

Jakub Możaryn

Wstęp do programowania wizualno - obiektowego

Materiały dydaktyczne, Ośrodek Kształcenia Na Odległość – OKNO

Zawartość

1. Instalacja środowiska Turbo Delphi.....	6
2. Tworzenie i kompilowanie projektów w Turbo Delphi.....	7
2.1. Pliki tworzone podczas kompilowania projektów.....	7
2.2. Komponenty podstawowe – część 1.....	8
2.3. Pierwszy program, tworzenie i kompilowanie projektu.....	15
2.4. Drugi program.....	15
2.5. Program 3.....	17
3. Podstawowe komponenty i ich właściwości.....	22
3.1. Pola Wyboru (TCheckBox, TRadioButton, TRadioGroup).....	22
3.2. Listy rozwijalne (TListBox, TComboBox).....	22
3.3. Panele (TPanel).....	23
1.2.1 Menu (TMainMenu, TPopupMenu).....	23
1.2.2 Pasek stanu (TStatusBar).....	24
1.2.3 Elastyczne obszary interfejsu (TSplitter).....	25
1.3 Scentralizowane zarządzanie wywoływaniem i udostępnianiem funkcji.....	28
1.3.1 Lista akcji (TActionList).....	28
3.7. Tabela (TStringGrid).....	30
3.8. Figura geometryczna (TShape).....	31
3.9. Rysunek (TImage).....	31
5. Okna dialogowe.....	32
5.1. Wykorzystanie standardowych okien dialogowych.....	32
5.2. Okno dialogowe informacyjne (informacja o programie).....	32
5.3. Okno dialogowe do pracy z plikami.....	33
6. Wykorzystanie wielu formularzy, przełączanie i wymiana danych pomiędzy formularzami.....	35
7. Program 4.....	35

1. Instalacja środowiska *Turbo Delphi*

Do instalacji środowiska *Turbo Delphi* wymagane jest uzupełnienie instalacji systemu operacyjnego *Microsoft Windows* o następujące elementy

Tabela 1. Pakiety instalacyjne wymagane do uruchomienia środowiska programistycznego *Turbo Delphi*.

pakiet	adres www
<i>Microsoft .NET Framework v1.1 Redistributable</i>	http://www.microsoft.com/downloads/details.aspx?familyid=262D25E3-F589-4842-8157-034D1E7CF3A3&displaylang=en
<i>Microsoft .NET Framework v1.1 SP1</i>	http://www.microsoft.com/downloads/details.aspx?FamilyID=a8f5654f-088e-40b2-bbdb-a83353618b38&displaylang=en
<i>Microsoft .NET SDK v1.1</i>	http://www.microsoft.com/downloads/details.aspx?familyid=9B3A2CA6-3647-4070-9F41-A333C6B9181D&displaylang=en
<i>Microsoft Internet Explorer 6 SP1</i>	http://www.microsoft.com/downloads/details.aspx?displaylang=pl&FamilyID=1e1550cb-5e5d-48f5-b02b-20b602228de6
<i>Microsoft Visual J# v1.1 Redistributable</i>	http://www.microsoft.com/downloads/details.aspx?FamilyID=E3CF70A9-84CA-4FEA-9E7D-D674D2C7CA1&displaylang=en
<i>Microsoft XML Core Services (MSXML) v4.0 SP2</i>	http://www.microsoft.com/downloadS/details.aspx?familyid=3144B72B-B4F2-46DA-B4B6-C5D7485F2B42&displaylang=en

Poszczególne pakiety należy instalować zgodnie z kolejnością podaną w Tabeli 1.

Pakiet instalacyjny **Turbo Delphi** należy ściągnąć ze strony

<https://downloads.embarcadero.com/free/turbo>

Przed ściągnięciem należy zarejestrować się i po ściągnięciu pakietu instalacyjnego zostanie wysłany na podany adres e-mail plik tekstowy z kluczem aktywacyjnym, który należy zapisać w odpowiednim katalogu na swoim komputerze.

W trakcie instalacji należy postępować zgodnie z krokami podanymi przez program instalacyjny.

2. Tworzenie i kompilowanie projektów w Turbo Delphi

2.1. Pliki tworzone podczas kompilowania projektów

Podczas zapisywania projektu aplikacji okienkowej VCL automatycznie tworzone są pliki określonych typów, które zostały zebrane w tabeli 3.1.

Tabela 3.1. *Typy plików używanych w Delphi*

Rozszerzenie	Opis
.pas	Plik źródłowy modułu. Zawiera jego kod źródłowy (każdy moduł ma swój plik źródłowy)
.dfm	Plik formularza. Jest to właściwie zakamuflowany binarny plik zasobów (.res). Zawiera opis formularza i umieszczonych na nim komponentów. Każdy formularz posiada swój własny plik .dfm
.exe	Wynikowy program wykonywalny
.cfg	Plik konfiguracyjny projektu. Zawiera głównie ustawienia kompilatora i konsolidatora
.dcu	Binarne pliki wynikowe kompilatora (skompilowane moduły)
.dpr	Plik źródłowy projektu
.res	Skompilowany binarny plik zasobów

Za każdym razem, kiedy tworzony jest nowy formularz, *Turbo Delphi* wykonuje następujące kroki:

- Tworzy plik formularza (.dfm)
- Tworzy nową klasę – potomek klasy *TForm*
- Tworzy plik źródłowy (.pas) zawierający definicję tej klasy
- Dodaje informacje o nowym formularzu do pliku źródłowego projektu (.dpr)

Nowo utworzonemu formularzowi nadawana jest domyślna nazwa *Form1*, a plikowi źródłowemu odpowiadającego mu modułu nadaje nazwę *Unit1.pas*. Następny formularz utworzony w projekcie będzie miał nazwę *Form2* a związany z nim moduł *Unit2.pas*, itd. Dla każdego nowo utworzonego formularza *Turbo Delphi* tworzy nową parę plików: .pas i .dfm.

Możliwe jest skasowanie części plików z projektów. Na podstawie pozostałych *Turbo Delphi* może w każdej chwili odtworzyć postać całego projektu. Niezbędne są tylko pliki .pas, .dfm, .dpr i .res. Wszystkie pozostałe zostaną odtworzone podczas ponownej kompilacji.

Istotne jest pamiętanie żeby nie kasować żadnych plików z katalogów, w których zainstalowano *Turbo Delphi*, poza katalogiem /EXAMPLES. **Jeżeli masz jakieś wątpliwości – nie kasuj.**

Zaraz po utworzeniu nowego projektu dobrze jest zapisać go na dysk nadając mu jakąś znaczącą nazwę. To samo odnosi się do każdego nowo utworzonego formularza. Ułatwia to późniejszą ich lokalizację i identyfikację. Sprawę ułatwia dodatkowo fakt, że moduły mogą mieć długie nazwy.

Podczas zapisywania projektu warto plik projektu oraz wszystkie inne pliki z nim związane zapisać w jednym katalogu. Generalnie dotyczy to aplikacji pisanych przez jednego programistę. Częstym błędem jest przenoszenie tylko części plików pomiędzy komputerami, ponieważ nie znana jest ich dokładna lokalizacja. Zapisywanie plików w jednym katalogu pozwala zapanować nad miejscem gdzie są przechowywane i ułatwia

pracę. W przypadku pracy grupowej należy ustalić i konsekwentnie przestrzegać, w których katalogach znajdują się pliki.

2.2. Komponenty podstawowe – część 1

2.2.1. Komponenty VCL

Turbo Delphi posiada standardową bibliotekę komponentów VCL (ang. *visual component library*). Komponenty są to klasy reprezentujące kontrolki Windows. Można je podzielić na:

- komponenty widoczne, które wyglądają na formularzu tak jak na ekranie np: etykiety, pola edycyjne, przyciski, listy rozwijalne.
- komponenty niewidoczne, które reprezentują elementy niewidoczne podczas działania programu np: komponenty związane z obsługą zegara systemowego, komponenty bazodanowe lub komponenty komunikacji sieciowej.

Z każdym z omawianych komponentów związane są odpowiednie właściwości, zdarzenia i metody.

2.2.2. Podstawowe właściwości komponentów VCL

Właściwość Name

Turbo Delphi po wstawieniu komponentu na formularz tworzy wskaźnik do komponentu i nadaje mu nazwę określoną we właściwości Name. Dzięki temu wskaźnikowi i nazwie użytkownik ma dostęp do komponentu w trakcie pracy programu. Tworząc procedury i ich nazwy dla danego komponentu *Turbo Delphi* wykorzystuje także własność Name tego komponentu.

Wartość właściwości Name może być modyfikowana w dowolnej chwili, pod warunkiem, że modyfikacja ta odbywa się wyłącznie poprzez Inspektor Obiektów. Należy nadawać dodanym do programu komponentom nazwy znaczące od razu po ich utworzeniu, aby uniknąć zamieszania i niepotrzebnej pracy w przyszłości. W nazwie warto stosować typowe przedrostki odpowiadające nazwie komponentu, np. `edt` – Pole edycyjne (ang. *edit*), `btn` – Przycisk (ang. *button*), `lbl` – Etykieta (ang. *label*).

Właściwość Color

Właściwość `Color` określa kolor tła komponentu.

Sposób obsługiwanienia właściwości `Color` przez Inspektor Obiektów jest nieco unikalny. W momencie kliknięcia na własności `Value` pokazuje się lista rozwijalna informująca o możliwości wyboru kolorów z predefiniowanego zbioru. Podwójne kliknięcie na własności `Value` powoduje że wyświetlone zostanie okno dialogowe *Kolor*. Okno to (rys. 4.1) umożliwia wybór jednego z predefiniowanych kolorów lub utworzenie własnych kolorów.

Nazwy kolorów standardowych zdefiniowanych w środowisku *Turbo Delphi* oznaczone są jako `cl+(nazwa koloru w języku angielskim)` np. `clBlack`. Te nazwy lub ich odpowiedniki numeryczne można stosować podczas pisania procedur.



Rysunek 4.1. Okno dialogowe Kolor

Właściwość Enabled

Właściwość Enabled pozwala na włączanie i wyłączanie dostępu do komponentów. Kiedy komponent jest zablokowany, nie może przyjąć stanu aktywności (klikanie na nim nie daje żadnego efektu) i zazwyczaj informuje o swoim stanie w sposób wizualny np. "przygaszony" przycisk.

Enabled jest właściwością typu Boolean, tak więc wartość True udostępnia dany komponent, podczas gdy wartość False blokuje dostęp do niego.

Właściwość Font

Właściwość Font jest tak naprawdę zmienną klasy TFont i jako taka posiada także charakterystyczne właściwości. Opisuje ona wygląd i formatowanie czcionek związanych z danym komponentem. Klasa TFont posiada on następujące podstawowe własności:

- Name - rodzaj czcionki np: Times New Roman
- Height - wysokość czcionki w pikselach.
- Size - wysokość czcionki w punktach.
- Style - ustawienie pogrubienia (fsBold:Boolean), pochylenia (fsItalic:Boolean), podkreślenia (fsUnderline:Boolean) lub przekreślenia czcionki (fsStrikeOut:Boolean).

Właściwość Hint

Właściwość ta służy do określenia tekstu tzw. pomocy kontekstowej dla komponentu. Tekst ten dzieli się na dwie części.

- *krótka pomoc kontekstowa* - Jest to tekst wyświetlany w chwili gdy użytkownik umieści kursor nad komponentem i zatrzyma go przez chwilę w tej pozycji.
- *długa pomoc kontekstowa* - Stanowi opcjonalny tekst wyświetlany na pasku stanu (komponent TStatusBar z palety Win32), gdy użytkownik przemieści kursor nad określony komponent.

Długą pomoc kontekstową oddziela się od krótkiej pomocy kontekstowej poziomą linią, *Krótką Pomoc Kontekstową | Długą Pomoc Kontekstową*

Podpowiedź kontekstowa będzie pokazywała się tylko wtedy kiedy ustawiona zostanie własność ShowHint komponentu na wartość True.

