisables également avec APL*PLUS III v1.2.

De quoi avez-vous besoin ?

Pour interfacer Dyalog APL/W et Delphi, vous avez besoin de:

la version 16-bits de Dyalog APL/W et Delphi ou Dyalog APL/W 95 et la version 32-bits de Delphi (à paraître)

Borland RadPack

Il est aussi recommandé de disposer de littérature supplémentaire concernant Delphi. Voici quelques bons livres:

" **Mastering Delphi** " by Marco Cantù, 1995
SYBEX
ISBN 0-7821-1739-2
1500 pages + CDROM

" **Delphi Unleashed** " by Charles Calvert, 1995
SAMS Publishing
ISBN 0-672-30499-6
950 pages + CDROM

" **Delphi, a Developer's Guide** " by Bill Todd & Vince Kellen, 1995
M&T books
ISBN 1-55851-455-4
820 pages + CDROM

" **Delphi Developer's Guide** " by Xavier Pacheco & Steve Texeira, 1995
Borland Press
ISBN 0-672-30704-9
910 pages + CDROM

“ **Delphi Programming for Dummies** ” by Neil J. Rubenking, 1995
IDG Books
ISBN 1-56884-200-7
376 pages

“ **Delphi pour Windows** ” by Philippe Spoljar, 1995
SYBEX
ISBN 2-7361-1537-6
671 pages + 3.5 inch disk

“ **Le Grand Livre de Borland Delphi** ” by Arthur Burda & Gunther Farber, 1995
Micro Application
ISBN 2-7429-0460-3
490 pages + CDROM

Le développement d’un projet avec Delphi

Mini didacticiel Delphi
Lorsque vous chargez Delphi, les fenêtres suivantes apparaissent :

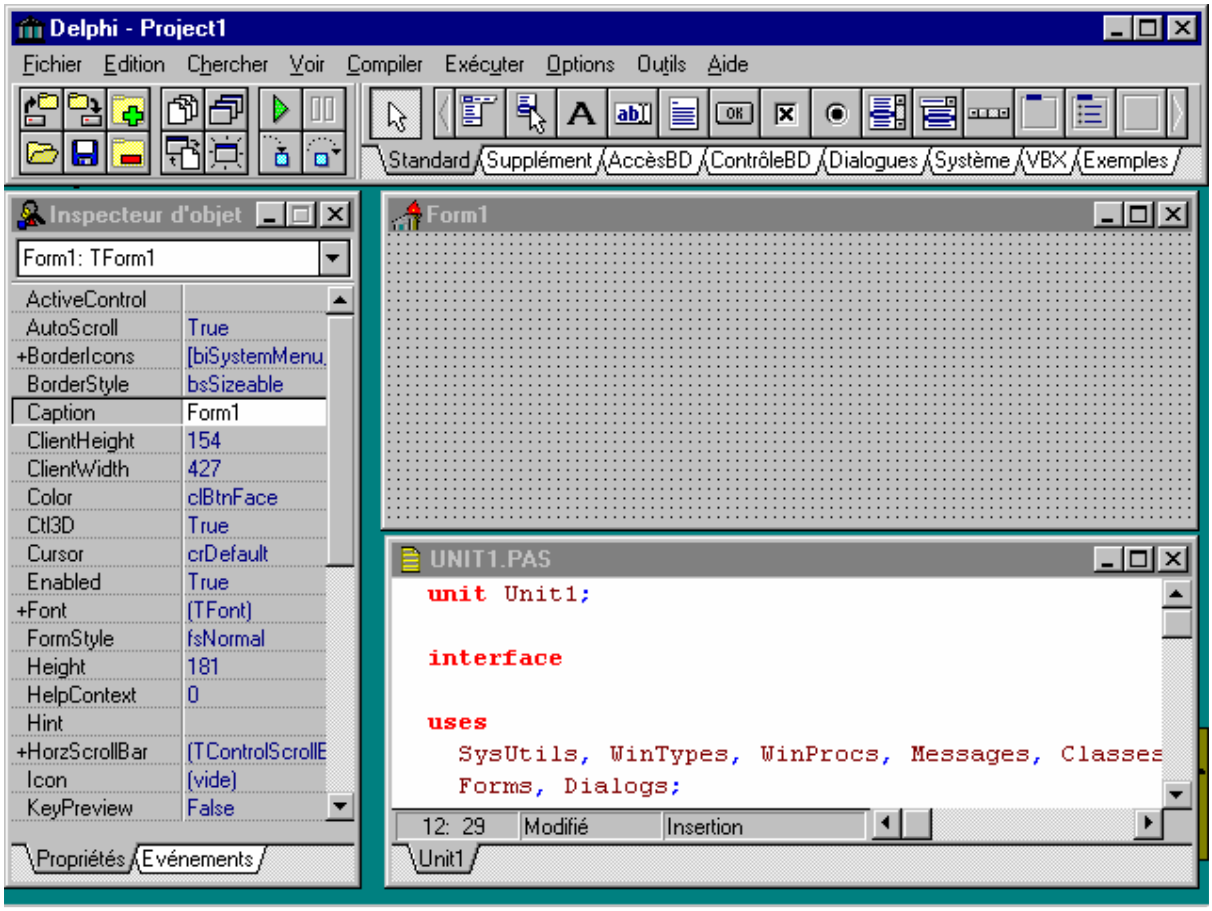

*Figure SEQ Figure * ARABIC 1 - Environnement de développement de Delphi*

La fenêtre du haut est la fenêtre principale Delphi, qui contient les menus, les barres d'icônes et les palettes de composants.

La fenêtre de gauche est l'Inspecteur d'Objets: elle vous permet de visualiser, définir ou modifier les propriétés et les événements relatifs aux objets.

La fenêtre intitulée **Form1** est la fenêtre de l'application que vous allez développer, celle dans laquelle vous allez installer des composants pris dans la palette de composants de la fenêtre principale.

La fenêtre du bas est l'éditeur de code qui contient le code Pascal des programmes que vous écrivez pour votre application.

Pour changer la propriété d'un objet, par exemple le titre de notre fenêtre, cliquez dans la partie droite de la zone **Caption** de l'Inspecteur d'Objets, et remplacez le texte **Form1** par :

My first Delphi application

Le nouveau titre apparaît dans la fenêtre au fur et à mesure que vous le tapez au clavier.

Le code Pascal Objet que vous écrivez est principalement du code destiné à gérer des événements.

Maintenant, ajoutez simplement un bouton dans la fenêtre, en double-cliquant sur l'icône représentant un bouton dans la palette de composants. Un bouton appelé **Button1** apparaît au centre de la fenêtre. Double-cliquez sur le bouton **Button1** dans votre fenêtre et Delphi génère le code nécessaire à la gestion de l'événement clic sur le bouton **Button1** et active l'éditeur de code.

Il vous suffit d'entrer l'instruction ou les instructions souhaitées pour gérer le clic sur le bouton **Button1**, ici :

Form1.Close;

ce qui fermera notre fenêtre lorsque l'utilisateur cliquera sur le bouton **Button1** (n'oubliez pas le point-virgule qui marque la fin d'une instruction Pascal).

