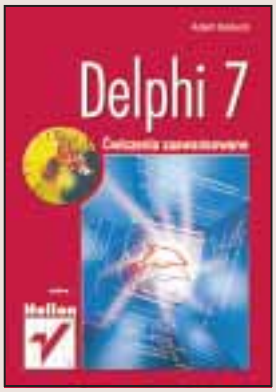
Delphi 7. Ćwiczenia zaawansowane



Autor: Adam Boduch
ISBN: 83-7361-076-6
Format: B5, stron: 158
Przykłady na ftp: 951 kB

W 2002 roku firma Borland zaprezentowała na polskim rynku nową wersję narzędzia typu RAD, służącego do szybkiego tworzenia aplikacji – Delphi. Jest to już 7. wersja tego popularnego pakietu. Wprowadzono sporo nowości: poprawiono środowisko programistyczne IDE, dodano nowe komponenty VCL i wprowadzono kilka zmian w bibliotece uruchomieniowej. Delphi 7 zostało także przystosowane do platformy .NET Microsoftu.

Jeśli chcesz szybko zapoznać się z możliwościami nowego Delphi, książka „Delphi 7. Ćwiczenia zaawansowane" to idealny przewodnik dla Ciebie. Znajdziesz tu wiele ćwiczeń dotyczących różnych obszarów zaawansowanego wykorzystania Delphi; od programowania baz danych po programowanie internetowe.

Dzięki książce poznasz:

- Zmiany i nowości wprowadzone w Delphi 7
- Programowanie sieciowe w Delphi 7: korzystanie z gniazd i protokołów SMTP i HTTP
- Tworzenie kontrolek ActiveX
- Szybkie pisanie aplikacji internetowych z użyciem komponentów IntraWeb
- Sposoby korzystania z baz danych za pomocą dbExpress
- Metody pozyskiwania z poziomu Delphi informacji o sprzęcie i oprogramowaniu, sterowanie procesami

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 3.

ActiveX

Ten rozdział będzie poświęcony w całości kontrolkom ActiveX. Do czego służą oraz co to jest ActiveX? W jaki sposób umieszczać kontrolki ActiveX w Internecie oraz jak je tworzyć na bazie komponentów VCL. Tego wszystkiego dowiesz się czytając ten rozdział.

Wcześniej jednak musisz zapoznać się z pojęciem COM...

Co to jest COM?

COM — od ang. *Component Object Model*. **Specyfikacja firmy Microsoft, która w założeniu dotyczy tworzenia obiektów wielokrotnego użytku, niezależnie od języka programowania.**

Żeby zrozumieć ActiveX, musisz zrozumieć COM — postaram się to zwięźle wytłumaczyć. Otóż firma Microsoft wymyśliła model obiektów, które mogą być wykorzystywane w każdym środowisku programistycznym Win32. Wynikiem powstania obiektu COM jest kontrolka z rozszerzeniem *.dll*. Kontrolka taka może być wykorzystana zarówno w Delphi, jak i Visual C++, C++ Builderze, czy Visual Basicu.

Tworzenie obiektów COM

COM jest podstawą dla ActiveX i dla OLE. Żeby zrozumieć istotę działania ActiveX, musisz zrozumieć istotę działania COM — stąd następujące ćwiczenie.

Ćwiczenie 3.1.

Tworzenie nowego projektu.

1. Z menu Delphi wybierz *New/New/Other*. Pojawi się Repozytorium. Zaznacz zakładkę *ActiveX* (rysunek 3.1).

Rysunek 3.1.
Zakładka ActiveX Repozytorium

2. Zaznacz w tym oknie ikonę *ActiveX Library* i naciśnij *OK*. W tym momencie zostanie utworzony pusty projekt.
3. Z polecenia *File* wybierz pozycję *Save*. Wskaż miejsce, gdzie Delphi ma zapisać plik.

Utworzyliśmy właśnie pusty projekt, ale na razie do niczego nam to nie służy. Utworzenie właściwego obiektu COM też jest proste — polega na wybraniu ikony *Com Object*.

Ćwiczenie 3.2.

Tworzenie obiektu COM.

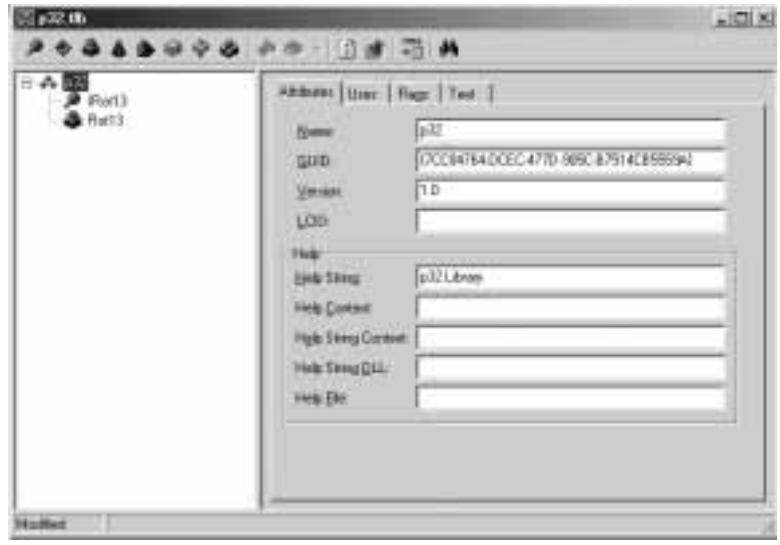
1. Mając otwarty projekt z ćwiczenia 3.1 ponownie wybierz pozycję *File/New/Other*.
2. Z zakładki *ActiveX* tym razem wybierz pozycję *Com Object*. Delphi wyświetli okno kreatora obiektów COM, widoczne na rysunku 3.2.

Rysunek 3.2.
Kreator obiektów COM

3. W polu `Class Name` wpisz `Rot13`.

4. Pole `Description` służy do wstawienia krótkiego opisu obiektu. Możesz wpisać np. `Algorytm Rot13`.
5. Przyciskiem *OK* zamknij okno. Obiekt COM został utworzony.
6. Z menu *File* wybierz *Save All* i wpisz nazwę dla modułu.

Na pierwszym planie znajduje się okno edytora biblioteki typu (rysunek 3.3). Za pomocą tego edytora sterujemy obiektem COM. Wszystkie zmiany dokonane w tym edytorze znajdą odzwierciedlenie w module.



Rysunek 3.3. *Okno edytora biblioteki typu; dodawanie właściwości oraz metod*

Dodawanie nowych metod oraz właściwości jest dość proste. Wszystko można zrobić za pomocą edytora biblioteki. Wystarczy kliknąć prawym przyciskiem myszy pozycję *IRot13* i z menu *New* wybrać *Properties* lub *Methods*.

Ćwiczenie 3.3.

Dodawanie nowej metody.

1. Kliknij prawym przyciskiem myszy pozycję *IRot13* i z menu *New* wybierz *Method*.

Rysunek 3.4. *Dodawanie nowej metody*

2. Po tym kroku pojawi się w obrębie edytora nowa pozycja — nazwij ją `Rot13` (zakładka *Attributes*).
3. Następnie przejdź do zakładki *Parameters*. Nasza funkcja `Rot13` z założenia nie będzie posiadać żadnych parametrów. Jedyne, co tu będziesz musiał ustalić, to wartość zwracaną przez funkcję.
4. Z pozycji `Return Type` wybierz `LPSTR`.
5. Możesz zapisać cały projekt (*Ctrl+Shift+S*).

Zmiany, których dokonaliśmy w edytorze zostały oddane w module `Rot13Frm`.