Tabela 1. *Spis wybranych dodatkowych właściwości komponentów VCL w Turbo Delphi*

Właściwość	Opis
BorderStyle	Określa rodzaj obramowania komponentu (wartość bsSingle lub bsNone).
BoundsRect	Prostokątny obszar całego komponentu (nie ograniczony jedynie do obszaru klienta).
Caption	Określa napis znajdujący się na komponencie (np. przycisk, panel).
ClientHeight	Zawiera wysokość obszaru klienta w komponencie.
ClientRect	Zawiera współrzędne prostokątnego obszaru klienta w komponencie.
ClientWidth	Zawiera szerokość obszaru klienta w komponencie.
Constraints	Określa wymiary ograniczające komponent (maksymalną i minimalną szerokość i wysokość). Właściwość ta wykorzystywana jest przy projektowaniu formularzy.
Ctl3D	Określa trójwymiarowe obramowanie komponentu.
Height	Określa wysokość komponentu.
HelpContext	Skojarzenie komponentu z numeru indeksu w pliku pomocy.
Left	Określa współrzędną x komponentu względem górnego lewego rogu formularza.
Parent	Wskaźnik do rodzica komponentu.
PopupMenu	Określa tzw. menu kontekstowe (komponent klasy TPopupMenu) wyświetlane w chwili, gdy użytkownik kliknie prawym przyciskiem myszki znajdując się nad danym komponentem.
TabOrder	Określa kolejność 'podświetlania' i wyboru komponentu znajdującego się na formularzu po kliknięciu klawisza Tab.
Top	Określa współrzędną y komponentu względem górnego lewego rogu formularza.
Visible	Określa czy komponent jest aktualnie widoczny na formularzu. Jest typu Boolean, wybór wartości False objawia się ukryciem komponentu zaś wybór wartości True jego wyświetleniem.
Width	Określa szerokość komponentu.
Tag	4-bajtowa zmienna zarezerwowana na potrzeby programisty

2.2.3. Podstawowe metody komponentów VCL

Metoda Refresh

Wymusza natychmiastowe przerysowanie komponentu i czyści go tuż przed ponownym namalowaniem.

Metoda Repaint

Wymusza natychmiastowe przerysowanie komponentu. Tło komponentu nie jest czyszczone przed ponownym namalowaniem.

Metoda Hide

Ukrywa komponent. Komponent jest nadal dostępny i może być ponownie wyświetlony w czasie późniejszym.

Tabela 2. *Spis wybranych metod komponentów VCL w Turbo Delphi*

Metoda	Opis
Broadcast	Wysyła komunikat do wszystkich potomnych komponentów.
ClientToScreen	Konwertuje współrzędne okna użytkownika (klienta) do współrzędnych ekranowych.
ContainsControl	Zwraca wartość True, jeżeli określony komponent jest potomkiem danego komponentu lub formularza.
HandleAllocated	Zwraca wartość True, jeżeli dla komponentu utworzony został uchwyt (Handle). Nie jest wymagane wcześniejsze utworzenie uchwytu.
Invalidate	Wymusza przerysowanie komponentu (jego odświeżenie) w najbliższym dogodnym dla systemu czasie.
Perform	Przekazuje komunikat bezpośrednio do komponentu.
SetBounds	Umożliwia jednoczesne ustawienie właściwości Top, Left, Width i Height. Operacja taka oszczędza czas w porównaniu z ręcznym ustawianiem każdej właściwości z osobna.
SetFocus	Czyni komponent aktywnym. Działa jedynie w przypadku komponentów typu okienkowego.
Update	Wymusza natychmiastowe odświeżenie komponentu. W celu odświeżenia komponentów należy stosować metody Refresh lub Repaint.

2.2.4. Podstawowe zdarzenia komponentów VCL

Zdarezeniem jest zapis zajścia określonej sytuacji w systemie komputerowym np. odświeżenie ekranu, ruch kursora, kliknięcie klawiszem myszy. Zazwyczaj z komponentami VCL w *Turbo Delphi* związane są określone zdarzenia. Wybrane zdarzenia zebrano w tabeli 3.

Tabela 3. *Spis wybranych zdarzeń komponentów VCL w Turbo Delphi*

Zdarzenie	Opis
OnChange	Zdarzenie generowane, gdy w komponencie zajdzie jakakolwiek zmiana.
OnClick	Zdarzenie generowane, gdy użytkownik kliknie dowolnym przyciskiem myszy na obszarze komponentu.
OnDblClick	Zdarzenie generowane gdy użytkownik kliknie podwójnie na komponencie.
OnEnter	Zdarzenie generowane gdy widoczny komponent (np. pole edycyjne) stanie się aktywny.
OnExit	Zdarzenie generowane gdy widoczny komponent przestanie być aktywny, co zostanie wymuszone przez użytkownika. Zdarzenie to nie pojawia się kiedy użytkownik przełącza się między formularzami lub przechodzi do innej aplikacji.
OnKeyDown	Zdarzenie generowane po naciśnięciu klawisza przez użytkownika w chwili gdy kontrolka posiada stan aktywności.
OnKeyPress	Zdarzenie generowane po naciśnięciu przez użytkownika jednego z klawiszy alfanumerycznych, Tab, Backspace, Enter lub Esc
OnKeyUp	Zdarzenie generowane się za każdym razem, gdy zwolniony zostanie naciśnięty klawisz.
OnMouseDown	Zdarzenie generowane jest w chwili gdy kursor znajdzie się nad komponentem i naciśnięty zostanie jeden z przycisków myszy. Parametry przekazywane do funkcji obsługującej to zdarzenie informują o tym który z przycisków myszy został kliknięty, które z klawiszy specjalnych (Alt, Shift, Ctrl) zostały naciśnięte, a także jakie były współrzędne położenia x i y końca kursora.
OnMouseMove	Zdarzenie generowane gdy mysz przemieszcza się nad komponentem.
OnMouseUp	Zdarzenie generowane gdy przycisk myszy zostanie zwolniony nad komponentem. Przycisk musi wcześniej zostać wciśnięty, gdy kursor znajduje się nad komponentem.
OnPaint	Zdarzenie generowane gdy komponent wymaga odświeżenia (ponownego narysowania).

2.2.4 Podstawowe komponenty i ich właściwości

2.2.4.1 Etykieta (*TLabel*)

Komponent etykieta - *Label* służy do wyświetlania tekstu w formularzu. Czasami tekst ten jest określany w fazie projektowania i później nie ulega już zmianom. W innych przypadkach, etykieta zachowuje się w sposób dynamiczny, ulegając zmianom w trakcie pracy, w sposób zależny od przebiegu programu. W przypadku kiedy etykieta zachowuje się w sposób dynamiczny warto nadać jej własną nazwę (własność *Name*).

Do zmiany tekstu etykiety w trakcie pracy programu należy korzystać z właściwości *Caption*.

Przydatne właściwości etykiety

<i>Caption</i>	- tekst wyświetlany na etykiecie
<i>Font</i>	- właściwości czcionki tekstu etykiety
<i>Visible</i>	- określa czy etykieta jest widoczna, czy niewidoczna na formularzu
<i>AutoSize</i>	- Własność typu <i>Boolean</i> . Jeśli jest ustawiona na wartość <i>True</i> etykieta automatycznie zmienia swój rozmiar dopasowując się do tekstu. Jeżeli wartością jest <i>False</i> , tekst wystający poza prawą krawędź jest obcinany.

2.2.4.2 Pola Edycyjne (*TEdit*, *TMemo*)

Komponent pola edycyjnego - *Edit* realizuje prostą kontrolkę edycji jednowierszowej, natomiast komponent *Memo* realizuje prostą kontrolkę edycji wielowierszowej.

Przydatne właściwości pola edycyjnego

(*TEdit*)

Rozmiar	- <i>Width</i> - szerokość komponentu, - <i>Height</i> - wysokość komponentu
<i>Text</i>	- Przechowuje tekst stanowiący zawartość komponentu.

(*TMemo*)

<i>Lines</i>	- Przechowuje tekst stanowiący zawartość komponentu, klasa <i>TStrings</i> .
<i>ScrollBars</i>	- Określa, które paski przewijania powinny być wyświetlane.

Przydatne metody pola edycyjnego

(*Lines* - klasa *TStrings*)

Add (*const AnsiString FileName*) – dodawanie nowej linii do pola tekstowego.

Np. Memo1.Lines.Add('jakiś tekst');

LoadFromFile (*const AnsiString FileName*) – wczytywanie danych z pliku tekstowego o nazwie *FileName*. *Np. Memo1.Lines.LoadFromFile('plik.txt');*

SaveToFile (*const AnsiString FileName*) – wczytywanie danych do pliku tekstowego o nazwie *FileName*, jeśli plik nie istnieje to zostanie utworzony, jeśli plik istnieje to informacje wcześniej w nim zapisane zostaną utracone.

Przydatne zdarzenia pola edycyjnego

<i>OnChange</i>	- zmiana zawartości pola edycyjnego
<i>OnEnter</i>	- znalezienie się w polu edycyjnym, uaktywnienie pola edycyjnego
<i>OnExit</i>	- opuszczenie pola edycyjnego

2.2.4.3. Przyciski (*TButton*)

Komponent przycisk – `Button` jest standardowym przyciskiem systemu Windows.

Przydatne własności przycisku

`Enabled` - określa czy przycisk jest aktywny czy nieaktywny (`true/false`)
`Caption` - jest to tekst wyświetlany na przycisku

Przydatne zdarzenia przycisku

`OnClick` - wciśnięcie przycisku

2.2.4.4. Formularz (*TForm*)

Komponent `Form` jest standardowym formularzem (oknem) systemu Windows, na którym znajdują się poszczególne komponenty aplikacji. Jedna aplikacja może mieć wiele formularzy.

Przydatne właściwości formularza

`Caption` - Opis formularza (tytuł), widoczny na ekranie po uruchomieniu.

Przydatne metody formularza

`Close` - Zamknięcie formularza.

Przydatne zdarzenia formularza

`OnCreate` - Utworzenie formularza;

2.3 Pierwszy program, tworzenie i kompilowanie projektu

Jako pierwszy program zostanie przedstawiona aplikacja, która będzie wypisywała na etykiecie tekst wpisywany w pole edycyjne po naciśnięciu przycisku.

Krok 1.

Otwieramy nową aplikację *File>New Project>VCL Forms Application*

Krok 2.

Zapisujemy projekt przy pomocy *File>Save All* w katalogu gdzie będzie kompilowany nasz projekt

Krok 3.

Tworzymy interfejs. Dodajemy odpowiednie komponenty w kolejności

1 pole edycyjne **TEdit** (paleta Standard) o nazwie (własność Name):

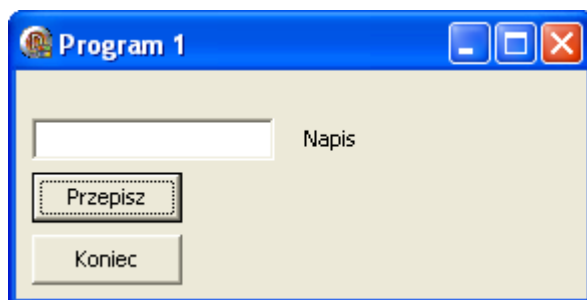
- edtTekst

1 etykieta **TLabel** (paleta Standard) o nazwie:

- lblNapis

2 przyciski **TButton** (paleta Standard) o nazwach::

- btnPrzepisz
- btnKoniec



Rysunek 1. Postać interfejsu użytkownika

Zmieniamy nazwę programu (komponent Form ,własność Caption: *Program 1*)

Krok 4.

Piszemy odpowiednie procedury - obsługa zdarzenia *OnClick* przycisków

Przepisywanie wartości pomiędzy polem edycyjnym a etykietą (przycisk btnPrzepisz)

```
procedure TForm1.btnPrzepiszClick(Sender: TObject);  
begin  
    self.lblNapis.Caption:=self.edtTekst.Text;  
end;
```

Zamknięcie aplikacji (przycisk btnKoniec)

```
procedure TForm1.btnKoniecClick(Sender: TObject);  
begin  
    self.Close;  
end;
```

2.4 Drugi program

Kolejnym zadaniem będzie napisanie aplikacji, która będzie zmieniała kolor napisu na etykiecie w zależności od przyciśniętego przycisku. Umożliwić wybór pomiędzy trzema kolorami (zielonym, żółtym i czerwonym).

Krok 1.

Otwieramy nową aplikację *File>NewApplication*

Krok 2.

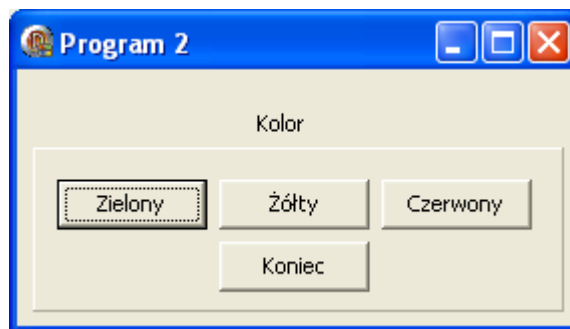
Zapisujemy projekt przy pomocy *File>Save All* w katalogu gdzie będzie kompilowany nasz projekt

Krok 3.