Voici l'écran que vous devez avoir obtenu :

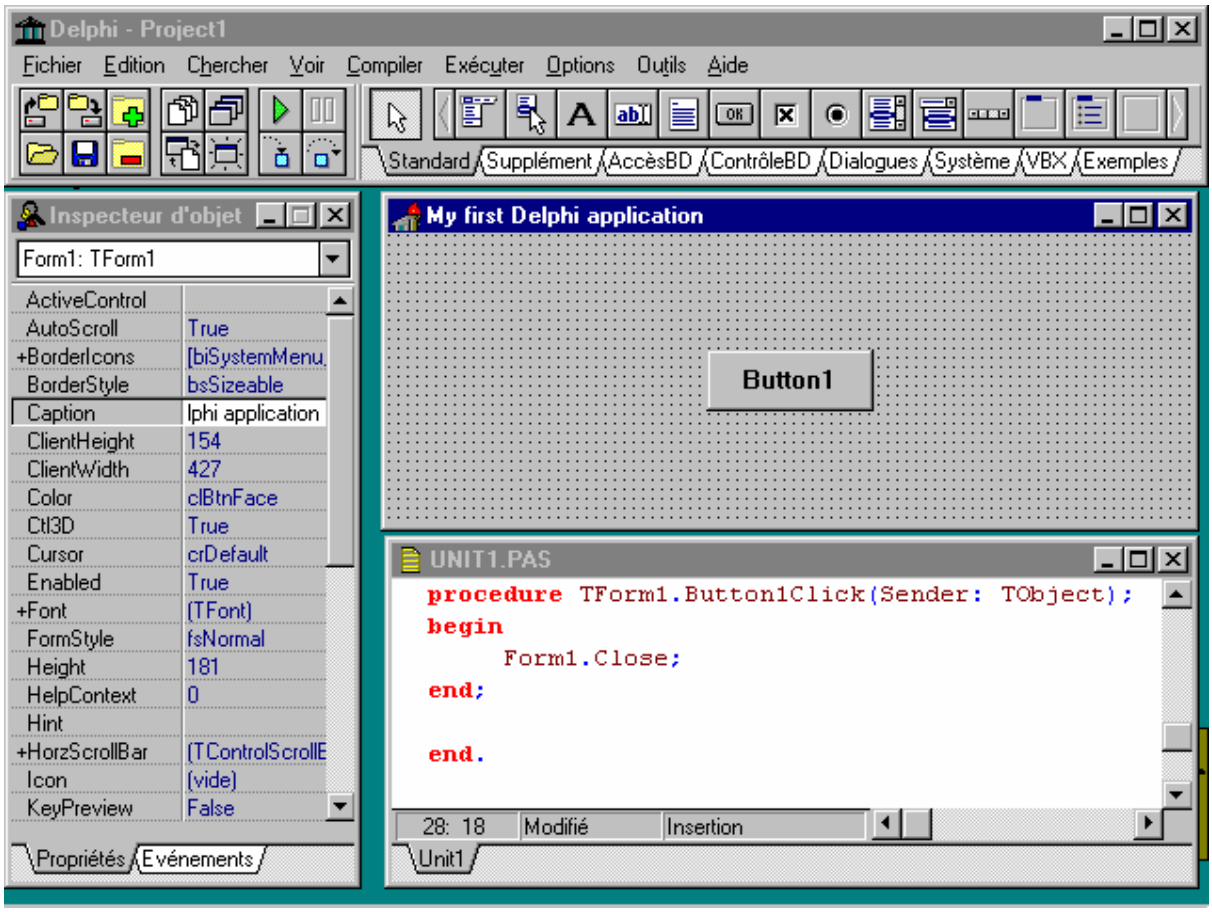

*Figure SEQ Figure * ARABIC 2 - Fenêtre avec titre et bouton*

Pour démarrer et tester votre petite application Delphi, cliquez sur l'icône en forme de triangle pointant vers la droite.

Votre fenêtre apparaît: cliquez sur le bouton **Button1**. Elle se termine.

Sauvegarde d'un projet Delphi
Avant de commencer à développer des DLL, il nous faut savoir comment sauvegarder un projet Delphi.

Choisissez simplement **Fichier/Enregistrer projet sous...** et Delphi vous demande le nom d'un fichier **.PAS**:

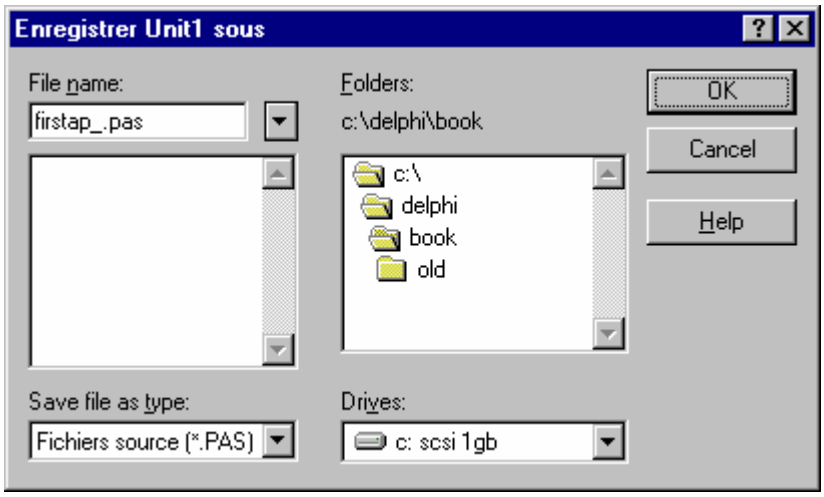

*Figure SEQ Figure * ARABIC 3 - Boîte de sauvegarde du code source d'une application Delphi*

Ce fichier correspond à votre éditeur de code Pascal. Appelons le **FIRSTAP_.PAS**.

Delphi vous demande alors d'entrer le nom de votre fichier projet dont l'extension sera **.DPR**.

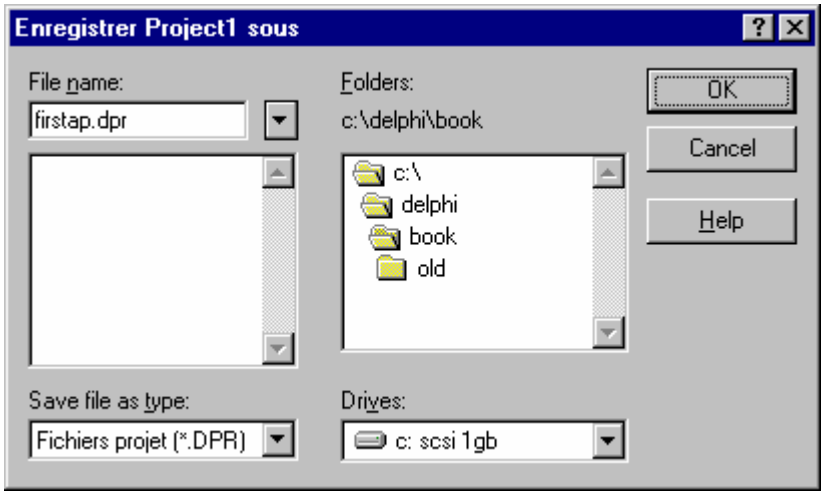

*Figure SEQ Figure * ARABIC 4 - Boîte de sauvegarde du projet Delphi*

Ce fichier correspond au code source de votre projet. Appelons le **FIRSTAP.DPR**.

Faîtes bien attention à toujours nommer ces deux fichiers avec des noms différents (ici, **FIRSTAP_** et **FIRSTAP**). Delphi impose que ces noms soient différents, sous peine de ne pouvoir faire fonctionner votre application correctement.