Wydruk 3.1. *Kod źródłowy modułu Rot13Frm.pas*

```
unit Rot13Frm;

{$WARN SYMBOL_PLATFORM OFF}

interface

uses
  Windows, ActiveX, Classes, ComObj, p32_TLB, StdVcl;

type
  TRot13 = class(TTypedComObject, IRot13)
  protected
    function Rot13: PChar; stdcall;
    {Declare IRot13 methods here}
  end;

implementation

uses ComServ;

function TRot13.Rot13: PChar;
begin

end;

initialization
  TTypedComObjectFactory.Create(ComServer, TRot13, Class_Rot13,
    ciMultiInstance, tmApartment);
end.
```

W module tym możesz wpisać kod funkcji `Rot13`. Podczas gdy Ty pracujesz z modułem `Rot13Frm`, Delphi pracuje nad tworzeniem modułu przechowującego kod biblioteki. W edytorze kodu cały czas masz otwartą zakładkę *p32_TLB.pas* (do nazwy projektu jest dodawana końcówka `_TLB`).

Wydruk 3.2. *Kod źródłowy biblioteki*

```
unit p32_TLB;

// ************************************************************************ //
// WARNING
// -------
// The types declared in this file were generated from data read from a
// Type Library. If this type library is explicitly or indirectly (via
// another type library referring to this type library) re-imported, or the
```

```
// 'Refresh' command of the Type Library Editor activated while editing the
// Type Library, the contents of this file will be regenerated and all
// manual modifications will be lost.
// ************************************************************************ //

// PASTLWTR : 1.2
// File generated on 2002-10-30 15:23:00 from Type Library described below.

// ************************************************************************  //
// Type Lib: C:\Moje dokumenty\Pliki textowe\delphi_cwiczenia_zaawansowane\
   ➥listingi\3\3.3\p33.tlb (1)
// LIBID: {7CC84764-DCEC-477D-9D5C-B7514CB5559A}
// LCID: 0
// Helpfile:
// HelpString: p32 Library
// DepndLst:
//   (1) v2.0 stdole, (C:\WINDOWS\SYSTEM\stdole2.tlb)
// ************************************************************************ //
{$TYPEDADDRESS OFF} // Unit must be compiled without type-checked pointers.
{$WARN SYMBOL_PLATFORM OFF}
{$WRITEABLECONST ON}
{$VARPROPSETTER ON}
interface

uses Windows, ActiveX, Classes, Graphics, StdVCL, Variants;


// *********************************************************************//
// GUIDS declared in the TypeLibrary. Following prefixes are used:
//   Type Libraries     : LIBID_xxxx
//   CoClasses          : CLASS_xxxx
//   DISPInterfaces     : DIID_xxxx
//   Non-DISP interfaces: IID_xxxx
// *********************************************************************//
const
  // TypeLibrary Major and minor versions
  p32MajorVersion = 1;
  p32MinorVersion = 0;

  LIBID_p32: TGUID = '{7CC84764-DCEC-477D-9D5C-B7514CB5559A}';

  IID_IRot13: TGUID = '{93ACF1F7-8577-4161-9AD6-F89EC5EF4899}';
  CLASS_Rot13: TGUID = '{921E69B0-394B-4561-A689-CD8ECE28A6E4}';
type

// *********************************************************************//
// Forward declaration of types defined in TypeLibrary
// *********************************************************************//
  IRot13 = interface;

// *********************************************************************//
// Declaration of CoClasses defined in Type Library
// (NOTE: Here we map each CoClass to its Default Interface)
// *********************************************************************//
  Rot13 = IRot13;


// *********************************************************************//
// Interface: IRot13
// Flags:     (256) OleAutomation
// GUID:      {93ACF1F7-8577-4161-9AD6-F89EC5EF4899}
```

```
// ************************************************************************//
  IRot13 = interface(IUnknown)
    ['{93ACF1F7-8577-4161-9AD6-F89EC5EF4899}']
    function Rot13: PChar; stdcall;
  end;

// ************************************************************************//
// The Class CoRot13 provides a Create and CreateRemote method to
// create instances of the default interface IRot13 exposed by
// the CoClass Rot13. The functions are intended to be used by
// clients wishing to automate the CoClass objects exposed by the
// server of this typelibrary.
// ************************************************************************//
  CoRot13 = class
    class function Create: IRot13;
    class function CreateRemote(const MachineName: string): IRot13;
  end;

implementation

uses ComObj;

class function CoRot13.Create: IRot13;
begin
  Result := CreateComObject(CLASS_Rot13) as IRot13;
end;

class function CoRot13.CreateRemote(const MachineName: string): IRot13;
begin
  Result := CreateRemoteComObject(MachineName, CLASS_Rot13) as IRot13;
end;

end.
```

Ten kod to na pozór wiele niezrozumiałych instrukcji. Przykładowo, zwróć uwagę na fragment:

```
  IRot13 = interface(IUnknown)
    ['{93ACF1F7-8577-4161-9AD6-F89EC5EF4899}']
    function Rot13: PChar; stdcall;
  end;
```

Jest to deklaracja interfejsu `IRot13`.

Interfejs COM jest sposobem łączności użytkownika z funkcjami COM.

Dzięki interfejsowi COM masz dostęp do tego, co oferuje sama kontrolka COM. W edytorze biblioteki należało utworzyć metodę, która następnie została dopisana do interfejsu, w module *p32_TBL.pas*. Dzięki temu użytkownik korzystający z kontrolki ma możliwość wykorzystania funkcji `Rot13`.

Tak jak nazwy komponentów zaczynają się od litery `T`, nazwy interfejsów COM rozpoczynają się od litery `I` — jest to zasada uniwersalna i nie należy jej łamać.

Podstawowym interfejsem jest interfejs `IUnknow`, tak samo jak podstawową klasą w VCL jest `TObject`.

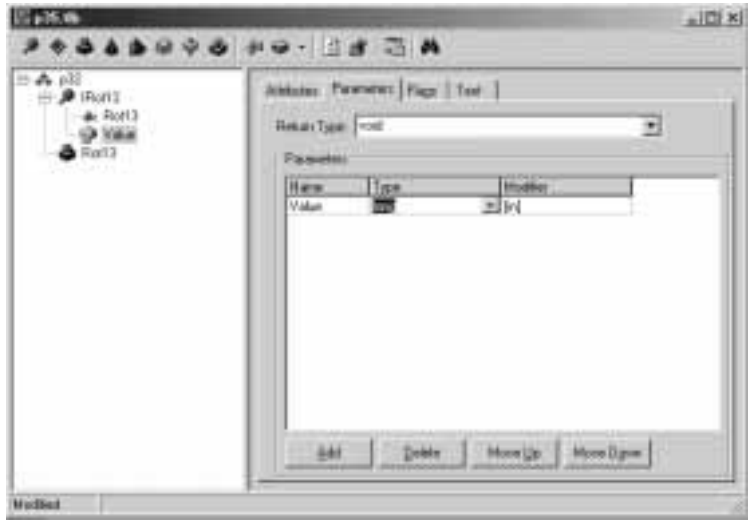
Ćwiczenie 3.4.

Dodawanie właściwości.

Gdy chcemy dodać właściwość, edytor biblioteki, tak jak w ćwiczeniu poprzednim, automatycznie wygeneruje dwie gałęzie. Domyślnie zakłada bowiem, że jedna z nich potrzebna jest do zapisywania danych do właściwości, a druga do ich odczytu. My jednak potrzebujemy właściwości „tylko do zapisu".