Projektujemy interfejs użytkownika

Dodajemy odpowiednie komponenty w kolejności

- 1 etykieta **TLabel** (paleta Standard) o nazwie LblNapis
- 1 panel **TPanel** (paleta Standard)
- 4 przyciski **TButton** (paleta Standard) o nazwach::
 - BtnZielony
 - BtnZolty
 - BtnCzerwony
 - BtnKoniec



Rysunek 2.2 Postać interfejsu użytkownika dla programu 2.

Krok 4.

Umieszczamy przyciski, które mają zmieniać kolor napisu na panelu. Nie daje się tego zrobić na samym formularzu, należy odpowiednio je przesunąć (sposób analogiczny do przeciągania plików do katalogów) w drzewku w oknie Structure. Komponent TPanel służy do grupowania komponentów.

Krok 5.

Piszemy odpowiednie procedury (obsługa zdarzenia *OnClick* przycisków)

Zamiana kolorów (przyciski *btnZielony*, *btnZolty*, *btnCzerwony*)

Przycisk *btnZielony* -> zdarzenie *OnClick*

```
procedure TForm1.btnZielonyClick(Sender: TObject);
begin
    self.lblNapis.Font.Color:=clGreen;
end;
```

Przycisk *btnŻółty* -> zdarzenie *OnClick*

```
procedure TForm1.btnZoltyClick(Sender: TObject);
begin
    self.lblNapis.Font.Color:=clYellow;
end;
```

Przycisk *btnCzerwony* -> zdarzenie *OnClick*

```
procedure TForm1.btnCzerwonyClick(Sender: TObject);
begin
    self.lblNapis.Font.Color:=clRed;
end;
```

Zamknięcie aplikacji (przycisk *KoniecBtn* -> zdarzenie *OnClick*)

```
procedure TForm1.KoniecClick(Sender: TObject);
begin
    self.Close;
end;
```

2.5. Program 3

Kolejnym zadaniem jest napisanie aplikacji, która będzie przechowywała w tablicy współrzędne n punktów (np. $n=10$). W aplikacji umożliwić użytkownikowi dodawanie nowych punktów, usuwanie punktów, zmianę wartości punktów oraz przeglądanie współrzędnych punktów. Informacje o wpisanych punktach należy wypisywać na etykietach. Umożliwić wyjście z programu po przyciśnięciu przycisku.

Krok 1.

Otwieramy nową aplikację *File>NewApplication*

Krok 2.

Zapisujemy projekt przy pomocy *File>Save All* w katalogu gdzie będzie kompilowany nasz projekt

Krok 3.

Tworzymy interfejs

Dodajemy odpowiednie komponenty w kolejności

2 pola edycyjne **TEdit** (paleta Standard) o nazwach (własność Name):

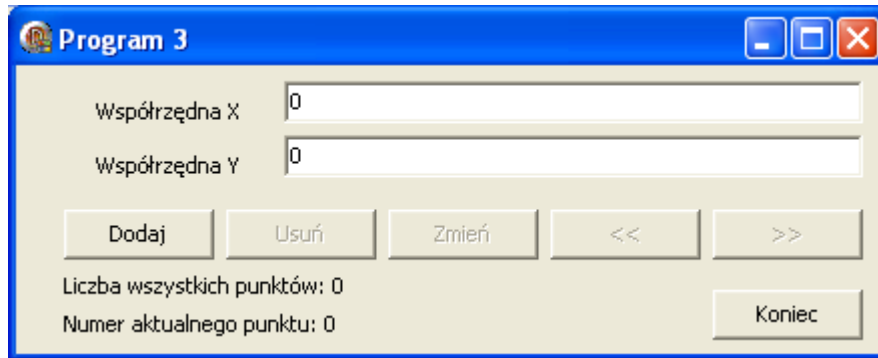
- edtWspX
- edtWspY

4 etykiety **TLabel** (paleta Standard) o nazwach (własność Name , tylko etykiety dynamiczne):

- lblAktualny
- lblOstatni

6 przycisków **TButton** (paleta Standard) o nazwach (własność Name):

- btnDodaj
- btnUsun
- btnZmien
- btnPoprzedni
- btnNastepny
- btnKoniec



Rysunek 2.3 Postać interfejsu użytkownika

Deklarujemy zmienne przechowujące informacje o dodanych punktach. W tym celu musimy dokonać edycji pliku .pas, związanego z naszym formularzem (Form1 -> Unit1.pas). Znajduje się tam definicja klasy opisującej projektowany formularz.

W części publicznej klasy TForm1 (public) wpisujemy deklarację zmiennej tablicowej przechowującej współrzędne punktów.

```
Tablica:array[1..MAX_TAB] of    record
                                x,y:real;
                                end;
```

Dodatkowo deklarujemy dwie zmienne typu całkowitego w których przechowujemy informacje o tym ile wpisano punktów (wszystkie), oraz o tym który aktualnie punkt jest edytowany (aktualny).

```
wszystkie:integer;
aktualny:integer;
```

Na początku musimy jeszcze określić maksymalny rozmiar tablicy MAX_TAB, jako stałą wartość typu całkowitego.

Ponieważ użytkownik może nie wiedzieć jak obsługiwać program warto udostępnić mu podpowiedzi wyświetlające się na najważniejszych komponentach. W przypadku projektowanej aplikacji wpisuje się odpowiednie teksty we własność Hint następujących komponentów:

Pola edycyjne

- edtWspX - 'Wpisz zmienną x punktu, wartość typu całkowitego'
- edtWspY - 'Wpisz zmienną y punktu, wartość typu całkowitego'

Przyciski

- btnDodaj - 'Naciśnij żeby dodać punkt'
- btnUsun - 'Naciśnij żeby usunąć punkt'
- btnZmien - 'Naciśnij żeby zmienić współrzędne aktualnego punktu'
- btnPoprzedni - 'Pokaż współrzędne poprzedniego punktu'
- btnNastepny - 'Pokaż współrzędne następnego punktu'
- btnKoniec - 'Wyście z programu'

Żeby ustawione podpowiedzi pokazywały się na ekranie, należy ustawić własność `ShowHint` każdego z tych komponentów na `True`.

Krok 4.

Piszemy odpowiednie procedury – obsługę kolejnych zdarzeń

Ponieważ zmienna wpisywana w pole edycyjne jest łańcuchem znaków (`string`) przydadzą się następujące funkcje do konwersji pomiędzy typami danych (funkcje te znajdują się w module `SysUtils`):

```
function FloatToStr(Value: Extended): string; - Dokonuje konwersji zmiennej typu rzeczywistego (zmiennoprzecinkowego) na łańcuch znaków.
```

```
function StrToFloat(const S: string): Extended; - Dokonuje konwersji łańcucha znaków na zmienną typu rzeczywistego (zmiennoprzecinkowego).
```

```
function IntToStr(Value: Int64): string; overload; - Dokonuje konwersji zmiennej typu całkowitego na łańcuch znaków.
```

```
function StrToInt(const S: string): Integer; - Dokonuje konwersji łańcucha znaków na zmienną typu całkowitego.
```

Inicjacja zmiennych

Inicjacja zmiennych powinna zostać zrealizowana podczas tworzenia Formularza (wywoływania zdarzenia `OnCreate`).

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  //inicjacja zmiennych
  self.el_ostatni:=0;
  self.el_aktualny:=0;
  //ustawienie tekstów w polu edycyjnym
  self.edtWspX.Text:='0';
  self.edtWspY.Text:='0';
  //zablokowanie wybranych przycisków
  self.btnUsun.Enabled:=False;
  self.btnZmien.Enabled:=False;
  self.btnPoprzedni.Enabled:=False;
  self.btnNastepny.Enabled:=False;
end;
```

Wyjście z programu

```
procedure TForm1.btnKoniecClick(Sender: TObject);
begin
  //zamknięcie aplikacji
  self.Close;
end;
```

Dodawanie elementów

```
procedure TForm1.btnDodajClick(Sender: TObject);
begin
  //sprawdzenie warunku przekroczenia rozmiaru tablicy
  if (self.el_ostatni<MAX_TAB) then
  begin
    self.Tablica[self.el_ostatni+1].x:=StrToFloat(self.edtWspX.Text);
    self.Tablica[self.el_ostatni+1].y:=StrToFloat(self.edtWspY.Text);
    self.el_ostatni:=self.el_ostatni+1;
    self.el_aktualny:=self.el_ostatni;
  end
  else
  begin
```

```

//blokowanie przycisku dodaj w przypadku przekroczenia maksymalnej liczby elementów
self.btnDodaj.Enabled:=False;
//wypisanie informacji o błędzie w postaci okna dialogowego (Message Box)
Application.MessageBox('tablica jest pełna','Error',MB_OK);
end;
//sprawdzenie warunku dodania pierwszego elementu i odblokowanie przyciskow
if (self.el_ostatni<2) then
begin
//odblokowanie przycisków jeśli wpisano pierwszy element element
self.btnUsun.Enabled:=True;
self.btnZmien.Enabled:=True;
self.btnPoprzedni.Enabled:=True;
self.btnNastepny.Enabled:=True;
end;
//wypisanie informacji o ilosci elementów i aktualnym punkcie
self.lblOstatni.Caption:='Liczba wszystkich punktów: '+intToStr(self.el_ostatni);
self.lblAktualny.Caption:='Numer aktualnego punktu: '+intToStr(self.el_aktualny);
end;

```

Przeglądanie elementów – poprzedni element

```

procedure TForm1.btnPoprzedniClick(Sender: TObject);
begin
//sprawdzenie warunku na osiagniecie pierwszego elementu w tablicy
if (self.el_aktualny>1) then
begin
self.el_aktualny:=self.el_aktualny-1;
self.edtWspX.Text:=FloatToStr(self.Tablica[self.el_aktualny].x);
self.edtWspY.Text:=FloatToStr(self.Tablica[self.el_aktualny].y);
end;
//wypisanie informacji o ilosci elementów i aktualnym punkcie
self.lblOstatni.Caption:='Liczba wszystkich punktów: '+intToStr(self.el_ostatni);
self.lblAktualny.Caption:='Numer aktualnego punktu: '+intToStr(self.el_aktualny);
end;

```

Przeglądanie elementów – następny element

```

procedure TForm1.btnNastepnyClick(Sender: TObject);
begin
//sprawdzenie warunku na przekroczenie ostatniego wpisanego elementu
if (self.el_aktualny<self.el_ostatni) then
begin
self.edtWspX.Text:=FloatToStr(self.Tablica[self.el_aktualny].x);
self.edtWspY.Text:=FloatToStr(self.Tablica[self.el_aktualny].y);
self.el_aktualny:=self.el_aktualny+1;
end;
//wypisanie informacji o ilosci elementów i aktualnym punkcie
self.lblOstatni.Caption:='Liczba wszystkich punktów: '+intToStr(self.el_ostatni);
self.lblAktualny.Caption:='Numer aktualnego punktu: '+intToStr(self.el_aktualny);
end;

```

Zmiana wartości aktualnego elementu

```
procedure TForm1.btnZmienClick(Sender: TObject);
begin
  self.Tablica[self.el_aktualny].x:=StrToFloat(self.edtWspX.Text);
  self.Tablica[self.el_aktualny].y:=StrToFloat(self.edtWspY.Text);
end;
```

Usuwanie elementu (zmiana informacji o tym, ile jest wszystkich elementów)