Le code source de votre projet

Une fois que vous avez sauvegardé votre projet, vous pouvez ouvrir la fenêtre d'édition du code source de ce projet. Choisissez **Voir/Source du projet** dans les menus. Le code source de votre projet est chargé dans l'éditeur et ressemble à ceci :

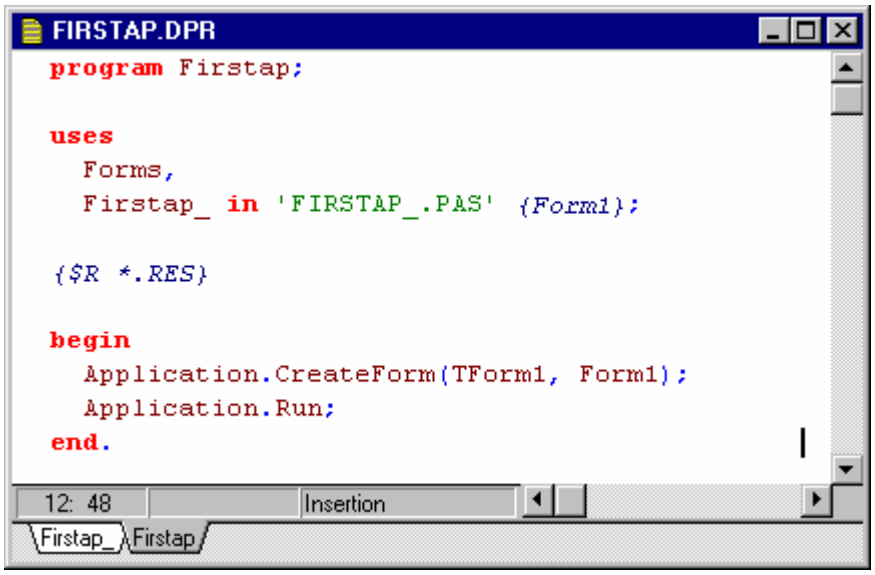

*Figure SEQ Figure * ARABIC 5 - Code source d'un projet Delphi*

Notez que la fenêtre de l'éditeur contient maintenant deux onglets, l'un pour le code source de votre fenêtre (**Firstap_**) et l'autre pour le code source de votre projet (**Firstap**).

Nous allons devoir modifier ces deux fichiers sources pour faire de notre projet Delphi une DLL, mais ceci est très simple.

Transformation d'un projet Delphi en DLL

Et voici ce qui est impressionnant !

Nous allons maintenant transformer notre projet Delphi en une DLL et nous allons l'appeler depuis Dyalog APL/W, en nous servant de cette DLL.

Vous devez procéder exactement comme suit :

dans l'onglet du code source du projet FIRSTAP

1. ouvrez le code source du projet dans l'éditeur en cliquant sur l'onglet **Firstap**
2. remplacez le mot **Program** par le mot **Library**
3. supprimez **Forms,** de la clause **Uses**
4. supprimez le corps du bloc **begin...end**
5. ajoutez une clause **exports**, au dessus du mot **begin**, comme suit:
   exports
   FirstApp index 1;

Votre code source de projet doit maintenant ressembler à ceci (noter que j'ai conservé les anciennes instructions sous forme de commentaires)[1]:

[1] Tout ce qui est inclus entre accolades est du commentaire en Delphi.

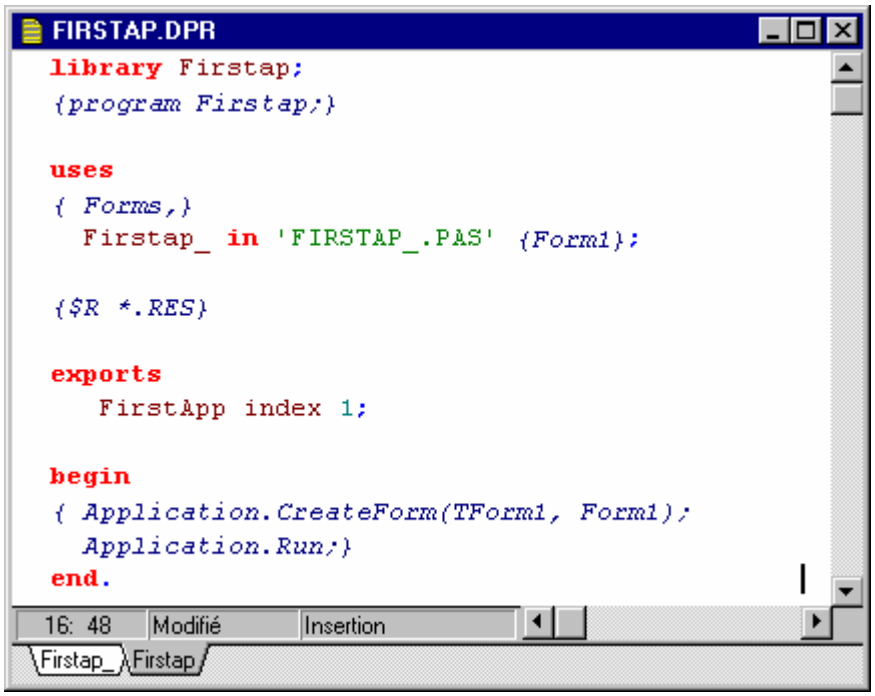

*Figure SEQ Figure * ARABIC 6 - Code source d'un projet de DLL Delphi*

dans la fenêtre de code source de votre application Firstap_

· ouvrez le code source le l'application dans l'éditeur en cliquant sur l'onglet **Firstap_**
· ajoutez l'instruction suivante juste au-dessus de la clause **implementation**

procedure FirstApp; export;

· ajoutez la procedure suivante juste au-dessus du dernier mot-clé **end:**

```
{interface function}
procedure FirstApp;
begin
    Form1 :=TForm1.Create(Application);
    try
      Form1.ShowModal;
    finally
      Form1.Free;
    end;
end;
```

Votre code source devrait maintenant ressembler à ceci :

```
unit Firstap_;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private-déclarations }
  public
    { Public-déclarations }
  end;

var
  Form1: TForm1;

procedure FirstApp; export;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
    Form1.Close;
end;

{interface function}
procedure FirstApp;
begin
    Form1 :=TForm1.Create(Application);
    try
      Form1.ShowModal;
    finally
      Form1.Free;
    end;
end;

end.
```

Commentaires concernant la procédure FirstApp
Etant donné que la seule chose que l'on peut faire avec une DLL est d'appeler une fonction ou procédure inclue dans cette DLL, il nous a fallu ajouter une fonction ou procédure (ici une procédure) dans notre code source. Nous l'avons nommée **FirstApp**.

Le rôle de la procédure **FirstApp** est de créer dynamiquement une instance de notre fenêtre d'application et la démarrer de façon modale avec l'instruction **ShowModal**.

En Delphi, lorsque vous créez un objet pendant l'exécution d'un programme avec la méthode **Create**, vous allouez de la mémoire pour cet objet: cela signifie que vous DEVEZ libérer cette mémoire lorsque l'objet n'est plus utilisé, c.a.d. quand l'utilisateur le ferme.

Ceci est facilement réalisé en invoquant la méthode **Free** sur l'objet.

Mais, de façon à être sûr de libérer la mémoire, même dans le cas où une erreur se produirait pendant l'exécution de notre fenêtre, nous incluons l'appel de cette fenêtre dans un bloc de programme de type **try...finally...end**.