1. W tym celu odszukaj na pasku przycisk *New Property*.
2. Obok tego przycisku znajduje się strzałka — w wyniku jej rozwinięcia pokażą się pozycje, wśród których znajduje się *Write Only* — wybierz ją.
3. Zostanie utworzona jedna gałąź.
4. Zmień jej nazwę na `Value`.
5. Z listy rozwijanej `Type` wybierz pozycję `LPSTR`.
6. Ostatnim krokiem będzie wejście w zakładkę *Parameters* i ustawienie opcji parametru.
7. Z listy rozwijanej `Return Type` wybierz `void` (oznacza, że nie będzie wartości zwracanej przez funkcję). Natomiast musisz ustawić parametr owej procedury. Spójrz na rysunek 3.5.

Rysunek 3.5. *Ustawienie typu parametru*



8. Po kliknięciu napisu `long` pojawi się strzałka służąca do rozwijania listy z możliwymi typami danych. Wybierz `LPSTR`.

To właściwie wszystko — teraz możesz zapisać cały projekt. W module `Rot13Frm` znajduje się taka linia:

```
procedure Set_Value(Value: PChar); stdcall;
```

Jeżeli użytkownik nada zmiennej `Value` nową wartość, zostanie ona przekazana jako parametr procedury `Set_Value`.

Wpisywanie kodu — ROT13

Nasz przykładowy obiekt COM będzie korzystał z algorytmu ROT13 w celu zakodowania tekstu. Zasada działania tego algorytmu jest bardzo prosta. Otóż przekształca on znak na numer ASCII i następnie dodaje do tego numeru liczbę 13, po czym ponownie przekształca liczbę w znak.

Funkcja `Rot13` wygląda mniej więcej tak:

```
function TRot13.Rot13: PChar;
var
  i : Integer;
  Sin, Sout : String;
begin
{ przypisanie danych do zmiennej String }
  Sin := FRot13;

  for I := 1 to Length(Sin) do  // pętla po wszystkich znakach
  begin
  { jeżeli litera jest większa (równa) od A, a mniejsza (równa) od M }
    if (UpCase(Sin[i]) >= 'A') and (UpCase(Sin[i]) <= 'M')
      then Sout := Sout + Chr(Ord(Sin[i]) + 13 )  // kodowanie
    else if (UpCase(Sin[i]) >= 'N') and (UpCase(Sin[i]) <= 'Z')
      then Sout := Sout + Chr(Ord(Sin[i]) - 13)
    else Sout := Sout + Sin[i];
  end;

  Result := PChar(Sout);
end;
```

Na końcu funkcja zwraca zakodowany ciąg znaków. Aby wszystko było bardziej zrozumiałe, przedstawię Ci cały moduł `Rot13Frm` (wydruk 3.3).

Wydruk 3.3. *Moduł Rot13Frm.pas zawierający funkcję Rot13*

```
unit Rot13Frm;

{$WARN SYMBOL_PLATFORM OFF}

interface

uses
  Windows, ActiveX, Classes, ComObj, p32_TLB, StdVcl;

type
  TRot13 = class(TTypedComObject, IRot13)
  private
    FRot13 : PChar;
  protected
    function Rot13: PChar; stdcall;
    procedure Set_Value(Value: PChar); stdcall;

    {Declare IRot13 methods here}
  end;
```

```
implementation

uses ComServ;

function TRot13.Rot13: PChar;
var
  i : Integer;
  Sin, Sout : String;
begin
{ przypisanie danych do zmiennej String }
  Sin := FRot13;

  for I := 1 to Length(Sin) do  // pętla po wszystkich znakach
  begin
  { jeżeli litera jest większa (równa) od A, a mniejsza (równa) od M }
    if (UpCase(Sin[i]) >= 'A') and (UpCase(Sin[i]) <= 'M')
      then Sout := Sout + Chr(Ord(Sin[i]) + 13 )  // kodowanie
    else if (UpCase(Sin[i]) >= 'N') and (UpCase(Sin[i]) <= 'Z')
      then Sout := Sout + Chr(Ord(Sin[i]) - 13)
    else Sout := Sout + Sin[i];
  end;

  Result := PChar(Sout);
end;

procedure TRot13.Set_Value(Value: PChar);
begin
  FRot13 := Value;
end;

initialization
  TTypedComObjectFactory.Create(ComServer, TRot13, Class_Rot13,
    ciMultiInstance, tmApartment);
end.
```

Na początku za pomocą procedury `Set_Value` do zmiennej `FRot13` (sekcja `private`) zostanie przypisana wartość z parametru owej procedury `Set_Value`. Dzięki temu funkcja `Rot13` będzie mogła odczytać tę wartość i zakodować algorytmem `Rot13`, po czym zwrócić ciąg znaków.

Budowa i rejestracja kontrolki

Mamy już cały potrzebny kod do skompilowania biblioteki. Z menu *Project* wybierz pozycję *Build*. Kontrolka zostanie skompilowana do postaci pliku DLL.

Jeżeli nie ma żadnych błędów, to możesz zarejestrować kontrolkę poprzez wybranie z menu *Run* pozycji *Register ActiveX Server*. Jeżeli wszystko się powiedzie, powinieneś otrzymać komunikat o prawidłowej rejestracji. Kontrolka jest gotowa do użycia.

Wykorzystanie obiektu COM

Ćwiczenie 3.5.

Wykorzystanie obiektu COM.

Nasza przykładowa aplikacja wcale nie musi być skomplikowana. Wystarczy, że będzie zawierać przycisk oraz kontrolkę edycyjną. W katalogu z naszą przykładową aplikacją musi się znaleźć plik *p32_TLB.pas*, gdyż zawiera interfejsy i klasy COM.

Aby cały projekt prawidłowo się skompilował, odszukaj sekcję `uses` i dodaj do niej następujące moduły:

```
uses p32_TLB, ComObj;
```

Pierwszy zawiera oczywiście deklarację interfejsu COM, a drugi zawiera nagłówek funkcji `CreateComObject`.

```
procedure TMainForm.btnConverClick(Sender: TObject);
var
  Rot : IRot13;
begin
  Rot := CreateComObject(CLASS_ROT13) as IRot13; // utwórz obiekt COM
  if Assigned(Rot) then
  begin
    Rot.Set_Value(PChar(edtValue.Text)); // przypisz wartość odczytaną z kontrolki edycyjnej
    edtValue.Text := Rot.Rot13; // do kontrolki przypisz zakodowany tekst
  end;
end;
```

Oto jak wygląda użycie kontrolki COM. Najpierw tworzymy obiekt COM, żeby później skorzystać z funkcji, jakie oferuje nam jej interfejs. Rysunek 3.6 przedstawia działanie przykładowego programu.

Rysunek 3.6.
Tekst „Delphi 7 Studio" zakodowany za pomocą algorytmu ROT13

Oczywiście algorytm działa w dwie strony. Po zakodowaniu tekstu i ponownym naciśnięciu przycisku następuje jego rozkodowanie.

Czym jest ActiveX?

ActiveX to technologia oparta na COM. Pozwala na tworzenie kontrolek *.ocx* lub *.dll*. Tak naprawdę ActiveX to obiekt COM, tyle że posiadający własny interfejs dostępny na poziomie projektowania. Wygląda to w ten sposób, iż tworzony jest zwykły formularz VCL będący w rzeczywistości kontrolką ActiveX. Można korzystać z wszystkich komponentów i, ogólnie rzecz biorąc, projektowanie jest łatwiejsze niż w przypadku zwykłych obiektów COM. Dodatkowo ActiveX umożliwia wygenerowanie kodu pozwalającego na umieszczenie jej w Sieci, na stronie WWW.