```
procedure TForm1.btnUsunClick(Sender: TObject);
begin
  // sprawdzenie warunku na pierwszy punkt
  if (self.el_ostatni=1) then
  begin
    if (self.el_aktualny=self.el_ostatni) then
    begin
      self.el_aktualny:=self.el_aktualny-1;
      self.edtWspX.Text:='0';
      self.edtWspY.Text:='0';
      self.btnUsun.Enabled:=False;
      self.btnZmien.Enabled:=False;
      self.btnPoprzedni.Enabled:=False;
      self.btnNastepny.Enabled:=False;
    end;
    self.el_ostatni:=self.el_ostatni-1;
    //wypisanie informacji o ilosci elementow i aktualnym punkcie
    self.lblOstatni.Caption:='Liczba wszystkich punktów: '+intToStr(self.el_ostatni);
    self.lblAktualny.Caption:='Numer aktualnego punktu: '+intToStr(self.el_aktualny);
  end;
  //sprawdzenie ile wpisano elementów
  if (self.el_ostatni>1) then
  begin
    if (self.el_aktualny=self.el_ostatni) then
    begin
      self.el_aktualny:=self.el_aktualny-1;
      self.edtWspX.Text:=FloatToStr(self.Tablica[self.el_aktualny].x);
      self.edtWspY.Text:=FloatToStr(self.Tablica[self.el_aktualny].y);
    end;
    self.el_ostatni:=self.el_ostatni-1;
    //wypisanie informacji o ilosci elementow i aktualnym punkcie
    self.lblOstatni.Caption:='Liczba wszystkich punktów: '+intToStr(self.el_ostatni);
    self.lblAktualny.Caption:='Numer aktualnego punktu: '+intToStr(self.el_aktualny);
  end;
  //umożliwienie dodawania nowych elementów (uruchomienie przycisku dodawania)
  if (self.el_ostatni<MAX_TAB) then
  begin
    self.btnDodaj.Enabled:=True;
  end;
```

3. Podstawowe komponenty i ich właściwości

3.1. Pola Wyboru (TCheckBox, TRadioButton, TRadioGroup)

Użycie komponentów `CheckBox`, `RadioButton` pozwala na wybranie określonej opcji poprzez zaznaczenie pola wyboru - krzyżyka (`CheckBox`) lub zaczernionego kółka (`RadioButton`). Komponenty te można grupować, co pozwala na automatyczne zaznaczenie jednej opcji i odznaczenie pozostałych.

Przyciski opcji umieszczane w formularzu będą automatycznie traktowane jako część tej samej grupy. Jeżeli zachodzi potrzeba wyróżnienia więcej niż jednej grupy przycisków opcji, które funkcjonują niezależnie od siebie, wtedy należy użyć komponentu `RadioGroup`. Komponent tego typu umożliwia szybkie ustawienie grupy przycisków opcji razem z otaczającą je trójwymiarową ramką i opisem.

Przydatne własności pola wyboru

(`TCheckBox`, `TRadioButton`)

`Caption` - jest to tekst wyświetlany opisujący pole wyboru

`Checked` - ustawienia stanu zaznaczenia, a także odczytanie bieżącego stanu.

(`TRadioGroup`)

`Items` - określa opisy elementów listy, pole typu `TStrings`

`ItemIndex` - określa indeks wybranego elementu (pierwszy element ma indeks 0)

Przydatne zdarzenia pola wyboru

`OnClick` - kliknięcie na danym polu

3.2. Listy rozwijalne (`TListBox`, `TComboBox`)

Komponent `ListBox` reprezentuje standardową listę wyboru w Windows. Jeżeli lista zawiera więcej elementów, niż jest w stanie jednocześnie wyświetlić, dostęp do pozostałych jest możliwy dzięki paskom przewijania.

Komponent `ComboBox` jest polem edycyjnym z listą rozwijalną. Użytkownik może wybrać element z listy lub wpisać wartość w polu edycji.

Przydatne własności listy rozwijalnej

`Items` - elementy listy, pole typu `TStrings`

`MultiSelect` - czy możliwe jest wybranie więcej niż jednego elementu

`ItemIndex` - określa indeks wybranego elementu (pierwszy element ma indeks 0)

Przydatne metody listy rozwijalnej

`ComboBox1.Items.Add('nowy element listy')`

- dodawanie nowych elementów listy (w sposób dynamiczny możemy edytować listę)

`ComboBox1.Items.Delete(nr_elementu:integer)`

- usuwanie elementów z listy (w sposób dynamiczny możemy modyfikować listę)

Przydatne zdarzenia listy rozwijalnej

`OnClick` - kliknięcie na danym polu

3.3. Panele (TPanel)

Komponent `Panel` służy między innymi do przechowywania przycisków paska narzędzi, wyświetlania etykiet tekstowych (takich jak tytuł formularza), wyświetlania grafiki, jak również przechowywania zwykłych przycisków. Jedną z zalet Panelu jest to, że umieszczone w nim komponenty stają się jego potomkami, co pozwala na wygodne łączenie elementów w grupy. Elementy znajdujące się na panelu można przesuwać razem z panelem i ustawiać względem panelu co znacznie ułatwia projektowanie interfejsu użytkownika.

Przydatne własności paneli

`Align` - automatyczne dopasowanie panelu do lewej/prawej/góry/dołu lub pozostałego wolnego miejsca. Pozwala na wygodne rozplanowanie wyglądu okna.

1.2.1 Menu (TMainMenu, TPopupMenu)

Komponent `MainMenu` pozwala na zaprojektowanie górnego menu tekstowego programu. Komponent `PopupMenu` pozwala na stworzenie menu kontekstowego programu, łączonego z danym komponentem i wyświetlane po najechaniu na komponent kursorem i kliknięciu prawego klawisza myszy. Chcąc dodać zaprojektowane menu kontekstowe (wywoływanie w kontekście danego komponentu) należy we własności `PopupMenu` danego komponentu wybrać kojarzone z nim menu kontekstowe. Z każdym z widocznych komponentów można skojarzyć inne menu kontekstowe.

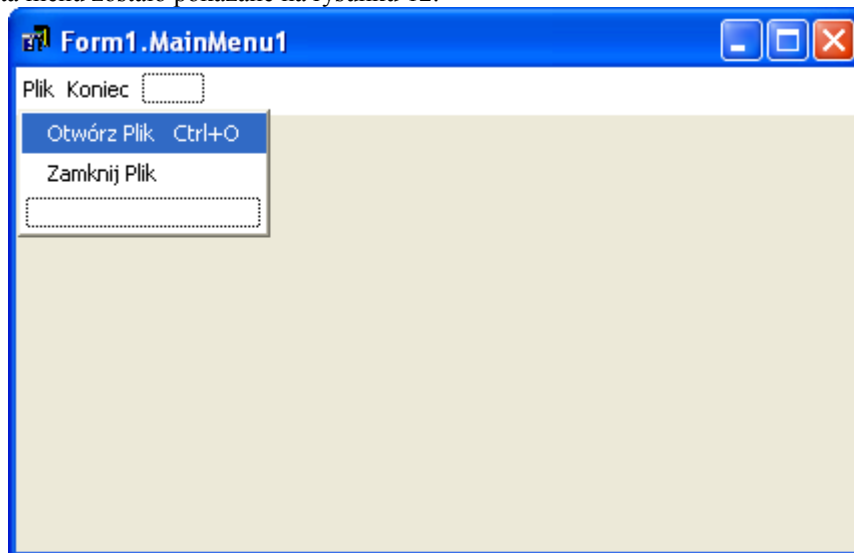
Projektowanie menu z wykorzystaniem Projektanta Menu (ang. Menu Designer)

Dwukrotne kliknięcie na komponenty `MainMenu`, `PopupMenu` dodające menu do formularza uruchamia program narzędziowy projektant menu (ang. Menu Designer), który pozwala na wizualne projektowanie menu tekstowego. Każdy z elementów menu jest oddzielnym komponentem, obiektem będącym potomkiem menu nadrzędnego.

Możliwe jest dodanie podmenu do danego elementu menu. Podmenu jest elementem menu, który po kliknięciu ukazuje nowe menu z nowymi opcjami.

Element menu można połączyć z odpowiednim skrótem klawiszowym, który powinien ułatwiać pracę ze stworzonym programem. Tworzenie skrótu do danego elementu menu polega na zmianie jego właściwości `ShortCut`. Po wybraniu tej właściwości można wybrać skrót klawiszowy z dostępnych tam skrótów (np. `Ctrl+C`).

Okno projektanta menu zostało pokazane na rysunku 12.

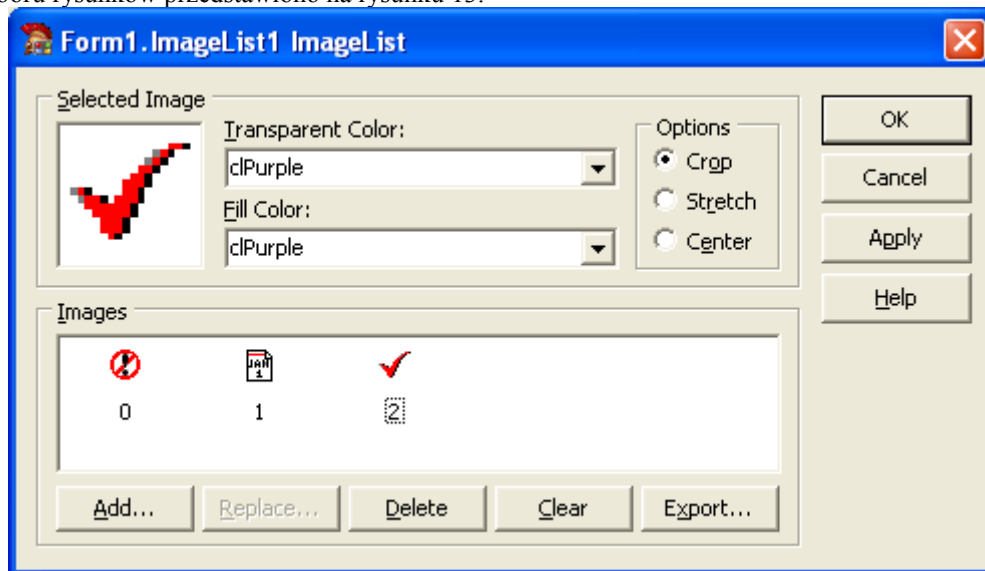


Rysunek 1. Okno projektanta menu, do opcji menu 'Otworz Plik' został dodany skrót klawiszowy Ctrl+O

Projektowanie menu przycisków szybkiego dostępu

Menu przycisków szybkiego dostępu zwykle znajduje się pod głównym menu i zawiera przyciski do najważniejszych opcji menu głównego oraz aplikacji. Menu przycisków szybkiego dostępu projektuje się wykorzystując komponenty `ImageList` (paleta Win32), oraz `ToolBar` (paleta Win32). Dodawanie przycisków szybkiego dostępu wykonuje się w następujący sposób.

Wkleić na formularz komponent `ImageList`. Kliknąć na komponent `ImageList`, powinno pokazać się okienko do ustawiania obrazków i ich własności, które później znajdą się na przyciskach szybkiego dostępu. Okno wyboru rysunków przedstawiono na rysunku 13.



Rysunek 2. Okno do ustawiania własności listy obrazków

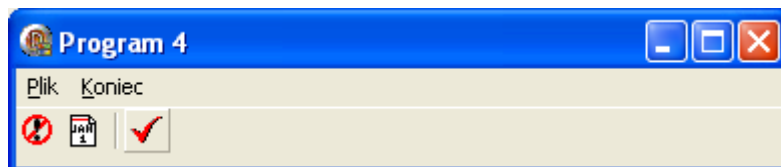
Każdy z rysunków, które indeksowane są od 0 można dodać wciskając *Add* i wybierając odpowiedni rysunek. Zbiór podstawowych rysunków znajduje się standardowo w katalogu: *Program Files->Common Files->Borland Shared->Images->Buttons*, ale oczywiście użytkownik może zaprojektować własny obrazek który znajdzie się na przycisku.

Jeśli podczas wybrania rysunku dzieli się on na dwa rysunki należy jeden z nich skasować (rysunek pokazujący się podczas dezaktywacji przycisku). Wybrać jeden rysunek odpowiednio do każdej z funkcji wywoływanej z paska skrótów.

Wkleić na formularz komponent `ToolBar`. Klikając prawym przyciskiem myszy na komponent `ToolBar` można dodać kolejne przyciski przy pomocy komendy *New Button*, oraz ewentualnie przerwę pomiędzy grupami przycisków przy pomocy komendy *New Separator*. Chcąc powiązać przyciski z rysunkami należy we własnościach komponentu `ToolBar` ustawić własność `Images` na nazwę komponentu z listą obrazków `ImageList` np. `ImageList1`.

Każdy z przycisków posiada własność `ImageIndex`, która jest numerem danego rysunku na liście rysunków. Należy pamiętać że pierwszy rysunek na liście ma indeks 0.

Postać programu z dodanym do menu z rysunku 12 menu przycisków szybkiego dostępu znajduje się na rysunku 14.



Rysunek 3. Program z menu głównym i paskiem przycisków szybkiego dostępu

1.2.2 Pasek stanu (TStatusBar)

Komponent `StatusBar` jest paskiem stanu zawierającym informacje o stanie aplikacji podczas jej działania. Może być pomocny podczas wyświetlania komunikatów istotnych dla użytkownika. Składa się z pól tekstowych zwanych panelami, których ilość i wielkości można kontrolować podczas działania programu.