Dans le cas où une erreur se produit dans le bloc **try**, Delphi bascule automatiquement l'exécution du programme au bloc **finally**, libérant ainsi la mémoire correspondant à notre fenêtre.

La procédure **FirstApp** présentée ci-dessus est le modèle général (et minimal) à appliquer lorsque vous désirez transformer une application Delphi en DLL, puis l'appeler depuis un autre langage.

Compilation de notre application dans une DLL
Lorsque vous avez scrupuleusement suivi les instructions énoncées ci-avant, il n'y a plus rien de particulier à faire pour être capable de compiler votre application sous forme d'une DLL.

Choisissez simplement **Compiler/Compiler** ou pressez **Ctrl+F9**.

Le compilateur Delphi constate que le code source de votre projet commence par le mot-clé **library** et donc comprend que vous souhaitez créer une DLL.

En moins de deux secondes, vous disposez de votre première DLL pour utilisation depuis Dyalog APL/W. Elle s'appelle **FIRSTAP.DLL** et se trouve dans le même répertoire que votre projet Delphi.

Appel d'une application Delphi depuis Dyalog APL/W

Nous sommes maintenant curieux et très impatient d'appeler notre application Delphi **FirstApp** depuis Dyalog APL/W.

Auriez-vous pensé que c'est aussi simple que de faire ce qui suit :

```
∩NA ßC:\DELPHI\BOOK\FIRSTAP.DLL.P16|FirstAppß
 FirstApp
```

où la première instruction charge la DLL en mémoire et crée la fonction associée **FirstApp** dans votre espace de travail.

La seconde instruction exécute simplement la fonction associée **FirstApp**.

Celle-ci lance votre application Delphi, depuis Dyalog APL/W.

Voici à quoi ressemble votre écran :

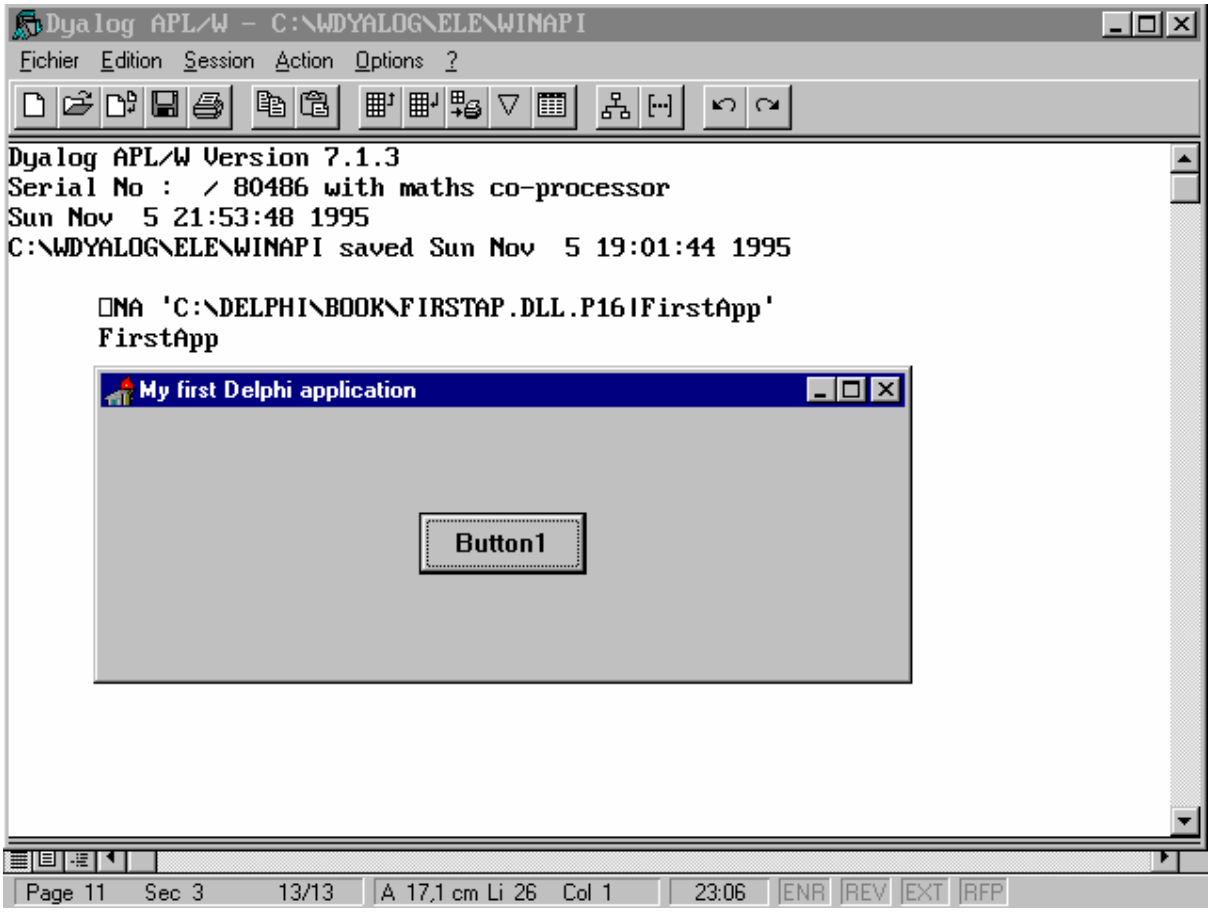

*Figure SEQ Figure * ARABIC 7 - Appel d'une boîte de dialogue Delphi depuis Dyalog APL/W*

Si vous aviez compilé votre projet Delphi avec la version 32-bits de Delphi (à paraître en Janvier 96) et désiriez utiliser la DLL résultante depuis Dyalog APL/W, il aurait fallu associer la fonction **FirstApp** avec l'instruction suivante :

```
⎕NA C:\DELPHI95\BOOK\FIRSTAP.DLL.P32|FirstApp
```

Passage des arguments et résultats
Mais supposons que nous désirions que notre fenêtre soit une vrai boîte de dialogue! Je veux dire une boîte de dialogue qui permette de saisir des informations.

Nous avons besoin de mettre en place un mécanisme pour échanger des données entre Dyalog APL/W et Delphi.

Les choses deviennent un peu plus compliquées.

Nous devons passer des arguments à Delphi de façon à lui donner des zones de mémoire dans lesquelles il pourra installer le résultat de la saisie pour le retourner à APL.

Comme nous devons pouvoir gérer plusieurs types de données (caractères, entiers, flottants) et diverses structures (vecteurs, matrices), nous devons apprendre plusieurs choses:

- comment passer des arguments à Delphi
- comment récupérer des résultats de Delphi
- les différents types de données de Delphi
- la syntaxe et les arguments de □NA

Si vous voulez maîtriser les relations entre APL et Delphi, il va vous falloir investir un peu de temps à étudier chacun de types de données de ces deux produits.

Mais nous allons vous donner quelques exemples pour mettre le pied à l'étrier.