Importowanie kontrolek ActiveX

Korzystając z Delphi, możesz nawet nie wiedzieć, że w rzeczywistości używasz kontrolki ActiveX. Po zaimportowaniu do Delphi taka kontrolka przedstawia się jak zwykły komponent i znajduje się na palecie komponentów. Przykład? Komponent `TWebBrowser` (paleta *Internet*). Komponent ten służy do wyświetlania stron WWW, ale w rzeczywistości jest to kontrolka ActiveX przeglądarki Internet Explorer. Tak więc mając zainstalowaną przeglądarkę, masz również kontrolkę ActiveX, której z kolei możesz użyć w Delphi.

Ćwiczenie 3.6.

Importowanie kontrolki ActiveX.

1. Z menu *Component* wybierz pozycję *Import ActiveX Control*. Powinno się pokazać okno *Import ActiveX* (rysunek 3.7).

Rysunek 3.7.
Okno Import ActiveX

Człon okna stanowi lista rozwijana, zawierająca kontrolki znajdujące się w systemie. Za pomocą przycisku *Add* możesz dodać do tej listy nową kontrolkę, ale tym zajmiemy się nieco później. W naszym ćwiczeniu do programu zaimportujemy kontrolkę *Shockwave Flash*. Dzięki tej kontrolce będziemy mogli w naszych aplikacjach wyświetlać filmy w formacie Flash. Jeżeli nie możesz na liście odnaleźć tej kontrolki, oznacza to, że nie masz jej w systemie.

2. Przed zaimportowaniem kontrolki wybierz jeszcze paletę, na której zostanie ona zainstalowana. Ja z listy *Palette page* wybrałem `Standard`.

3. Teraz możesz nacisnąć przycisk *Install*. Spowoduje on wyświetlenie okna *Install*. Właściwie w tym oknie nie musimy dokonywać żadnych zmian — naciśnij *OK*. Po tej operacji wyświetli się okno, takie jak na rysunku 3.8.

Rysunek 3.8. *Dodawanie obiektu do pakietu*

4. Delphi zapyta Cię, czy skompilować pakiet. Jeżeli naciśniesz *Yes*, spowoduje to zainstalowanie kontrolki w palecie komponentów — potwierdzeniem tego będzie komunikat (rysunek 3.9).

Rysunek 3.9. *Potwierdzenie instalacji obiektu*

Od tej chwili na palecie komponentów (zakładka *Standard*) pojawi się `TShockwaveFlash`.

Wykorzystanie komponentu TShockwaveFlash

Pobawmy się trochę kontrolką Flash i wyświetlaniem animacji.

Ćwiczenie 3.7.

Ładowanie i wyświetlanie animacji w komponencie.

1. Umieść komponent `TShockwaveFlash` na formularzu i nazwij go `Flash`.
2. Umieść także komponent `TButton` — będzie on służył do ładowania filmu. Po naciśnięciu przycisku wyświetli się okno służące do wyboru pliku — potrzebujemy więc jeszcze komponentu `OpenDialog`. Procedura ładująca plik *SWF* (format Flasha) wygląda następująco:

```
procedure TMainForm.btnLoadClick(Sender: TObject);
begin
  if OpenDialog.Execute then // podczas wyświetlania okna
  begin
    Flash.Movie := OpenDialog.FileName; // przypisz ścieżkę do pliku
    Flash.Play; // rozpocznij odtwarzanie
  end;
end;
```

Jak widzisz, odtwarzanie animacji Flasha jest dziecinnie proste. Wystarczy do właściwości `Move` przypisać ścieżkę do pliku. Po wykonaniu metody `Play` rozpoczyna się odtwarzanie.

Jedyna pułapka, jaka czyha na programistę to konieczność podawania **pełnej** ścieżki do pliku. Można by pomyśleć, że skoro plik *swf* znajduje się w tym samym katalogu, co program, wystarczy, że ścieżka będzie przypisana w następujący sposób:

```
Flash.Move := 'plik.swf';
```

Nic bardziej mylnego — w takim wypadku plik nie zostanie załadowany. Należy to zrobić w taki sposób:

```
Flash.Movie := ExtractFilePath(Application.ExeName) + '3.2.swf';
```

Tworzenie kontrolek ActiveX

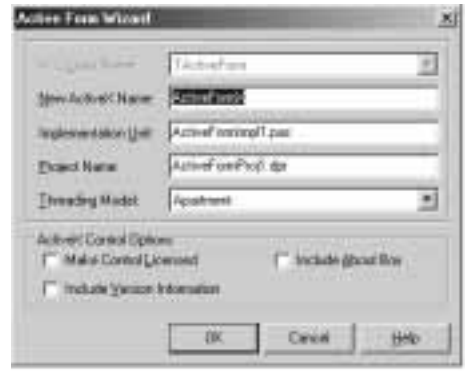
Tworzenie obiektów ActiveX jest równie proste jak tworzenie obiektów COM. Jak już wspomniałem, podczas projektowania kontrolek ActiveX możemy używać wizualnej biblioteki komponentów (VCL).

Ćwiczenie 3.8.

Tworzenie nowej kontrolki.

Proces ten jest podobny do tworzenia obiektów COM.

1. Z menu *File* wybierz *New*, a następnie *Other*.
2. Kliknij zakładkę *ActiveX*, wybierz ikonę *ActiveX Form*. Na rysunku 3.10 przedstawione jest okno, które zostanie wyświetlone w wyniku tej operacji.



Rysunek 3.10. *Okno Active Form Wizard*

3. W polu *New ActiveX Name* wpisz `ActiveXTest`.
4. Pole *Implementation Unit* niech zawiera wartość `ActiveXTestFrm.pas`, a *Project Name*: `ActiveXTestProj.dpr`.
5. Naciśnij *OK*, a Delphi utworzy kontrolkę ActiveX i wyświetli formularz.
6. Z menu *File* wybierz *Save All*. Podczas zapisywania domyślną nazwą pliku, jaką zaproponuje Delphi, będzie ta, którą wpisałeś w oknie tworzenia kontrolki ActiveX.

Katalog, który wybrałeś wzbogacił się w parę nowych plików. Oprócz standardowych plików znajduje się tam również plik *ActiveXTestProj_TLB.pas*. Plik ten zawiera interfejsy COM.

Przykładowa kontrolka ActiveX

Napiszmy przykładową kontrolkę ActiveX, która pozwoli Ci zrozumieć istotę działania tych obiektów. Nasza przykładowa kontrolka będzie umożliwiać wyświetlanie napisów do filmów.

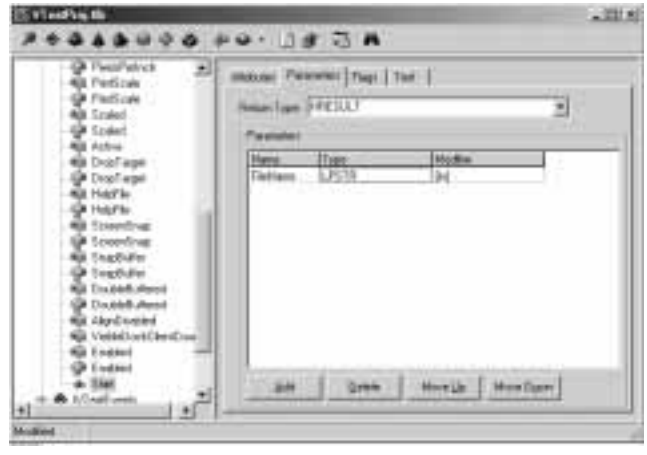
Tworzenie interfejsu COM

Na samym początku procesu tworzenia kontrolki musimy, tak jak w przypadku kontrolek COM, stworzyć dwie metody `Start` i `Stop`.

Ćwiczenie 3.9.

Tworzenie metod Start i Stop.