Złożony pasek stanu to pasek składający się z wielu paneli. Po wybraniu złożonego paska stanu do ułożenia na nim paneli można wykorzystać Edytor Paneli Paska Stanu (`StatusBar Panels Editor`). Wywołanie edytora wymaga dwukrotnego kliknięcia na obszarze pola wartości właściwości `Panels`. Przycisk `Add Selected` dodaje nowy obszar, przycisk `Delete Selected` usuwa wybrany obszar. Aby dokonać edycji panelu należy wybrać go w Edytorze Paneli a następnie zmodyfikować jego właściwości w oknie Inspektora Obiektów. W celu wpisania jakiejś wartości do paska stanu należy wcześniej utworzyć przynajmniej jeden panel, inaczej aplikacja zwróci błąd.

Aby utworzyć panel na pasku stanu, lub zmodyfikować jego własności z poziomu kodu należy odnieść się do jego własności `Panels` (klasa `TStatusPanels`). Posiada ona następujące metody i własności:

Tabela 1. Własności i metody klasy `TStatusPanels`

Własności i metody	Opis
<code>Panels.Add</code>	Dodanie nowego panelu
<code>Panels.Text</code>	Tekst wyświetlany na panelu (trzeba podać numer panelu).
<code>Panels.Width</code>	Szerokość panelu
<code>Panels[numer panelu]</code>	Dotyczy pasków stanu z wydzielonymi panelami. Własność ta definiuje indywidualne panele paska
<code>AutoHint</code>	Kiedy kursor myszy przemieszcza się nad dowolnym komponentem, dla którego ustawiona została właściwość <code>Hint</code> , na pasku stanu automatycznie wyświetlany jest tekst pomocy kontekstowej
<code>SimplePanel</code>	Określa, czy pasek stanu wyświetla prosty panel, czy zbiór paneli. <i>Prosty pasek stanu</i> posiada tylko jeden panel zajmujący całą jego długość. <code>SimplePanel</code> działa jak przełącznik, dzięki któremu można przełączać się między prostym, a złożonym paskiem stanu (w trakcie pracy programu) przez zwykłą zmianę wartości tej właściwości na odpowiednio <code>True</code> lub <code>False</code>
<code>SimpleText</code>	Tekst przeznaczony dla prostego panelu paska stanu
<code>SizeGrip</code>	Określa, czy w prawym dolnym rogu paska stanu wyświetlany będzie uchwyt zmiany rozmiaru okna, poprzez który w łatwy sposób można zmieniać rozmiar okna

Sposób wykorzystania komponentu `StatusBar` został opisany podczas projektowania przykładowej aplikacji w rozdziale **Przykład 4??**.

1.2.3 Elastyczne obszary interfejsu (TSplitter)

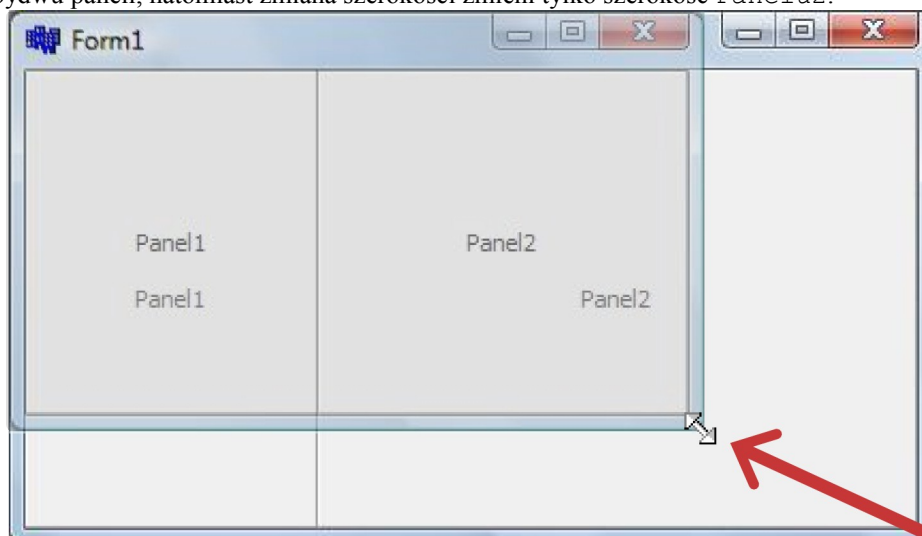
Zależy nam na tym, żeby interfejs wyglądał dobrze i umożliwiał pracę niezależnie od rozmiaru okna, a najlepiej, gdyby użytkownik miał możliwość samodzielnego dostosowania rozmiaru obszarów interfejsu. Przedstawiony już został komponent `TPanel`, który jest idealnym nośnikiem dla innych komponentów. Omówmy dokładniej własność `Align`, którą posiada większość komponentów. Najważniejsze opcje własności `Align` zostały przedstawione w tabeli 2.

Tabela 2. Opcje własności `Align`

Opcja	Opis
-------	------

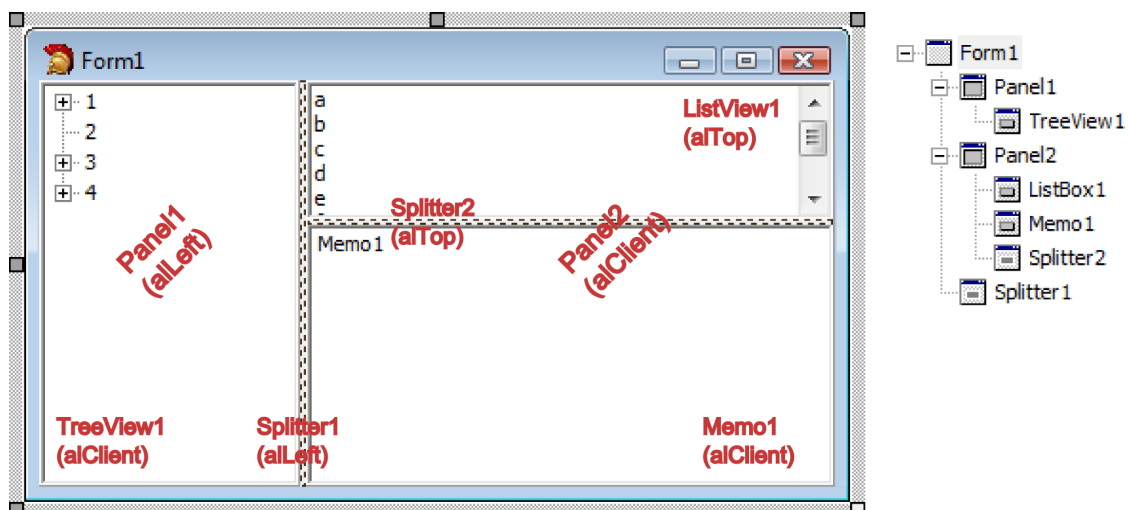
alBottom	Komponent zajmuje dolny obszar na całej szerokości. Zmiana szerokości okna zmienia szerokość komponentu.
alClient	Komponent zajmuje cały dostępny obszar. Zmiana rozmiaru okna zmienia rozmiary komponentu.
alLeft	Komponent zajmuje lewy obszar na całej wysokości. Zmiana wysokości okna zmienia wysokość komponentu.
alNone	Komponent zajmuje obszar o wymiarach podanych we własnościach Width i Height. Zmiana rozmiaru okna nie wpływa na rozmiar komponentu.
alRight	Komponent zajmuje prawy obszar na całej wysokości. Zmiana wysokości okna zmienia wysokość komponentu.
alTop	Komponent zajmuje górny obszar na całej szerokości. Zmiana szerokości okna zmienia szerokość komponentu.

Na rysunku 15 pokazano przykład wykorzystania własności Align. Panel1 wyrównany jest do lewej strony okna (alLeft), a Panel2 zajmuje cały pozostały obszar (alClient). Zmiana wysokości okna zmienia wysokość obydwu paneli, natomiast zmiana szerokości zmienia tylko szerokość Panelu2.

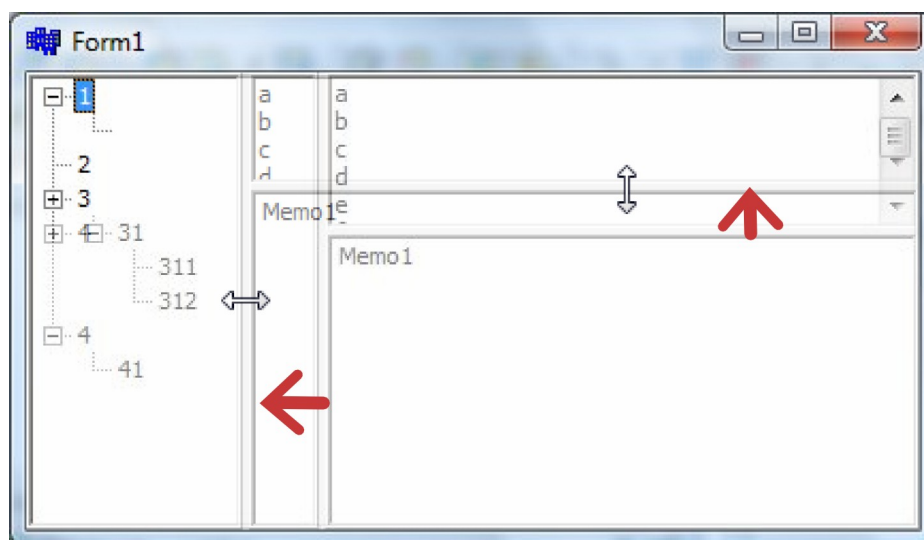


Rysunek 4. Przy zmianie rozmiarów okna szerokość Panelu1 nie zmieniła się

W tym prostym przykładzie nie ma możliwości zmiany szerokości Panelu1. Do tego potrzebny będzie komponent klasy TSplitter z zakładki Additional. Obiekt klasy TSplitter także posiada własność Align. Splitter musi zostać umieszczony pomiędzy panelami 1 a 2. Zakładając, że mamy czysty formularz najprościej zacząć od umieszczenia panelu pierwszego i ustawienia jego własności Align na alLeft. Następnie na pozostałym obszarze umieszczamy komponent TSplitter i ustawiamy jego własność Align również na alLeft. Na koniec umieszczamy panel 2 tak aby wypełniał pozostałą przestrzeń (alClient). Chwytając i przesuwać krawędź oddzielającą panele zmieniamy ich szerokość. Na rysunku 16 pokazany został projekt bardziej rozbudowanego interfejsu z wykorzystaniem Splittera pionowego i poziomego. Na Panelu1 umieszczono dodatkowo drzewo TreeView1, które ma ustawioną własność Align na alClient, co oznacza, że wypełnia cały obszar Panelu1 i będzie zmieniało swój rozmiar wraz ze zmianą rozmiaru tego panelu. Na Panelu2 umieszczono w odpowiedniej kolejności listę ListView1 (alTop), Splitter2 (alTop) oraz pole tekstowe Memo1 (alClient). Przesunięcie Splittera1 wpływa na wszystkie komponenty, podczas gdy przesunięcie Splittera2 tylko na listę ListView1 i pole tekstowe Memo1 (rys.17).



Rysunek 5. Zastosowanie komponentu klasy TSplitter



Rysunek 6. Zmiana rozmiarów komponentów za pomocą obiektu klasy TSplitter

1.3 Scentralizowane zarządzanie wywoływaniem i udostępnianiem funkcji

1.3.1 Lista akcji (TActionList)

Komponent `ActionList` może zostać wykorzystany do łatwej aktywacji i dezaktywacji wielu komponentów (przyciski, pasek przycisków szybkiego dostępu, opcje menu) za jednym razem. Pozwala też na utworzenie jednej procedury wywoływanej przez kilka komponentów, bez potrzeby wielokrotnego jej pisania.

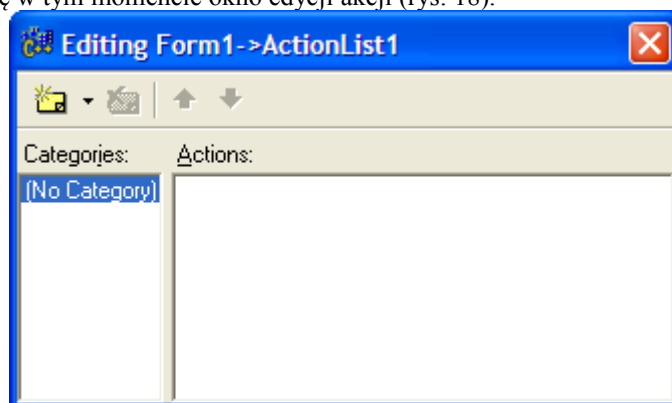
Przy pomocy Edytora Akcji, aktywowanego po kliknięciu na komponent `ActionList` można utworzyć akcje dotyczące najczęściej wykorzystywanych, lub standardowych procedur w programie. Powiązanie ich z odpowiednimi komponentami następuje poprzez ustawienie własności `Action` (klasa `TAction`) tych komponentów.

W Delphi można na wiele różnych sposobów utworzyć menu główne programu i procedury wykonujące podstawowe operacje. Tutaj zaprezentujemy podejście odpowiednie przy budowie dużych aplikacji, z wykorzystaniem komponentu `ActionList`. Komponent ten znajdziecie na palecie `Standard`. Należy on do grupy komponentów niewizualnych – po wstawieniu takiego komponentu pojawia się on na formularzu w postaci ikony, ale tylko w trybie projektowania aplikacji (`Design Mode`), w czasie wykonywania programu (`Run Time Mode`) ikona ta nie jest widoczna. Umieszczamy go więc gdziekolwiek na formie.

Zanim przejdziemy jednak do definiowania akcji, najpierw wyjaśnienie czym jest owa *akcja*. W dużym uproszczeniu możecie ją potraktować jako procedurę wywoływaną przez użytkownika – to z punktu widzenia programisty. Z punktu widzenia użytkownika programu akcja jest po prostu poleceniem wydawanym przez niego programowi. Tak więc w Delphi akcją jest np. *Run*. Za każdym razem zostanie wykonana kompilacja i uruchomiony aktualnie otwarty projekt, niezależnie od tego czy użytkownik wcisnął klawisz z zielonym trójkątem, czy wybrał z *Menu* pozycję *Run* | *Run* czy też wcisnął klawisz **F9**.

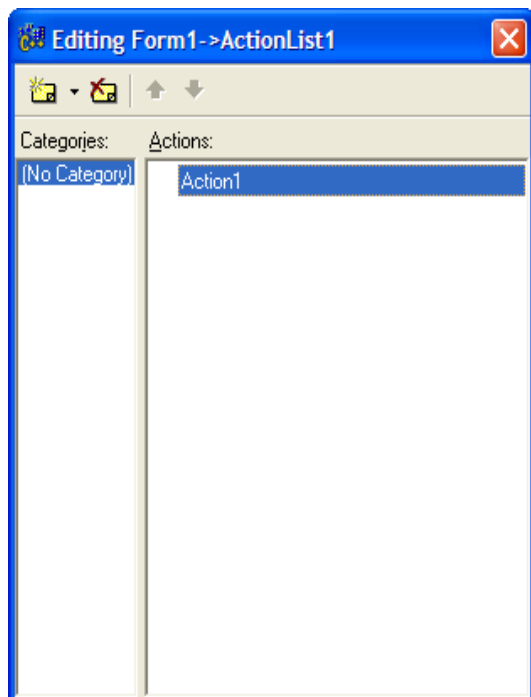
W naszym programie taką akcją może być np. *Zakończ*. Spróbujmy więc ją najpierw zdefiniować, a następnie udostępnić użytkownikowi w postaci odpowiedniej pozycji w menu, przycisku na pasku przycisków szybkiego dostępu oraz skrótu klawiaturowego.

Zacniemy od definicji samej akcji. Klikamy dwukrotnie na komponencie `ActionList` umieszczonym na formularzu – otworzy się w tym momencie okno edycji akcji (rys. 18).

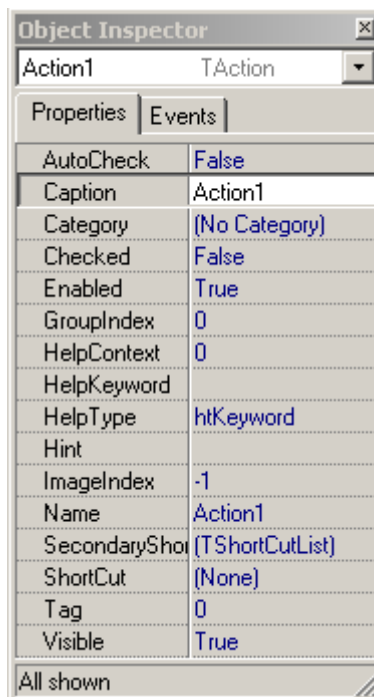


Rysunek 7. Okno edycji akcji

Teraz możemy przystąpić do definiowania akcji. Poprzez wybranie żółtej ikonki *New Action* lub wcisnięcie klawisza **Insert** dodamy nową akcję. Okienko powinno się zmienić do postaci pokazanej na rysunku 19.



Rysunek 8. Okno edycji po dodaniu nowej akcji

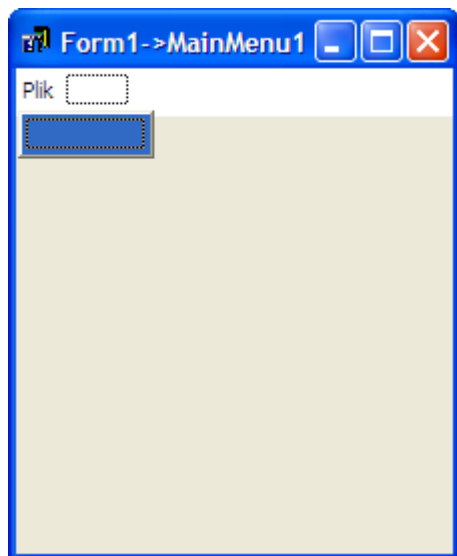


Rysunek 9. Inspektor obiektów – własności Action1

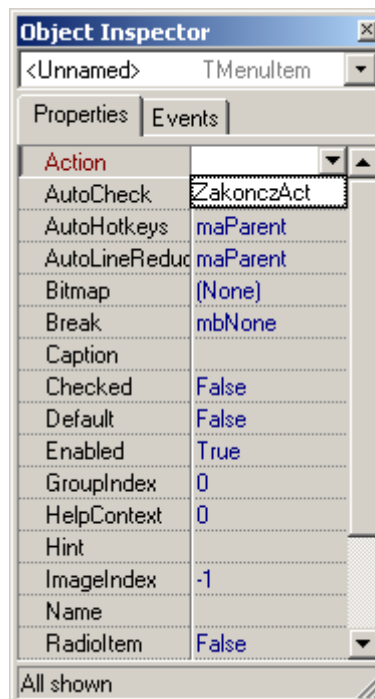
Widzicie, że pojawiła się nowa akcja, domyślnie nazwana `Action1`. Zaznaczamy ją i przechodzimy do inspektora obiektów (rys. 20). Zmieniamy właściwość `Caption` akcji na naszą etykietkę – powiedzmy niech to będzie `Zakończ`. Właściwość `Shortcut` definiuje klawisz (bądź kombinację klawiszy) która będzie wykorzystywana jako skrót do danej akcji. Możecie wybrać swoją własną kombinację – np. **Ctrl+K** oznacza kombinację klawiszy **Ctrl** i **K** wciśniętych równocześnie. Wypadałoby także zmienić nazwę naszej akcji – właściwość `Name` – na jakąś nazwę znaczącą. Niech to będzie `ZakonczAct`. Pozostaje jeszcze do zdefiniowania procedura – w tym celu przechodzimy do zakładki `Events` i klikamy dwukrotnie w pole obok napisu `OnExecute`. Delphi przeniesie nas automatycznie do edytora kodu, gdzie wstawi gotowy szablon procedury. Uzupełniamy ją do następującej postaci:

```
procedure TForm1.ZakonczActExecute(Sender: TObject);
begin
    Close();
end;
```

Zdefiniowaliśmy już akcję – pozostaje jedynie umieszczenie jej w interfejsie użytkownika. Zaczniemy od menu. W Delphi do tworzenia menu głównego służy komponent `MainMenu` z palety `Standard`. Również umieszczamy go gdziekolwiek na formie i klikamy w niego dwa razy. Pojawi się edytor menu. Za jego pomocą dodajemy poszczególne pozycje do menu. W każdej chwili możemy przejść do inspektora obiektów (przycisk **F11**) i ustawić właściwości podświetlonej pozycji. Najbardziej interesująca dla nas na początek będzie właściwość `Caption` definiująca napis, jaki jest wyświetlany jako etykieta danej pozycji w menu. Spróbujcie więc utworzyć menu jak na rysunku 22.



Rysunek 10. Okno edycji menu głównego (MainMenu)



Rysunek 11. Inspektor obiektów - własności komponentu MainMenu

Teraz, skoro mamy pusty wpis (podświetlony na niebiesko) który chcielibyśmy powiązać z wybraną akcją – przechodzimy do inspektora obiektów (rys. 21) i ustawiamy właściwość `Action`, wybierając po prostu interesującą nas akcję z rozwijanej listy.

3.7. Tabela (TStringGrid)

Komponent `StringGrid` służy do wyświetlania i organizacji danych w postaci tabeli. Podczas projektowania i pracy aplikacji możliwe jest określenie własności i wyglądu tabeli, opisu odpowiednich danych oraz dodawanie i usuwanie nowych kolumn, rzędów i wybranych komórek.

Komponenty o zbliżonych właściwościach do tabeli `StringGrid` to m.in. `ValueListEditor`, `DrawGrid`, `DBGrid`.

Przydatne własności tabeli

<code>RowCount</code> , <code>ColCount</code>	- Określenie ilości rzędów i kolumn, zwiększenie tej wartości pozwala na dodanie nowego rzędu.
<code>FixedRows</code> , <code>FixedCols</code>	- Określenie rzędów i kolumn o zablokowanych komórkach, czyli takich które się nie przesuwają podczas przewijania z wykorzystaniem pasków przewijania (ang. <i>scrollbars</i>).
<code>Cells[wiersz][kolumna]</code>	- Odniesienie się odpowiedniej komórki.
<code>Scrollbars</code>	- Wyświetlenie pasków przewijania, własność ta może przyjmować wartości:

- `ssVertical` – paski przewijania pionowe.
- `ssHorizontal` – paski przewijania poziome.
- `ssBoth` – paski przewijania poziome i pionowe.
- `ssNone` – bez pasków przewijania.

Przydatne zdarzenia tabeli

OnSelectCell

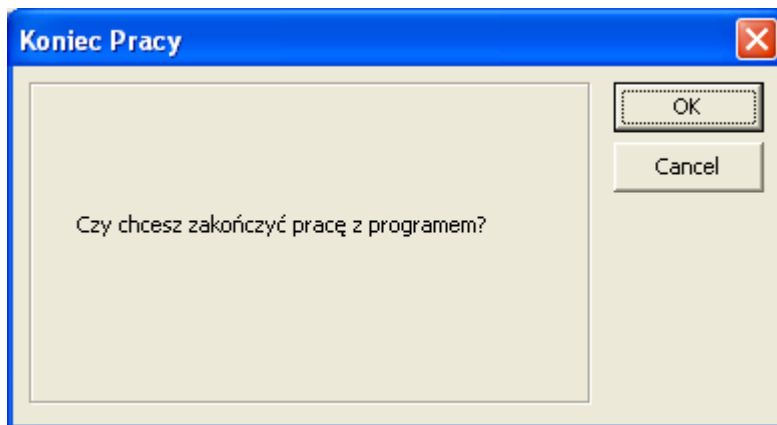
– wywołane w momencie wyboru jednej z komórek tabeli.

5. Okna dialogowe

5.1. Wykorzystanie standardowych okien dialogowych

Okna dialogowe pozwalają na komunikację z użytkownikiem i czekają na jego reakcję jak wybór, lub potwierdzenie albo przekazują mu informacje np. o programie, o wykonaniu zadania. Ponieważ są oddzielnymi formularzami, dodaje się je podobnie jak formularze. Należą one jednak do wyspecjalizowanych klas formularzy.

Nie zamykając projektu dodajemy do niego okno dialogowe (nowy formularz) wybierając opcje *File->New->Other->Delphi Projects->Delphi Files* i wybierając jedno z dostępnych standardowych okien dialogowych *Standard Dialog*. Przykładowe okno dialogowe znajduje się na rysunku 5.1.



Rysunek 5.1. Standardowe okno dialogowe do zakończenia pracy z programem.

Następnie należy w pliku związanym z tym oknem dialogowym (np. Unit2.pas) sprawdzić, w linii zdefiniowanych zmiennych jak nazywa się obiekt odpowiadający za okno dialogowe