Commençons par changer notre fenêtre, en y ajoutant quelques autres composants de Delphi, de façon à ce qu'elle ressemble à la suivante :

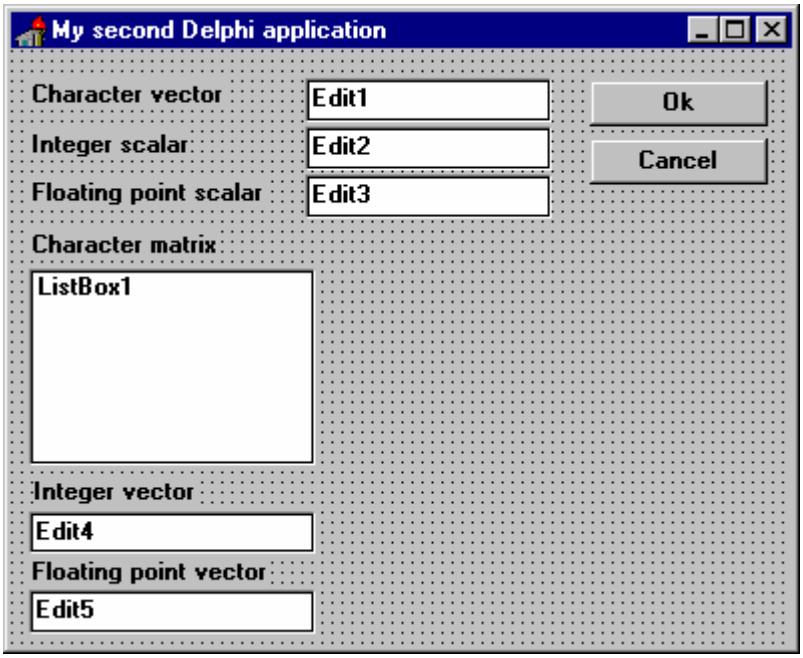

*Figure SEQ Figure * ARABIC 8 - Boîte de dialogue Delphi en cours de développement*

Nous avons volontairement laissé visible le nom de chacun des composants (**Edit1**, **Edit2**, ...) de façon à ce que vous puissiez les identifier plus facilement.

Pour commencer, nous allons nous concentrer à essayer de retourner vers Dyalog APL/W, le texte entré par l'utilisateur dans le composant **Edit1**.

Pour ce faire, nous devons modifier notre application **FirstApp** en une procédure que nous rebaptisons **SecondApp** et que nous sauvons dans un nouveau projet **SECNDAP.DPR**.

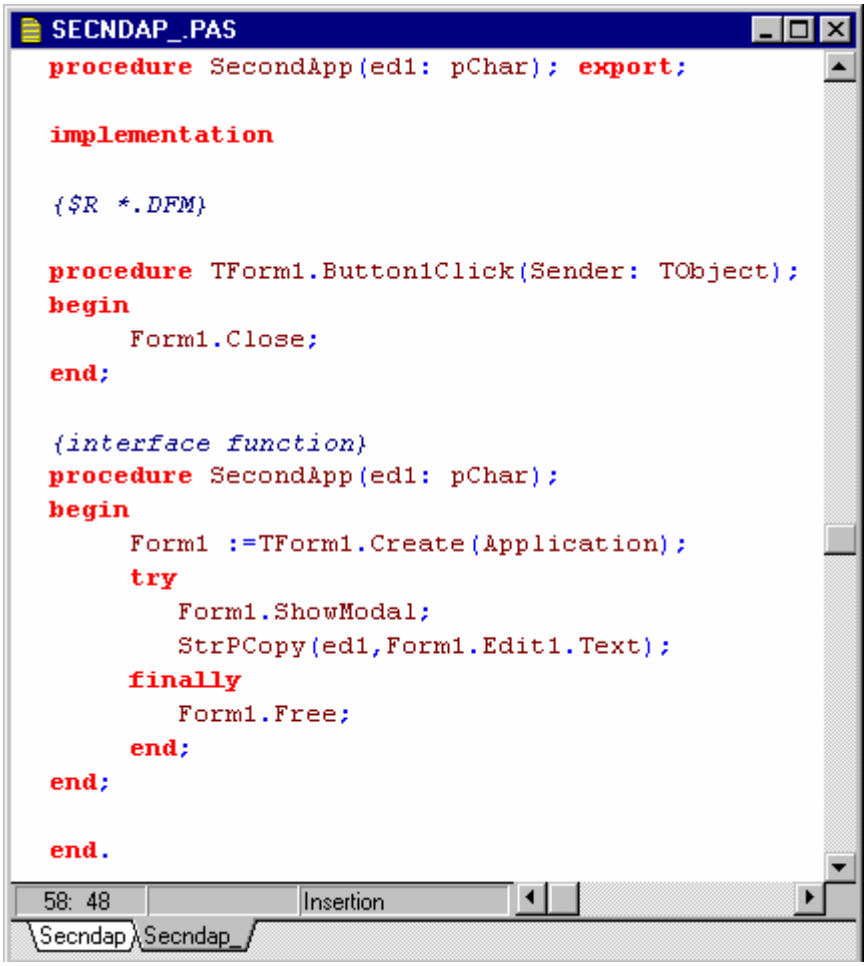

*Figure SEQ Figure * ARABIC 9 - Source Pascal montrant la fonction d'appel SecondApp d'une boîte de dialogue*

Nous avons ajouté deux choses:

- un argument **ed1** de type **pChar**
- une instruction pour affecter à **ed1** le contenu du champ **Edit1**

lorsque l'utilisateur ferme la fenêtre.

N'oubliez pas d'effectuer le même changement dans la déclaration du prototype de la fonction **SecondApp** (au dessus de la clause **implementation**) et dans l'entête de la fonction elle-même. Ces deux déclarations doivent être absolument identiques (excepté pour le mot-clé **export**; qui ne doit apparaître que dans le prototype).

Le langage Pascal Objet reconnaît deux type de données caractère: les chaîne de type **pChar** et les chaînes de type **string**.

Les chaînes de type **pChar** correspondent aux chaînes à 0 terminal bien connues des programmeurs C++ et des programmeurs Windows: ce sont celles-ci qui sont universellement utilisées dans l'API Windows. Nous DEVONS utiliser ce type de données pour nos besoins, ici.

Les chaînes de caractère de type string représentent un type de données interne au Pascal Objet et sont limitées à une longueur de 255 caractères : le premier caractère d'une telle chaîne contient la longueur de la chaîne.

Bon. N'oubliez pas de changer aussi le mot FirstApp en SecondApp dans la clause exports du code source de votre projet, puis de choisir Fichier/Enregistrer projet sous... pour maintenant sauvegarder votre projet sous les noms **SECNDAP_.PAS** pour le code source de la fenêtre et **SECNDAP.DPR** pour le code source du projet.

Nous sommes désormais prêt à appeler notre second application depuis Dyalog APL/W.