1. Stwórz nową kontrolkę ActiveX. Postępuj tak, jak w poprzednim ćwiczeniu — w polu `New ActiveX Name` kreatora wpisz `VText`.
2. W polu `Implementation Unit` wpisz `VTextFrm.pas`, a w `Project Name` — `VTextProj.dpr`.
3. Zapisz cały projekt.
4. W katalogu z kontrolką znajduje się plik *VTextProj_TLB.pas*. Domyślnie ten plik nie jest otwarty w projekcie. Otwórz więc ten plik — w edytorze kodu utworzona zostanie nowa zakładka, a na pierwszy plan wysunie się edytor biblioteki.
5. Naciśnij prawym przyciskiem myszy pozycję `IVText` i z menu wybierz *New/Method*.
6. Metodę tę nazwij `Start`.
7. Kliknij zakładkę *Parameters*. Będziesz musiał dodać nowy parametr dla funkcji. Doprowadź parametr do takiej postaci, jak na rysunku 3.11.



Rysunek 3.11. *Parametr FileName metody Start*

Stwórz teraz metodę `Stop`, powtórz cały proces, lecz nasza procedura nie będzie zawierała żadnych parametrów — nie musisz robić nic więcej. Zapisz cały projekt.

Tworzenie kontrolki ActiveX

Nasza kontrolka będzie wyświetlać napisy do filmów. Wiele odtwarzaczy multimedialnych oferuje możliwość załadowania napisów do filmu, który akurat oglądamy w innej wersji językowej. Nasza kontrolka będzie uwzględniała plik z napisami, którego poszczególne linie zapisane są w ten sposób:

```
...
00:37:46:Proszę.
00:37:53:Pan musi być naprawdę ważny...
...
```

Czyli format czasowy. Kontrolka w momencie wywołania metody Start zaczyna działać w pętli `while`. Zresztą zaraz się przekonasz, jak to wygląda...

Ćwiczenie 3.10.

Wygląd kontrolki ActiveX.

1. Przede wszystkim otwórz plik `VTextFrm` (jeżeli jeszcze nie jest otwarty w projekcie).
2. Klawiszem *F12* przełącz się do formularza.
3. Zmniejsz ten formularz — dostosuj go do własnych wymagań.
4. Umieść na formularzu komponent `TPanel`, a jego właściwość `Align` ustaw na `alClient`.
5. Właściwość `BevelInner` zmień na `bvLowered`, a sam komponent nazwij `pnlMessage`.
6. Na komponencie `pnlMessage` umieść obiekt `TLabel` i zmień właściwość `Align` na `alBottom`.
7. Nazwij komponent `lblCurrent`.

To by było na tyle, jeżeli chodzi o projektowanie kontrolki od strony wizualnej. Kolejnym krokiem jest tworzenie samego kodu kontrolki.

Ćwiczenie 3.11.

Kod źródłowy kontrolki.

Przełącz się do kodu pliku `VTextFrm`. Odnajdź sekcję `private` i dodaj następujące linie:

```
FLines : TStringList; // zmienna przechowuje napisy
FTime, FText : TArray; // tablica - czas oraz napis
FBroken : WordBool; // określa, czy proces jest uruchomiony
procedure PrepareText; // przygotuj (przeanalizuj) plik tekstowy
```

Skorzystałem tutaj z nowego typu danych — `TArray`. Jest to tablica dynamiczna, dodaj deklarację tego typu w sekcji `type`:

```
TArray = array of String;
```

Procedura PrepareText, którą zadeklarowaliśmy w sekcji private służy do przygotowania pliku tekstowego. Zaraz po wywołaniu metody Start do zmiennej FLines zostaje odczytany plik z napisami. Procedura PrepareText ma oddzielić z każdej linii: czas, w którym napis ma być wyświetlony oraz samą treść napisu.

```
procedure TVText.PrepareText;
var
  i : Integer;
begin
{ określ wielkość tablicy na podstawie liczby linii }
  SetLength(FTime, FLines.Count);
  SetLength(FText, FLines.Count);

  { pętla po wszystkich liniach... }
  for I := 0 to FLines.Count -1 do
  begin
  { do tego elementu tablicy przypisz czas, w którym powinien wyświetlić się napis }
    FTime[i] := (Copy(FLines[i], 1, 8));
  { tutaj przypisz samą treść }
    FText[i] := (Copy(FLines[i], 10, Length(FLines[i]) - 8));
  end;
end;
```

Dzięki temu mamy tablice FTime oraz FText gotowe do użycia. Teraz jedyny problem to wyświetlenie odpowiedniego elementu tablicy w odpowiednim czasie. A odpowiedzialną za to procedurą jest procedura Start.

```
procedure TVText.Start(FileName: PChar);
var
  Counter : Integer; // licznik - ile już napisów zostało wyświetlonych
  FPause : Integer; // czas wyświetlania napisu
  CurrentTime : TTime; // czas odtwarzania filmu
  wHour, wMin, wSec : Integer;
begin
  FLines := TStringList.Create;
  FLines.LoadFromFile(FileName); // załaduj plik tekstowy

  PrepareText; // przygotuj dwie tablice
  FBroken := False;

  Counter := -1;
  FPause := 0;
  wHour := 0;
  wMin := 0;
  wSec := 0;

  { pętla wyświetlana co 1000 milisekund dopóki zmienna FBroken nie ma wartości False }
  while (not FBroken) or (not Application.Terminated) do
  begin
    Application.ProcessMessages;
    Sleep(1000);  // odczekaj 1 sek.

    if FBroken then Break; // jeżeli zmienna = TRUE, przerwij działanie

    Inc(wSec); // zwiększ liczbę sekund
    if wSec >= 60 then // jeżeli liczba większa od 60...
    begin
      Inc(wMin); // zwiększ liczbę min.
      wSec := 0; // wyzeruj zmienną
    end;
```

```
    if wMin > 60 then // jeżeli liczba min > 60
    begin
      Inc(wHour); // zwiększ liczbę godzin
      wMin := 0; // wyzeruj minuty
    end;

    // na podstawie danych utwórz zmienną TTime
    CurrentTime := EncodeTime(wHour, wMin, wSec, 0);
    lblCurrent.Caption := TimeToStr(CurrentTime);

    if AnsiMatchStr(TimeToStr(CurrentTime), FTime) then
    begin
      Inc(Counter);
      pnlMessage.Caption := FText[Counter];
      FPause := 0;
    end else
    begin
      if Length(pnlMessage.Caption) > 0 then
      begin
        Inc(FPause);

        if FPause = 5 then
        begin
          FPause := 0;
          pnlMessage.Caption := '';
        end;
      end;
    end;
  end;
end;
```

Po załadowaniu napisów i wywołaniu procedury `PrepareText` mamy gotowe tablice. Pętla, która jest wykonywana w odstępie 1 sekundy, za każdym razem zwiększa liczbę sekund, następnie minut (jeżeli liczba sekund osiągnie 60) itd. Następnie za pomocą funkcji `EncodeTime`, dzięki zmiennym `wHour`, `wMin`, `wSec` możemy skonstruować typ `TTime`. Funkcja `AnsiMatchStr` sprawdza, czy dana wartość `CurrentTime` znajduje się w tablicy `FTime`.

Funkcja `AnsiMatchStr` znajduje się w module `StrUtils.pas`. Żeby wszystko zadziałało, musisz ten moduł dodać do listy *uses*.

Jeżeli tak, następuje wyświetlenie tekstu z tablicy `FText`. Jedynym problemem jest rozróżnienie, który element tablicy powinien być w tym momencie wyświetlony. Aby to zrealizować, należy wprowadzić zmienną `Counter`, która zwiększy się o jeden za każdym razem, gdy napis zostanie wyświetlony.