```
var  
    OKRightDlg: TOKRightDlg;
```

Chcąc umożliwić wywoływanie okienka dialogowego z programu głównego, oraz pobierać z niego informacje, należy dołączyć plik z nim związany w sekcji *uses* na początku programu związanego z formularzem, w którym to okno dialogowe będzie wywoływane.

Otwarcie okienka dialogowego wykonuje się poprzez wybranie metody (analogicznie do przełączania pomiędzy formularzami):

```
OKRightDlg.Show() - możliwe jest przełączanie pomiędzy oknami programu  
OKRightDlg.ShowModal() - przełączanie pomiędzy oknami programu jest zablokowane
```

Informacje o tym, który przycisk okna dialogowego został naciśnięty można uzyskać sprawdzając własność okienka dialogowego *ModalResult*. Każdy z przycisków ma przyporządkowaną wartość o standardowej nazwie:

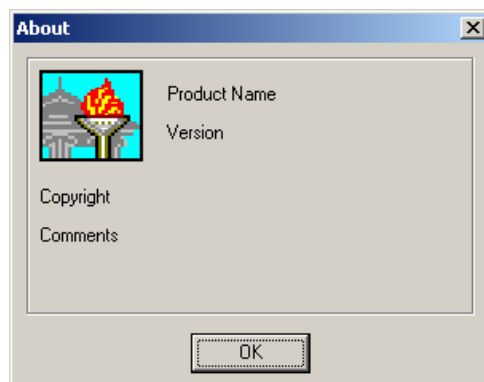
Przycisk 'OK'	- <code>OKRightDlg.ModalResult=mrOk</code>
Przycisk 'Cancel'	- <code>OKRightDlg.ModalResult=mrCancel</code>

Po dodaniu okna dialogowego do aplikacji można zaprojektować jego wygląd analogicznie jak w przypadku formularza, korzystając ze standardowych komponentów dostępnych w środowisku *TurboDelphi*.

5.2. Okno dialogowe informacyjne (informacja o programie)

Nie zamykając projektu dodajemy do niego okno informacyjne o programie (nowy formularz) wybierając opcje *File->New->Other->Delphi Projects->Delphi Files ->About box*. Okno informacyjne pokazano na rysunku 5.2.

Wykorzystanie tego okna jest analogiczne do wykorzystania standardowego okna dialogowego. Należy zaznaczyć, że jest to przykład formularza o pewnych szczególnych właściwościach (np. `ProductName`, `Comments`, `ProgramIcon`, `Version`), które można modyfikować. Okno dialogowe informacyjne może być otwarte zarówno w trybie zwykłym jak i w trybie modalnym.



Rysunek 5.2. Okno informacyjne `AboutBox`.

5.3. Okno dialogowe do pracy z plikami

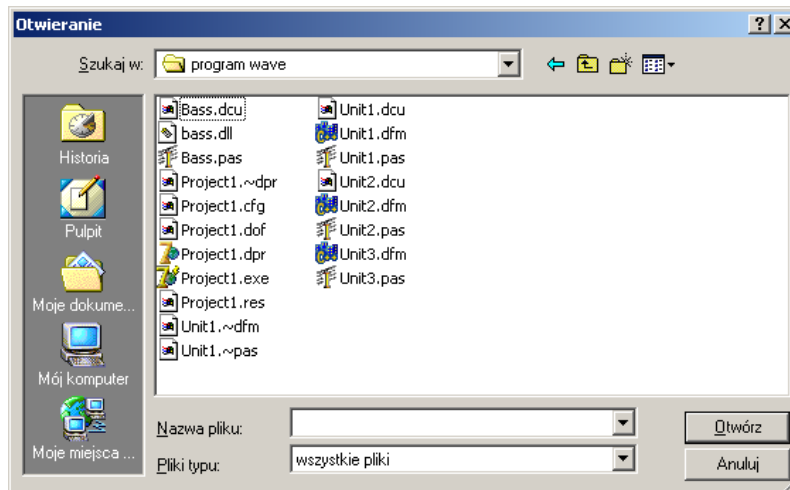
Podczas pisania programów często korzysta się z odczytu danych z pliku, lub zapisu danych do pliku. Do wyboru nazw plików można wykorzystać komponenty standardowych okien dialogowych `OpenDialog` (wybór pliku do odczytu) i `SaveDialog` (wybór pliku do zapisu) – obydwa komponenty można znaleźć na palecie `Dialogs`. Wybór nazwy pliku z wykorzystaniem okna dialogowego wykonuje się w następujący sposób.

Na początku należy dodać do formularza komponent `OpenDialog`.

Wczytywanie pliku tekstowego z wykorzystaniem okienka dialogowego zwykle jest wykonywane poprzez obsługę zdarzenia `OnClick` odpowiedniego elementu (np. elementu głównego menu, przycisku itp.). Przykładowa procedura może być następująca:

```
procedure TForm1.OtworzPlik1Click(Sender: TObject);
begin
    if self.OpenDialog1.Execute() then
    begin
        //tu napisac procedurę wykonujaca operacje na wybranym pliku
        ...
        //przykładowo wypisanie na etykiecie nazwy pliku
        self.lblNazwa.Caption:='wczytano plik'+self.OpenDialog1.FileName;
        ...
    end;
end;
```

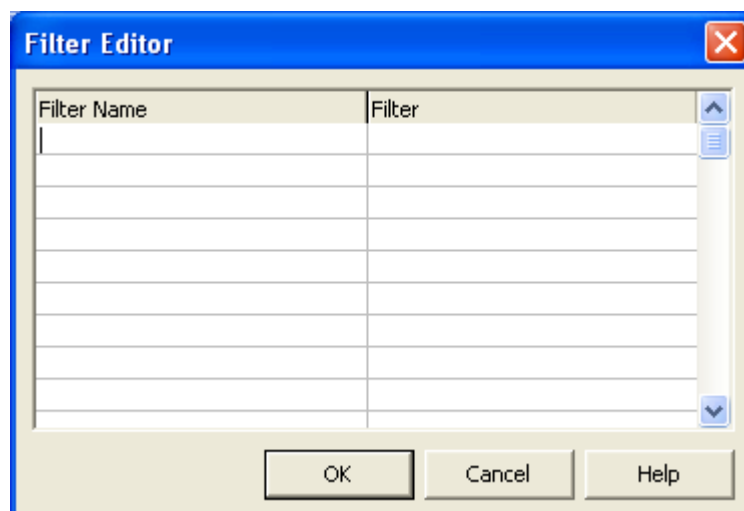
Otwarcie okna dialogowego do odczytu/zapisu pliku wykonuje się poprzez wywołanie metody tego okna `Execute()`. Powoduje to otwarcie tego okna w trybie modalnym (bez możliwości przełączania pomiędzy oknami programu). Po wybraniu odpowiedniego pliku, jego nazwa przechowywana jest w zmiennej `OpenDialog.FileName`. Przykładowe okno dialogowe do wyboru pliku do odczytu zostało przedstawione na rysunku 5.3.



Rysunek 5.3. Postać okienka dialogowego
OpenDialog

We własnościach komponentu `OpenDialog` można określić rodzajem plików, które będą wyświetlane wybierając własność `Filter` i definiując filtry do wyboru odpowiednich plików. Otwiera się wtedy tzw. edytor filtrów (ang. `Filter Editor`) (rysunek 5.4) i modyfikując pola:

- Filter Name – nazwa odpowiedniego filtru np. 'plik tekstowy'.
- Filter – możliwe rozszerzenia np. '*.txt'.



Rysunek 5.4. Dodawanie filtrów do okna dialogowego
OpenDialog.

6. Wykorzystanie wielu formularzy, przełączanie i wymiana danych pomiędzy formularzami

W *Turbo Delphi* możliwe jest zaprojektowanie aplikacji posiadającej wiele formularzy. Z każdym formularzem związany jest odpowiedni moduł. W ramach jednej aplikacji można wywoływać kilka formularzy. Należy umieścić informacje o modułach nowych formularzy w sekcji `uses` formularza, z którego będziemy otwierać inne formularze. Dodanie nowego formularza do aplikacji można wykonać następująco

Na początku należy utworzyć nowy formularz. Nie zamykając projektu dodajemy do niego nowy formularz wybierając opcje z menu *File->New->Form*

Następnie powinno się ustawić komunikację pomiędzy formularzami. W tym celu w głównym formularzu należy dodać moduł związany z drugim formularzem, poprzez umieszczenie nagłówka z nim związanego go w sekcji `uses`. Formularze nie mogą wywoływać siebie nawzajem (nie mogą mieć ustawionych siebie nawzajem w sekcji `uses`)

Otwarcie dodanego formularza (podrzednego) wykonuje się poprzez wybranie metody:

`Show()` – możliwe jest przełączanie pomiędzy oknami programu
`ShowModal()` – przełączanie pomiędzy oknami programu jest zablokowane

Chcąc odwoływać się do komponentów dodanego formularza (podrzednego), należy wykorzystywać jego nazwę, która znajduje się w jego sekcji zmiennych np.

```
var  
    Form2: TForm2;
```

w treści formularza 1 (`Form1`), w obsłudze zdarzenia lub odpowiedniej procedurze, można więc napisać