Dans ce but, écrivons la petite fonction de couverture suivante :

```
  S R¢SecondApp;SecondApp
[1]    ∩NAßC:\DELPHI\BOOK\SECNDAP.DLL.P16|SecondApp =0Tß
[2]    R¢SecondApp∘16Vß ß
     S
```

Essayons-la. Tapez SecondApp : votre fenêtre Delphi apparaît à l'écran ! Entrez **Dyalog APL/W** dans le champ correspondant à **Character vector.** Voici ce que vous devriez avoir :

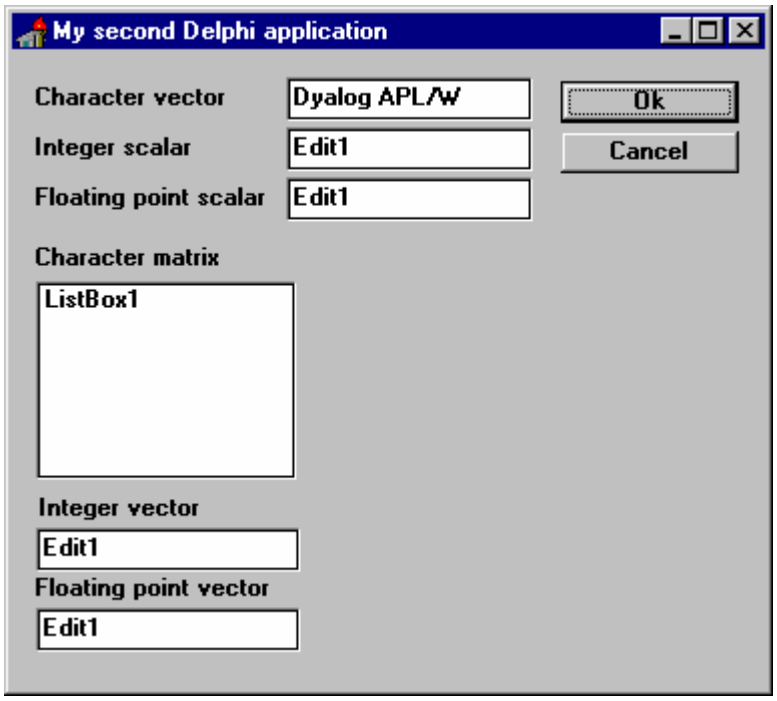

*Figure SEQ Figure * ARABIC 10 - Application Delphi appelée depuis Dyalog APL/W*

Cliquez maintenant sur le bouton OK.

Vous devriez voir ceci dans votre session APL.

```
 SecondApp
Dyalog APL/W
```

Nous avons atteint notre objectif.

Nous avons réussi à passer des données depuis APL à une fenêtre Delphi (sous la forme d'un « buffer » de 16 caractères de large: voir la fonction SecondApp ci-dessus) et appris à Delphi à nous retourner l'information saisie par l'utilisateur dans l'un des champs de la fenêtre (par l'intermédiaire de ce « buffer »).

Mise en garde
Sachez cependant, qu'il est en fait facile (en fait TRES facile) d'avoir à « rebooter » son micro-ordinateur lorsque l'on effectue ce genre de travail.

Vous devez faire très attention aux arguments et aux résultats de votre DLL.

Les déclarations des arguments en Delphi doivent EXACTEMENT correspondre aux types déclarés dans l'instruction □NA qui correspond à votre application.

Lorsque vous écrivez votre procédure Delphi, vous devez aussi faire très attention à ne pas commettre une erreur comme d'utiliser le type **string** au lieu du type de données **pChar** pour les chaînes de caractères, par exemple.

Si vous changiez **ed1: pChar** en **ed1: String** partout dans le code source de l'unité **SECNDAP_.PAS**, recompiliez votre projet et appeliez la DLL depuis Dyalog APL/W, vous seriez désappointé de recevoir une **GPF** (**Faute de Protection Générale**) et d'avoir peut être à « rebooter » votre micro-ordinateur.

GPF signifie « General Protection Fault » et, en pratique, signifie que votre application a utilisé une zone de mémoire qu'elle n'aurait pas dû. Cela signifie aussi qu'elle va être immédiatement fermée et que, le plus souvent vous devrez redémarrer Windows, voire « rebooter » votre machine, en particulier si vous ne travaillez pas sous Windows 95.

Après avoir eu une GPF sous Windows 95 et en supposant que vous ayez eu seulement à fermer Dyalog APL/W, annulez la modification précédente en remplaçant dans **SECNDAP_.PAS ed1: String** par **ed1: pChar**, puis essayez de recompiler votre DLL.

Malheureusement une boîte de dialogue apparaît vous indiquant que la compilation est impossible, cette DLL (**SECNDAP.DLL**) étant en cours d'utilisation. Il faut commencer par la décharger de la mémoire.

Si vous ne disposez pas d'un outil pour effectuer ce déchargement, votre seule chance de décharger cette DLL de la mémoire est de rebooter votre ordinateur.

C'est pourquoi je vous recommande d'acquérir un outil de déchargement de DLL.

Celui que j'utilise s'appelle **DLL Unloader**: il a été écrit par Dan Ruder et est sous Copyright 1992 Microsoft.

Vous pouvez le télécharger librement depuis CompuServe ou Internet.

Quand vous démarrez DLL Unloader, la fenêtre suivante apparaît à l'écran: un ascenseur vous aide à sélectionner la DLL coupable dans la liste de toutes les DLL chargées en mémoire. Lorsqu'elle est sélectionnée, cliquez sur le bouton **Unload**.

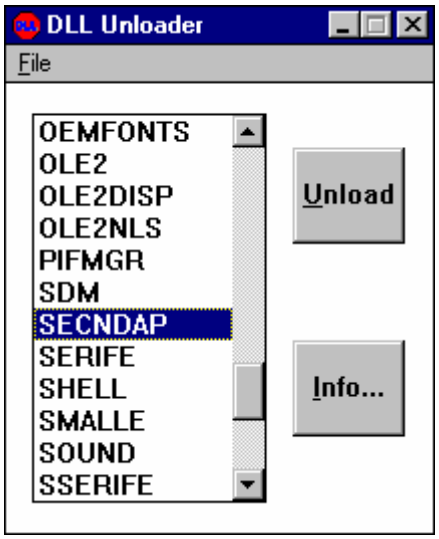

*Figure SEQ Figure * ARABIC 11 - DLL Unloader*

Mais ne vous trompez pas de DLL, ou votre système pourrait devenir immédiatement instable!

Utilisation d'autres types de données

Il serait trop long de montrer ici comment échanger des données d'autres types, comme des entiers ou flottants, directement entre Dyalog APL/W de Delphi. Nous vous laissons explorer ces possibilités, mais restons à votre disposition, si vous aviez besoin d'aide en ce domaine.

Une application Delphi plus complète

A titre d'exemple, voici une application Delphi multi-média complète, installée dans une DLL et utilisable depuis APL :

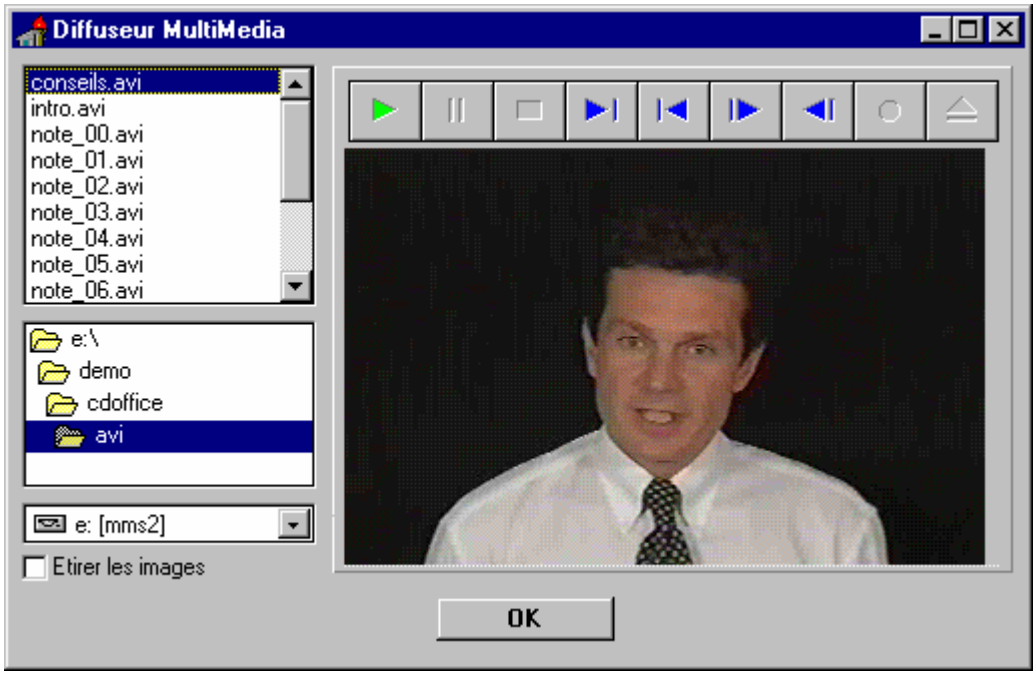

*Figure SEQ Figure * ARABIC 12 - Film vidéo en cours de diffusion dans une application multimedia*