W dokumentacji Delphi jest błąd i funkcja `AnsiMatchStr` wcale nie zwraca liczby w postaci `Integer`, jak to jest napisane. W rzeczywistości zwraca `True`, jeżeli element został znaleziony, lub `False`, w przeciwnym razie.

Pozostało jeszcze napisanie procedury `Stop`. Procedura ta będzie służyć do wstrzymywania całego procesu. Jej kod jest prosty:

```
procedure TVText.Stop;
begin
  FBroken := True;
end;
```

Zmiana wartości zmiennej `FBroken` na `True` powoduje zatrzymanie działania pętli `while`. Cały kod źródłowy modułu jest przedstawiony na wydruku 3.4.

Wydruk 3.4. *Moduł VTextFrm.pas*

```
{
  Copyright (c) 2002 by Adam Boduch <adam@4programmers.net>
}

unit VTextFrm;

{$WARN SYMBOL_PLATFORM OFF}

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ActiveX, AxCtrls, VTextProj_TLB, StdVcl, StdCtrls, ExtCtrls, StrUtils;

type
  TArray = array of String;

  TVText = class(TActiveForm, IVText)
    pnlMessage: TPanel;
    lblCurrent: TLabel;
  private
    { Private declarations }
    FEvents: IVTextEvents;

    FLines : TStringList; // zmienna przechowuje napisy
    FTime, FText : TArray; // tablica - czas oraz napis
    FBroken : WordBool; // określa, czy proces jest uruchomiony
    procedure PrepareText; // przygotuj (przeanalizuj) plik tekstowy

    procedure ActivateEvent(Sender: TObject);
    procedure ClickEvent(Sender: TObject);
    procedure CreateEvent(Sender: TObject);
    procedure DblClickEvent(Sender: TObject);
    procedure DeactivateEvent(Sender: TObject);
    procedure DestroyEvent(Sender: TObject);
    procedure KeyPressEvent(Sender: TObject; var Key: Char);
    procedure PaintEvent(Sender: TObject);
  protected
    { Protected declarations }
    procedure DefinePropertyPages(DefinePropertyPage: TDefinePropertyPage); override;
    procedure EventSinkChanged(const EventSink: IUnknown); override;
    function Get_Active: WordBool; safecall;
    function Get_AlignDisabled: WordBool; safecall;
    function Get_AutoScroll: WordBool; safecall;
    function Get_AutoSize: WordBool; safecall;
    function Get_AxBorderStyle: TxActiveFormBorderStyle; safecall;
    function Get_Caption: WideString; safecall;
    function Get_Color: OLE_COLOR; safecall;
    function Get_DoubleBuffered: WordBool; safecall;
    function Get_DropTarget: WordBool; safecall;
    function Get_Enabled: WordBool; safecall;
    function Get_Font: IFontDisp; safecall;
```

```
    function Get_HelpFile: WideString; safecall;
    function Get_KeyPreview: WordBool; safecall;
    function Get_PixelsPerInch: Integer; safecall;
    function Get_PrintScale: TxPrintScale; safecall;
    function Get_Scaled: WordBool; safecall;
    function Get_ScreenSnap: WordBool; safecall;
    function Get_SnapBuffer: Integer; safecall;
    function Get_Visible: WordBool; safecall;
    function Get_VisibleDockClientCount: Integer; safecall;
    procedure _Set_Font(var Value: IFontDisp); safecall;
    procedure Set_AutoScroll(Value: WordBool); safecall;
    procedure Set_AutoSize(Value: WordBool); safecall;
    procedure Set_AxBorderStyle(Value: TxActiveFormBorderStyle); safecall;
    procedure Set_Caption(const Value: WideString); safecall;
    procedure Set_Color(Value: OLE_COLOR); safecall;
    procedure Set_DoubleBuffered(Value: WordBool); safecall;
    procedure Set_DropTarget(Value: WordBool); safecall;
    procedure Set_Enabled(Value: WordBool); safecall;
    procedure Set_Font(const Value: IFontDisp); safecall;
    procedure Set_HelpFile(const Value: WideString); safecall;
    procedure Set_KeyPreview(Value: WordBool); safecall;
    procedure Set_PixelsPerInch(Value: Integer); safecall;
    procedure Set_PrintScale(Value: TxPrintScale); safecall;
    procedure Set_Scaled(Value: WordBool); safecall;
    procedure Set_ScreenSnap(Value: WordBool); safecall;
    procedure Set_SnapBuffer(Value: Integer); safecall;
    procedure Set_Visible(Value: WordBool); safecall;
    procedure Start(FileName: PChar); safecall;
    procedure Stop; safecall;
  public
    { Public declarations }
    procedure Initialize; override;
  end;

implementation

uses ComObj, ComServ;

{$R *.DFM}

{ TVText }

procedure TVText.DefinePropertyPages(DefinePropertyPage: TDefinePropertyPage);
begin
  { Define property pages here.  Property pages are defined by calling
    DefinePropertyPage with the class id of the page.  For example,
      DefinePropertyPage(Class_VTextPage); }
end;

procedure TVText.EventSinkChanged(const EventSink: IUnknown);
begin
  FEvents := EventSink as IVTextEvents;
  inherited EventSinkChanged(EventSink);
end;

procedure TVText.Initialize;
begin
  inherited Initialize;
  OnActivate := ActivateEvent;
```

```
  OnClick := ClickEvent;
  OnCreate := CreateEvent;
  OnDblClick := DblClickEvent;
  OnDeactivate := DeactivateEvent;
  OnDestroy := DestroyEvent;
  OnKeyPress := KeyPressEvent;
  OnPaint := PaintEvent;
end;

function TVText.Get_Active: WordBool;
begin
  Result := Active;
end;

function TVText.Get_AlignDisabled: WordBool;
begin
  Result := AlignDisabled;
end;

function TVText.Get_AutoScroll: WordBool;
begin
  Result := AutoScroll;
end;

function TVText.Get_AutoSize: WordBool;
begin
  Result := AutoSize;
end;

function TVText.Get_AxBorderStyle: TxActiveFormBorderStyle;
begin
  Result := Ord(AxBorderStyle);
end;

function TVText.Get_Caption: WideString;
begin
  Result := WideString(Caption);
end;

function TVText.Get_Color: OLE_COLOR;
begin
  Result := OLE_COLOR(Color);
end;

function TVText.Get_DoubleBuffered: WordBool;
begin
  Result := DoubleBuffered;
end;

function TVText.Get_DropTarget: WordBool;
begin
  Result := DropTarget;
end;

function TVText.Get_Enabled: WordBool;
begin
  Result := Enabled;
end;
```

```
function TVText.Get_Font: IFontDisp;
begin
  GetOleFont(Font, Result);
end;

function TVText.Get_HelpFile: WideString;
begin
  Result := WideString(HelpFile);
end;

function TVText.Get_KeyPreview: WordBool;
begin
  Result := KeyPreview;
end;

function TVText.Get_PixelsPerInch: Integer;
begin
  Result := PixelsPerInch;
end;

function TVText.Get_PrintScale: TxPrintScale;
begin
  Result := Ord(PrintScale);
end;

function TVText.Get_Scaled: WordBool;
begin
  Result := Scaled;
end;

function TVText.Get_ScreenSnap: WordBool;
begin
  Result := ScreenSnap;
end;

function TVText.Get_SnapBuffer: Integer;
begin
  Result := SnapBuffer;
end;

function TVText.Get_Visible: WordBool;
begin
  Result := Visible;
end;

function TVText.Get_VisibleDockClientCount: Integer;
begin
  Result := VisibleDockClientCount;
end;

procedure TVText._Set_Font(var Value: IFontDisp);
begin
  SetOleFont(Font, Value);
end;

procedure TVText.ActivateEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnActivate;
end;
```

```
procedure TVText.ClickEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnClick;
end;

procedure TVText.CreateEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnCreate;
end;

procedure TVText.DblClickEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnDblClick;
end;

procedure TVText.DeactivateEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnDeactivate;
end;

procedure TVText.DestroyEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnDestroy;
end;

procedure TVText.KeyPressEvent(Sender: TObject; var Key: Char);
var
  TempKey: Smallint;
begin
  TempKey := Smallint(Key);
  if FEvents <> nil then FEvents.OnKeyPress(TempKey);
  Key := Char(TempKey);
end;

procedure TVText.PaintEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnPaint;
end;

procedure TVText.Set_AutoScroll(Value: WordBool);
begin
  AutoScroll := Value;
end;

procedure TVText.Set_AutoSize(Value: WordBool);
begin
  AutoSize := Value;
end;

procedure TVText.Set_AxBorderStyle(Value: TxActiveFormBorderStyle);
begin
  AxBorderStyle := TActiveFormBorderStyle(Value);
end;

procedure TVText.Set_Caption(const Value: WideString);
begin
  Caption := TCaption(Value);
end;
```

```
procedure TVText.Set_Color(Value: OLE_COLOR);
begin
  Color := TColor(Value);
end;

procedure TVText.Set_DoubleBuffered(Value: WordBool);
begin
  DoubleBuffered := Value;
end;

procedure TVText.Set_DropTarget(Value: WordBool);
begin
  DropTarget := Value;
end;

procedure TVText.Set_Enabled(Value: WordBool);
begin
  Enabled := Value;
end;

procedure TVText.Set_Font(const Value: IFontDisp);
begin
  SetOleFont(Font, Value);
end;

procedure TVText.Set_HelpFile(const Value: WideString);
begin
  HelpFile := String(Value);
end;

procedure TVText.Set_KeyPreview(Value: WordBool);
begin
  KeyPreview := Value;
end;

procedure TVText.Set_PixelsPerInch(Value: Integer);
begin
  PixelsPerInch := Value;
end;

procedure TVText.Set_PrintScale(Value: TxPrintScale);
begin
  PrintScale := TPrintScale(Value);
end;

procedure TVText.Set_Scaled(Value: WordBool);
begin
  Scaled := Value;
end;

procedure TVText.Set_ScreenSnap(Value: WordBool);
begin
  ScreenSnap := Value;
end;

procedure TVText.Set_SnapBuffer(Value: Integer);
begin
  SnapBuffer := Value;
end;
```

```
procedure TVText.Set_Visible(Value: WordBool);
begin
  Visible := Value;
end;

procedure TVText.Start(FileName: PChar);
var
  Counter : Integer; // licznik - ile już napisów zostało wyświetlonych
  FPause : Integer; // czas wyświetlania napisu
  CurrentTime : TTime; // czas odtwarzania filmu
  wHour, wMin, wSec : Integer;
begin
  FLines := TStringList.Create;
  FLines.LoadFromFile(FileName); // załaduj plik tekstowy

  PrepareText; // przygotuj dwie tablice
  FBroken := False;

  Counter := -1;
  FPause := 0;
  wHour := 0;
  wMin := 0;
  wSec := 0;

  { pętla wyświetlana co 1000 milisekund dopóki zmienna FBroken nie ma wartości False
}
  while (not FBroken) or (not Application.Terminated) do
  begin
    Application.ProcessMessages;
    Sleep(1000);  // odczekaj 1 sek.

    if FBroken then Break; // jeżeli zmienna = TRUE, przerwij działanie

    Inc(wSec); // zwiększ liczbę sekund
    if wSec >= 60 then // jeżeli liczba większa od 60...
    begin
      Inc(wMin); // zwiększ liczbę min.
      wSec := 0; // wyzeruj zmienną
    end;

    if wMin > 60 then // jeżeli liczba min > 60
    begin
      Inc(wHour); // zwiększ liczbę godzin
      wMin := 0; // wyzeruj minuty
    end;

    // na podstawie danych utwórz zmienną TTime
    CurrentTime := EncodeTime(wHour, wMin, wSec, 0);
    lblCurrent.Caption := TimeToStr(CurrentTime);


    if AnsiMatchStr(TimeToStr(CurrentTime), FTime) then
    begin
      Inc(Counter);
      pnlMessage.Caption := FText[Counter];
      FPause := 0;
    end else
    begin
      if Length(pnlMessage.Caption) > 0 then
      begin
        Inc(FPause);
```

```
        if FPause = 5 then
        begin
          FPause := 0;
          pnlMessage.Caption := '';
        end;
      end;
    end;
  end;
end;

procedure TVText.Stop;
begin
  FBroken := True;
end;

procedure TVText.PrepareText;
var
  i : Integer;
begin
{ określ wielkość tablicy na podstawie liczby linii }
  SetLength(FTime, FLines.Count);
  SetLength(FText, FLines.Count);

  { pętla po wszystkich liniach... }
  for I := 0 to FLines.Count -1 do
  begin
  { do tego elementu tablicy przypisz czas, w którym powinien wyświetlić się napis }
    FTime[i] := (Copy(FLines[i], 1, 8));
  { tutaj przypisz samą treść }
    FText[i] := (Copy(FLines[i], 10, Length(FLines[i]) - 8));
  end;
end;

initialization
  TActiveFormFactory.Create(
    ComServer,
    TActiveFormControl,
    TVText,
    Class_VText,
    1,
    '',
    OLEMISC_SIMPLEFRAME or OLEMISC_ACTSLIKELABEL,
    tmApartment);
end.
```