Cette application présente une fenêtre contenant des " list box " et " combo box " permettant de choisir son lecteur et son répertoire de travail, puis de sélectionner un fichier de type image **(.BMP)**, son (**.WAV** et **.MID**) et vidéo (**.AVI**).

La partie droite de la nt la zone de diffusion. Il suffit de double-cliquer sur un nom de fichier pour qu'il soit automatiquement diffusé.

Cette application, installée dans une DLL Delphi avec la même technique que celle décrite dans la première partie de cet article, est appelable depuis APL en exécutant la fonction suivante:

```
   S R¢CallMedia;MediaViewer
[1]     ∩NAβC:\DELPHI\AFAPL\MEDIA.DLL.P16|MediaViewer >0Tβ
[2]     R¢MediaViewer 255
      S
```

Une autre utilisation intéressante de Delphi : un générateur de programmes APL

Voici, à titre d'exemple, une autre utilisation possible de Delphi.

La société Uniware a développé un outil permettant de traduire automatiquement une fenêtre ou une boîte de dialogue développée sous Delphi en un programme Dyalog APL/W. Le gain de temps obtenu en phase de développement d'application Windows est considérable et la qualité des boîtes de dialogue produite est remarquable, grâce au puissant éditeur de ressources de Delphi. Ce générateur de programme s'appelle ParseDFM. Voici un exemple de boîte de dialogue développée sous Delphi :

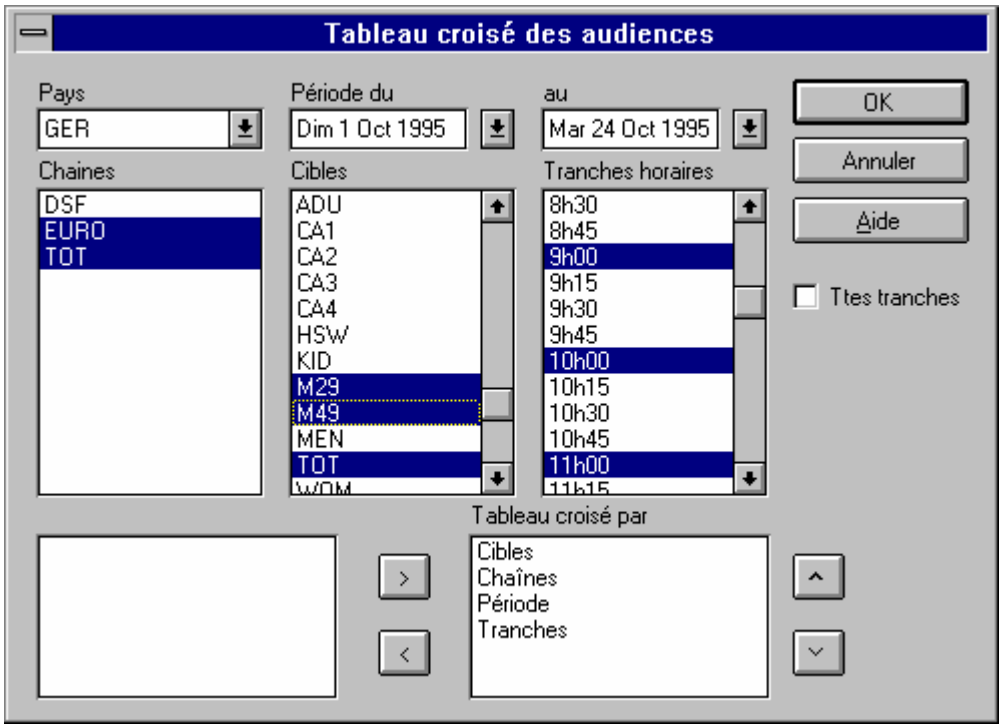

*Figure SEQ Figure * ARABIC 13 - Exemple de boîte de dialogue développée sous Delphi et traduite en APL*

et voici sur les pages suivantes, à titre d'exemple, le code du programme APL auto-généré par ParseDFM.

```
     S R¢{⍞←≥Δ⊤⍲}DlgCroiseAUD A;B;C;D;E;F;G;H;I;J;K;L;↑Δ▽⍞;⍞←≥ ;Fnt1;Fnt2;∩
IO;∩ML
[1]     ∩IO¢1 P ∩ML¢2
[2]     £(∩NCβ⍞←≥Δ⊤⍲β)Va P ⍞←≥Δ⊤⍲¢ββ
[3]    a:⍞←≥¢0÷V⍞←≥Δ⊤⍲¢,⍞←≥Δ⊤⍲
[4]     ↑Δ▽⍞¢⍞←≥Δ⊤⍲,(⍞←≥/β.β),βDLGβ,βCroiseAUDβ
[5]     βFnt1β∩WCβFontβ(βSizeβ 17)(βPNameβ βAPLβ)(βWeightβ 1000)
[6]     βFnt2β∩WCβFontβ(βSizeβ 15)(βPNameβ βMS Sans Serifβ)
[7]     ↑Δ▽⍞ ∩WC((3F~⍞←≥)¦βSubβ,βFormβ)(βCoordβ βPixelβ)(βAcceleratorβ 27 0)
(βEventβ 1001 1)(βPosnβ 97 187)(βSizeβ 325 439)(βCaptionβ βTableau croisé d
es audiencesβ)(βFontβ βFnt2β)
[8]     (↑Δ▽⍞,β.Label1β)∩WCβLabelβ(βPosnβ 14 11)(βSizeβ 13  23)(βCaptionβ
      βPaysβ)
[9]     (↑Δ▽⍞,β.Label2β)∩WCβLabelβ(βPosnβ 14 221)(βSizeβ 13 12)(βCaptionβ
      βauβ)
[10]    (↑Δ▽⍞,β.Label3β)∩WCβLabelβ(βPosnβ 14 116)(βSizeβ 13 51)(βCaptionβ
      βPériode duβ)
[11]    (↑Δ▽⍞,β.Label4β)∩WCβLabelβ(βPosnβ 59 116)(βSizeβ 13 28)(βCaptionβ
      βCiblesβ)
[12]    (↑Δ▽⍞,β.Label5β)∩WCβLabelβ(βPosnβ 59 11)(βSizeβ 13  40)(βCaptionβ
      βChaînesβ)
[13]    (↑Δ▽⍞,β.Label7β)∩WCβLabelβ(βPosnβ 199 190)(βSizeβ 13 88)(βCaptionβ
      βTableau croisé parβ)
[14]    (↑Δ▽⍞,β.Label8β)∩WCβLabelβ(βPosnβ 59 221)(βSizeβ 13 85)(βCaptionβ
      βTranches horairesβ)
[15]    (↑Δ▽⍞,β.Button1β)∩WCβButtonβ(βPosnβ 13 332)(βSizeβ 23 89)(βCaptionβ
      βOKβ)
[16]    (↑Δ▽⍞,β.Button2β)∩WCβButtonβ(βPosnβ 43 332)(βSizeβ 23 89)(βCaptionβ
      βAnnulerβ)
[17]    (↑Δ▽⍞,β.Button3β)∩WCβButtonβ(βPosnβ 73 332)(βSizeβ 23 89)(βCaptionβ
      βAideβ)
[18]    (↑Δ▽⍞,β.CheckBox1β)∩WCβButtonβ(βStyleβ βCheckβ)(βPosnβ
      126 332)(βSizeβ 17 85)(βCaptionβ βTtes tranchesβ)
[19]    (↑Δ▽⍞,β.ComboBox1β)∩WCβComboβ(βPosnβ 31 11)(βSizeβ 21 96)
[20]    (↑Δ▽⍞,β.ComboBox2β)∩WCβComboβ(βPosnβ 31 116)(βSizeβ 21 96)
[21]    (↑Δ▽⍞,β.ComboBox3β)∩WCβComboβ(βPosnβ 31 221)(βSizeβ 21 96)
[22]    (↑Δ▽⍞,β.ListBox1β)∩WCβListβ(βPosnβ 76 11)(βSizeβ 120 97)
[23]    (↑Δ▽⍞,β.ListBox2β)∩WCβListβ(βPosnβ 76 116)(βSizeβ 120    97)
[24]    (↑Δ▽⍞,β.ListBox3β)∩WCβListβ(βPosnβ 76 221)(βSizeβ 120    97)
[25]    (↑Δ▽⍞,β.ListBox4β)∩WCβListβ(βPosnβ 215 10)(βSizeβ 76 131)
[26]    (↑Δ▽⍞,β.ListBox5β)∩WCβListβ(βPosnβ 215 190)(βSizeβ 76 128)
[27]    (↑Δ▽⍞,β.Button4β)∩WCβButtonβ(βPosnβ 218 153)(βSizeβ 23 25)(βCaptionβ
      β>β)
[28]    (↑Δ▽⍞,β.Button5β)∩WCβButtonβ(βPosnβ 248 153)(βSizeβ 23 25)(βCaptionβ
      β<β)
[29]    (↑Δ▽⍞,β.Button6β)∩WCβButtonβ(βPosnβ 218 332)(βSizeβ 23 25)(βFontβ
      βFnt1β)
[30]    (↑Δ▽⍞,β.Button7β)∩WCβButtonβ(βPosnβ 248 332)(βSizeβ 23 25)(βFontβ
      βFnt1β)
```

```
[31]   ¨↑Δ▽▯,ß.↑Δ▽▯¢↑Δ▽▯ß
[32]   ¨↑Δ▽▯,ß.▯←≥¢▯←≥ß
[33]   ∩SE.storefnsßß∩NSßß
[34]  K∩NQ(↑Δ▽▯,ß.___ß)ßGotFocusß
[35]   £▯←≥V0
[36]   R¢3○3U∩DQ ↑Δ▽▯
[37]   ∩EX ↑Δ▽▯
[38]   £0
[39]
[40]   S
[41]   A¢Close A K ¨↑Δ▽▯
[42]   £▯←≥V0
[43]   A¢0
[44]   ¨ß#.ß,↑Δ▽▯,ß.Button2.Selectß
[45]
[46]   S
[47]   Select;A;B;C;D;E;F;G;H;I;J;K;L K ¨↑Δ▽▯
[48]   £▯←≥V¬
[49]   ¨ß#.ß,↑Δ▽▯,ß.Button2.Selectß
[50]   £0
[51]  ¬:
[52]   ∩EXßß∩NSßß
[53]
[54]   S
[55]   Select;A;B;C;D;E;F;G;H;I;J;K;L K ¨↑Δ▽▯,ß.Button1ß
[56]   £##.▯←≥V¬
[57]   A¢0
[58]   ∩NQ(ßß##.∩NSßß)1001 A
[59]   £0
[60]  ¬:
[61]   ∩EXßß##.∩NSßß
[62]
[63]   S
[64]   Select;A;B;C;D;E;F;G;H;I;J;K;L K ¨↑Δ▽▯,ß.Button2ß
[65]   £##.▯←≥V¬
[66]   ∩NQ(ßß##.∩NSßß)1001 å32768
[67]   £0
[68]  ¬:
[69]   ∩EXßß##.∩NSßß
[70]
[71]   S
[72]   Select;A;B;C;D;E;F;G;H;I;J;K;L;FindWindow;
     GetWindowsDirectory;SetWindowPos;WinHelp
      K ¨↑Δ▽▯,ß.Button3ß
[73]   ∩NAßI4 user.exe.P16|FindWindow I4 <0Tß
[74]   ∩NAßI2 kernel.exe.P16|GetWindowsDirectory =0T I2ß
[75]   ∩NAßI2 user.exe.P16|SetWindowPos I4 I2 I2 I2 I2 I2 Uß
[76]   ∩NAßI2 user.exe.P16|WinHelp I4 =0T U U4ß
[77]   A¢(ßß##.∩NSßß)∩WGßHandleß
```

```
[78]   B¢2oGetWindowsDirectory(128Vß ß)128
[79]   B¢B,ß\MPLAYER.HLPß K change to your own help file!
[80]   C¢WinHelp A B 3 0 K change (3 0) to (1,your own topic    #)!
[81]   C¢WinHelp A B 3 0 K call it twice for next instructions  to work
[82]   D¢FindWindow 0 ßAide du Lecteur multimédiaß K change to your help
     file title!
[83]   E¢SetWindowPos D å1 0 0 0 0 3
     S
```

Il ne reste plus dans ce cas qu'à compléter le programme en initialisant les différents contrôles avec les bonnes valeurs et à compléter ou écrire de nouvelles petites fonctions de gestion d'événements en bas du programme.

Ce programme fait appel à l'utilitaire storefns développé par Uniware et publié dans Vector Vol.11 No.3 Jan. 95.

Conclusion

Nous n'avons fait qu'abordé très succinctement l'écriture d'une DLL Delphi pour utilisation depuis APL. Toutefois, les principes généraux ont été énoncés de façon à permettre à chacun d'entre vous, s'il le désire, d'explorer plus avant ces possibilités.

L'utilisation de fonctions compilées dans une DLL depuis APL, offre de nombreux avantages : appel instantané, rapidité d'exécution, accès aisé à certaines parties de Windows auxquelles APL n'a pas ou pas facilement accès (comme le multimedia, par exemple). Jusqu'à présent, le développement d'une DLL était l'affaire de spécialistes : avec Delphi, la tâche devient à la portée de tout APListe qui accepte d'investir un peu de son temps.

Le fait de pouvoir facilement incorporer des fenêtres Delphi dans une DLL est vraiment quelque chose de très puissant, qui ouvre des horizons pour le développeur.

Cet article est la traduction du début d'un chapitre d'un livre en cours de préparation, sur la Programmation APL sous Windows.