Budowa, rejestracja i instalacja kontrolki

Nasza kontrolka jest już gotowa. Poprzez wybranie pozycji *Build* z menu *Project* kod źródłowy zostanie skompilowany do postaci pliku *.ocx*. Jeżeli projekt zawiera jakieś błędy — informacja o tym pojawi się w oknie (ang. *Message View*).

Poprzez wybranie pozycji *Register ActiveX Server* z menu *Run* kontrolka zostanie zarejestrowana w systemie.

Ćwiczenie 3.12.

Instalacja kontrolki w palecie komponentów.

1. Zamknij projekt — wybierz z menu *File* pozycję *Close All*.
2. Teraz z menu *Component* wybierz *Import ActiveX Control*. Pokaże się okno z rysunku 3.7.
3. Naciśnij przycisk `Add` i odszukaj skompilowaną kontrolkę. Po tym zabiegu kontrolka zostanie dodana do listy.
4. Po naciśnięciu przycisku `Install` powtórzy się proces z ćwiczenia 3.7.

Wykorzystanie kontrolki TVText

Kontrolka została umieszczona na wybranej przez Ciebie palecie. Możesz utworzyć nowy projekt i umieścić ową kontrolkę na formularzu.

Ćwiczenie 3.13.

Obsługa kontrolki.

Oprócz samej kontrolki na formularzu umieść także dwa komponenty `TButton` oraz `TOpenDialog`. Jeden przycisk będzie służył do rozpoczęcia odtwarzania, a drugi — do jego zatrzymania. Cały listing programu przedstawiono na wydruku 3.5.

Wydruk 3.5. *Program wykorzystujący kontrolkę TVText*

```
{
  Copyright (c) 2002 by Adam Boduch <adam@4programmes.net>
}

unit MainFrm;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, OleCtrls, VTextProj_TLB, StdCtrls;

type
  TMainForm = class(TForm)
    VText: TVText;
    btnLoad: TButton;
    OpenDialog: TOpenDialog;
    btnStop: TButton;
    procedure btnLoadClick(Sender: TObject);
    procedure btnStopClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  MainForm: TMainForm;

implementation

{$R *.dfm}

procedure TMainForm.btnLoadClick(Sender: TObject);
begin
  if OpenDialog.Execute then
  begin
    VText.Stop; // zatrzymanie, jeżeli teraz coś się odtwarza...
    btnStop.Enabled := True;
    VText.Start(PChar(OpenDialog.FileName)); // wywołanie procedury
  end;
end;

procedure TMainForm.btnStopClick(Sender: TObject);
begin
  VText.Stop;
  btnStop.Enabled := False;
end;

procedure TMainForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  VText.Stop; // zatrzymanie, jeżeli użytkownik chce wyjść z programu
end;

end.
```

Zauważ, że w kodzie znajduje się zdarzenie `OnClose` formularza. Zdarzenie to występuje w przypadku, gdy użytkownik zdecyduje się na zamknięcie programu. Nim to nastąpi, należy wywołać metodę `Stop`, aby zatrzymała odtwarzanie napisów. Rysunek 3.12 przedstawia program w działaniu. Oczywiście, Ty możesz „dorobić" do tego programu możliwość wyświetlania samego filmu (można zrealizować to przy użyciu komponentu `TMediaPlayer`).

Rysunek 3.12. *Program podczas wyświetlania napisów do filmu*

Publikowanie ActiveX w Internecie

Jak już wspomniałem na początku tego rozdziału, istnieje możliwość umieszczenia własnej kontrolki ActiveX na swojej stronie WWW.

Ćwiczenie 3.14.

Tworzenie przykładowej kontrolki.

Istotą tego podrozdziału jest zaprezentowanie możliwości publikowania własnych kontrolek. Z tego też względu obiekt, który teraz stworzymy będzie bardzo prosty. Utwórz nową kontrolkę, nazwij ją `ActiveWWW`. W polu *Implementation Unit* wpisz `ActiveWWWFrm.pas`, a w polu *Project Name*: `ActiveWWWProj.dpr`. Nasza kontrolka będzie w pętli wyświetlać napis, dając przy tym efekt maszyny do pisania, czyli litera po literze. Na formularzu umieść komponent `TGroupBox`, a na nim `TLabel`. Rozciągnij etykietę na całą szerokość komponentu `TGroupBox` i zmień właściwość `AutoSize` na `False`.

Jeżeli chodzi o kod, to umieść dodatkowo dwa przyciski. Jeden będzie służył do rozpoczęcia animacji, a drugi do jej zatrzymania.

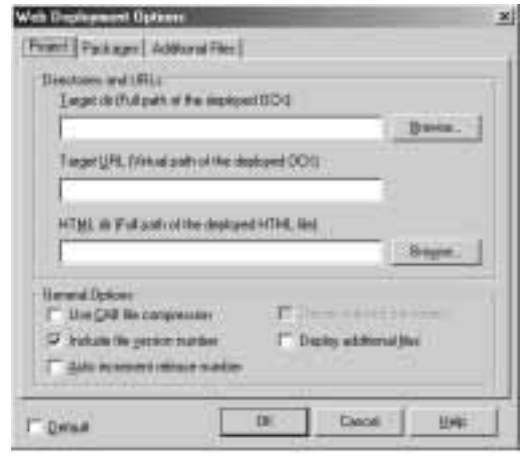
```
var FBroken : Boolean;

procedure TActiveWWW.btnGoClick(Sender: TObject);
const
  ExMsg = 'To jest przykładowa kontrolka :-)';
var
  i : Integer;
begin
  FBroken := False;
{ pętla while wykonywana przez cały czas trwania programu }
  while (not Application.Terminated) or (not FBroken) do
  begin
    lblMessage.Caption := '';
  { pętla for powoduje wyświetlenie na etykiecie kolejnych liter }
    for i := 1 to Length(ExMsg) do
    begin
      Application.ProcessMessages;
      if FBroken then Break;
      Sleep(100);
      lblMessage.Caption := lblMessage.Caption + ExMsg[i];
    end;
  end;
end;

procedure TActiveWWW.btnStopClick(Sender: TObject);
begin
  FBroken := True;
end;
```

Kod daje efekt maszyny do pisania, wyświetla litera po literze przykładowy napis.

To by było na tyle, jeżeli chodzi o samą kontrolkę. Do opublikowania kontrolki w Sieci posłużymy się dwiema pozycjami z menu *Project*: *Web Deployment Options* oraz *Web Deploy*. Na samym początku wybierz pierwszą pozycję, aby ustalić opcję publikacji (rysunek 3.13).

W oknie tym musimy podać parę informacji, które są potrzebne do zbudowania kontrolki. Załóżmy, że będzie ona uruchamiana na lokalnym serwerze Apache.

Rysunek 3.13. *Okno Web Deployment Options*



Ćwiczenie 3.15.

Publikowanie kontrolki.

Pierwsze pole *Target dir* okna *Web Deployment Options* musi zawierać ścieżkę do katalogu, w którym kontrolka zostanie umieszczona po zbudowaniu. W kolejnym polu, *Target URL*, wpisz adres URL, który będzie prowadził do odpowiedniej strony — ja wpisałem `http://localhost`. Ostatnie pole *HTML Dir* określa ścieżkę, gdzie wygenerowany zostanie odpowiedni plik HTML. Ja wpisałem tę samą wartość, co w pozycji *Target Dir*.

To właściwie wszystko, zamknij okno przyciskiem *OK*. Wybierz z menu *Project* pozycję *Web Deploy*. Kontrolka powinna zostać skompilowana i zapisana w wybranym przez Ciebie katalogu.

W oknie *Web Deployment Options* możesz zaznaczyć opcję *Use CAB file compression*. Dzięki temu kontrolka ActiveX zostanie skompilowana do pliku **.cab*.

Po tym zabiegu plik *ActiveWWWProj.htm* zawiera treść przedstawioną na wydruku 3.6.

Wydruk 3.6. *Kod HTML strony wygenerowanej przez Delphi*

```
<HTML>
<H1> Delphi 7 ActiveX Test Page </H1><p>
You should see your Delphi 7 forms or controls embedded in the form below.
<HR><center><P>
<OBJECT
    classid="clsid:6013039B-7D1B-4DAE-98D2-232733529810"

  codebase="http://localhost/ActiveWWWProj.ocx#version=1,0,0,0"
    width=350
    height=250
    align=center
    hspace=0
    vspace=0
>
</OBJECT>
</HTML>
```

Aby kontrolka ActiveX była lepiej wyświetlana, zmieniłem szerokość i wysokość obiektu na takie wartości:

```
width=400
height=150
```

Rezultat działania programu możesz zobaczyć na rysunku 3.14.

Rysunek 3.14. *Kontrolka ActiveX w działaniu!*

Względy bezpieczeństwa

Używanie kontrolek ActiveX w Internecie nie jest zbyt popularną usługą. Wielu użytkowników ze względu na niebezpieczeństwo wynikające z korzystania z ActiveX ma wyłączoną opcję ich ładowania (zdaje się, że jest to domyślne ustawienie). Z tego względu, chcąc załadować kontrolkę, zobaczą tylko taki komunikat, jak pokazany na rysunku 3.15.

Rysunek 3.15. *Komunikat informujący o niemożności obsłużenia ActiveX*

Ćwiczenia dodatkowe

Ćwiczenie 3.16.

Napisz program do wyświetlania filmów, który będzie korzystać z kontrolki TVText.

Podsumowanie

Co prawda ActiveX nie jest popularną technologią na stronach WWW, ale dość wygodną, jeżeli chodzi o tworzenie obiektów środowiskowych. Pamiętaj, że utworzoną kontrolkę można również uruchomić w innych środowiskach programistycznych systemu Windows.