```
Form2.Show();
```

Co spowoduje otwarcie nowego formularza.

7. Program 4

Kolejna aplikacja będzie umożliwiała utworzenie prostej bazy czasopism oraz wyświetlanie jej w formie tabeli na oddzielnym formularzu. Pokazane zostanie wykorzystanie komponentów wyboru opcji z obsługą ich podstawowych własności, oraz okna dialogowe. Wyświetlanie informacji o czasopismach zrealizowane będzie na oddzielnym formularzu wykorzystując obsługę kilku formularzy, oraz pole edycji wielolinijkowej. Dodatkowo zostanie opisane dokładnie projektowanie menu, menu z przyciskami skrótów i menu kontekstowego oraz programowanie akcji.

Sposób projektowania programu jest następujący. Należy utworzyć nową aplikację *File>New>VCL Forms Application*, a następnie zapisać projekt przy pomocy *File>Save All* w katalogu, gdzie będzie kompilowany projekt. Na początku zostanie zaprojektowany główny formularz. W tym celu należy dodać odpowiednie komponenty w kolejności

3 panele `Panel` (paleta `Standard`) do ustawiania na nich komponentów

1 pole edycyjne `Edit` (paleta `Standard`) o nazwie (własność `Name`)

- `edtTytul`

5 etykiet `Label` (paleta `Standard`)

- `lblTytul`
- `lblRodzaj`
- `lblDostepnosc`

1 lista wyboru `ComboBox` (paleta `Standard`) o nazwie :

- `cbRodzaj`

1 panel wyboru przy pomocy przycisków `RadioGroup` (paleta `Standard`) o nazwie:

- `rgDostepnosc`

3 przyciski `Button` (paleta `Standard`) o nazwach:

- `btnDodaj`

- btnDane
- btnUsun
- btnWyswietl

1 komponent MainMenu (paleta Standard)

1 komponent ImageList (paleta Win32)

1 komponent Toolbar (paleta Win32)

1 komponent Statusbar (paleta Win32)

1 komponent AboutBox (paleta Dialogs)

Zaprojektowany formularz znajduje się na rysunku 7.1.

Rysunek 7.1. Formularz dodawania elementów (formularz główny)

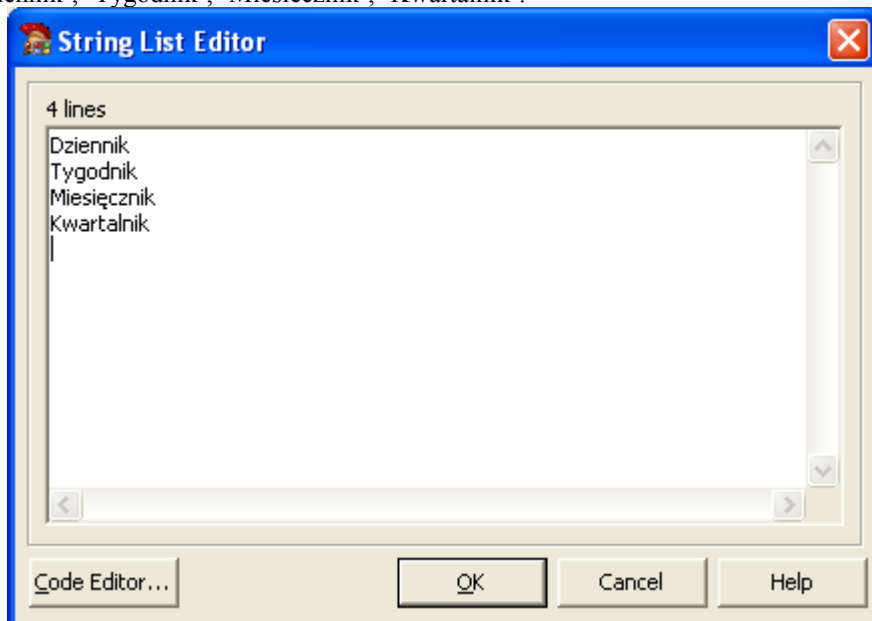
W kolejnym kroku zostanie zaprojektowane menu główne. Po kliknięciu na komponent MainMenu otworzy się tzw. Menu Designer służący do zaprojektowania głównego menu tekstowego. Z każdą opcją można skojarzyć odpowiednią kombinację klawiszy, która ją wywołuje. Skróty klawiszowe można dodać do każdego elementu menu zmieniając jego własność ShortCut. Należy zaprojektować menu główne tak, aby miało następujące funkcje

- Wizualizacja (skrót *Ctrl+W*)
- Koniec (skrót *Ctrl+Q*)

Następnie zostanie zaprojektowane menu przycisków szybkiego dostępu (patrz podrozdział: **Projektowanie menu przycisków szybkiego dostępu**)

Podczas projektowania formularza dodać odpowiednie opcje do komponentu pola wielokrotnego wyboru ComboBox (cbRodzaj). Opcje do tego komponentu dodaje się edytując własność Items. Otwiera się wtedy Edytor kolejnych elementów do wyboru (ang. *String List Editor*), przedstawiony na rysunku 7.2.

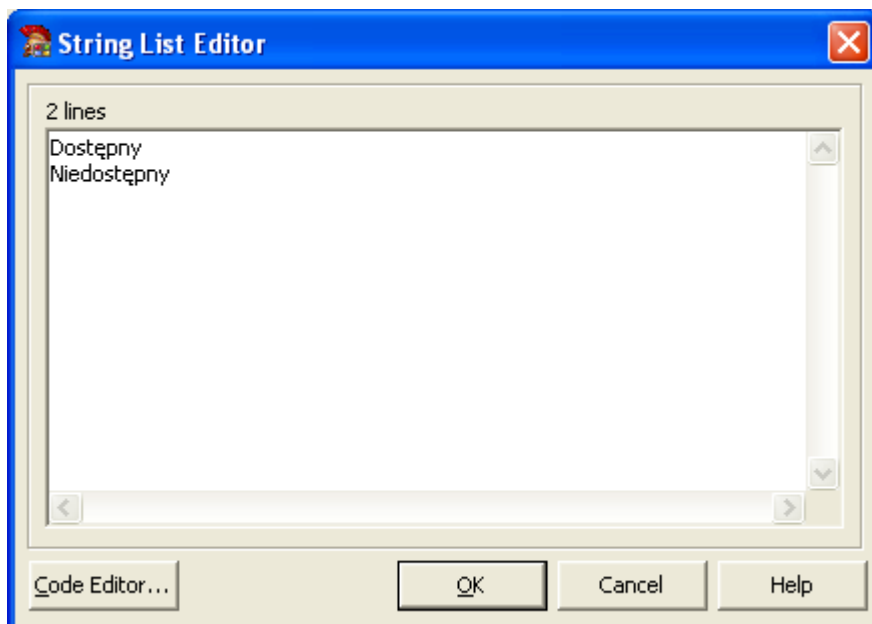
Utworzyć 4 elementy określające rodzaje wpisywanych czasopism (kolejne elementy zapisywane są w kolejnych liniach): 'Dziennik', 'Tygodnik', 'Miesięcznik', 'Kwartalnik'.



Rysunek 7.2. Edytor listy rozwijanej

Ustawić listę opcji na pierwszy element, poprzez ustawienie własności `ItemIndex` (indeks elementu w `ComboBox`) na 0.

Dodać odpowiednie opcje do komponentu wyboru opcji `TRadioGroup` (`rgDostepnosc`). Opcje do tego komponentu dodaje się edytując własność `Items`. Otwiera się wtedy Edytor kolejnych elementów do wyboru (analogicznie jak w przypadku komponentu `ComboBox`), przedstawiony na rysunku 7.3. Utworzyć 2 elementy określające dostępność wpisywanych czasopism (kolejne elementy zapisywane są w kolejnych liniach): 'Dostępny', 'Niedostępny'.



Rysunek 7.3. Edytor panelu wyboru opcji

Ustawić listę opcji na pierwszy element, poprzez ustawienie własności `ItemIndex` (indeks elementu w `TRadioGroup`) na 0. Ponieważ na panelu `TRadioGroup` wszystkie przyciski są zgrupowane i przekazują

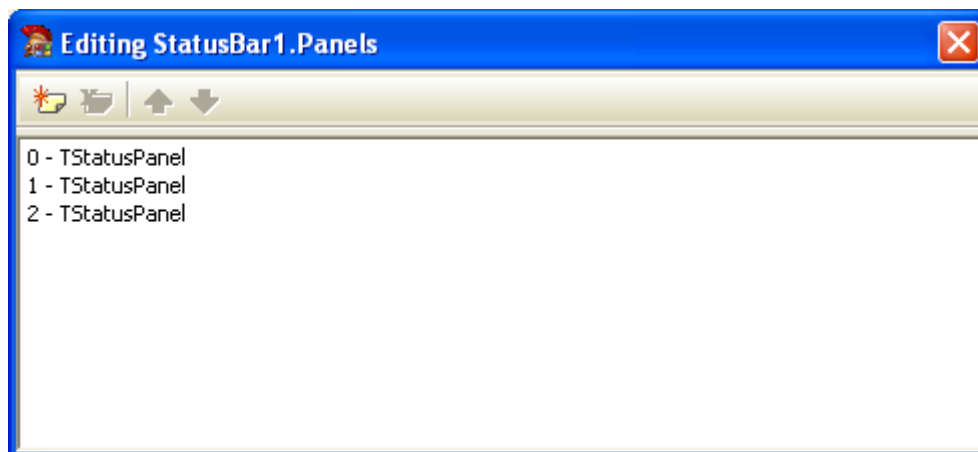
informacje o swoim statusie, czy są, czy też nie są wybrane (własność `Enabled`) to aktywacja jednego elementu automatycznie dezaktywuje pozostałe.

Utworzyć pasek stanu z trzema panelami. Panele dodaje się edytując własność *Panels*. Otwiera się wtedy Edytor paneli, przedstawiony na rysunku 7.4. Utworzyć pasek stanu z trzema panelami. Odpowiednio o szerokościach (własność *width* panelu):

Panel 0: 100

Panel 1: 100

Panel 2: reszta dostępnego miejsca

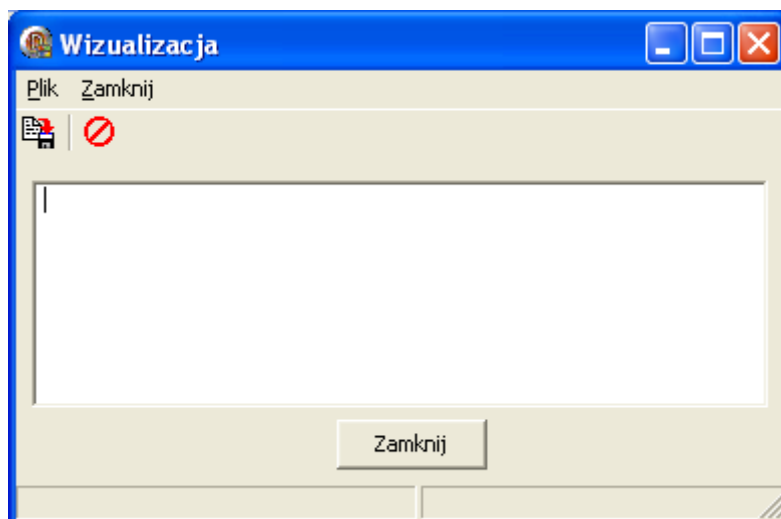


Rysunek 7.4. Edytor paneli paska stanu

W następnej kolejności zostanie utworzony drugi formularz służący do wyświetlenia listy wpisanych Czasopism. Zostanie wykorzystany w tym celu komponent `Memo`. Oprócz wyświetlania listy czasopism wyświetlana będzie informacja o ilości wpisanych czasopism, oraz umożliwione zapisywanie zbioru czasopism w pliku tekstowym. Nie zamykając projektu dodajemy do niego nowy formularz wybierając opcje *File->New->Form* Należy zaprojektować formularz do wizualizacji elementów. Dodajemy następujące komponenty

- 1 komponent `Button` (paleta `Standard`): nazwa `btnZamknij`
- 1 komponent `PopupMenu` (paleta `Standard`)
- 1 komponent `Memo` (paleta `Standard`): nazwa `memCzasopisma`
- 1 komponent `MainMenu` (paleta `Standard`)
- 1 komponent `ImageList` (paleta `Win32`)
- 1 komponent `ToolBar` (paleta `Win32`)
- 1 komponent `Statusbar` (paleta `Win32`)
- 1 komponent `SaveDialog` (paleta `Dialogs`)

Postać przykładowego formularza do wizualizacji znajduje się na rysunku 7.6.



Rysunek 7.6. Formularz wizualizacji elementów (formularz dodatkowy)

Następnie utworzyć pasek stanu z dwoma panelami. Panele dodaje się edytując własność *Panels*. Otwiera się wtedy Edytor paneli. Odpowiednio o szerokościach (własność *Width* panelu):

Panel 0: 200

Panel 1: reszta dostępnego miejsca

Należy zaprojektować menu główne tak, aby miało następujące funkcje.

- Plik -> Zapisz do pliku (skrót *Ctrl+S*)
- Zamknij

Utworzyć menu z przyciskami szybkiego dostępu (patrz podrozdział: **Projektowanie menu przycisków szybkiego dostępu**)

Następnie zaprojektować menu kontekstowe z jedną funkcją

- Czyszczenie pola edycyjnego (skrót *Ctrl+C*)

Menu kontekstowe należy skojarzyć z komponentem *memCzasopisma*, poprzez ustawienie własności *PopupMenu* na nazwę zaprojektowanego menu kontekstowego.

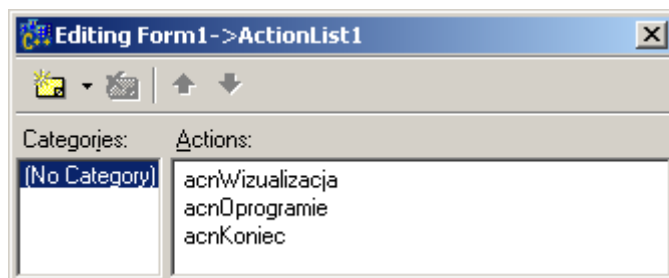
W kolejnym kroku ważne jest określenie komunikacji pomiędzy dwoma formularzami znajdującymi się w projekcie. Należy umożliwić otwieranie z formularza głównego (Form 1, plik "Unit1.pas") dodatkowego formularza do wizualizacji (Form 2, plik "Unit2.pas").

Aby można było wyświetlać i obsługiwać formularz do wizualizacji należy dołączyć odpowiedni plik nagłówkowy z nim związany. Dołączenie kolejnego formularza powinno znaleźć się w pliku *Unit1.pas* który związany jest z formularzem głównym. Należy dodać moduł *Unit2.pas* z wykorzystaniem dyrektywy *uses*.

```
uses
    ..., Unit2;
```

Jak widać, w projekcie jest kilka komponentów, które wywołują te same procedury. Przykładem może być wywołanie formularza z wyświetlaną tablicą z danymi na temat czasopism. Procedura otwierająca ten formularz wywoływana z elementu menu głównego 'wizualizacja', oraz z paska przycisków szybkiego dostępu. Dla takiej procedury warto utworzyć tzw. akcję, która będzie powiązana z każdym z tych komponentów. Akcje zapisuje się w komponentcie *ActionList*. Należy kliknąć na komponent *ActionList* i utworzyć akcje obsługujące funkcje programu (np. o nazwie *acnWizualizacja*). Zaletą akcji jest możliwość definiowania dodatkowych

własności, które będą przejmowane przez komponenty: napis - Caption, skrót klawiszowy - ShortCut, numer obrazka - ImageIndex. Z wykorzystaniem Edytora Akcji komponentu ActionList (dwukrotne kliknięcie na komponent otwiera edytor akcji) utworzyć następujące akcje.



Rysunek 7.7. Edytor Akcji (ActionList)

W przypadku projektowanej aplikacji należy utworzyć 3 akcje dla formularza głównego (Form1):

- acnWizualizacja - Otwieranie formularza do wizualizacji
- acnOprogramie - Wyświetlenie okna z informacją o programie
- acnKoniec - Zakończenie aplikacji

Otwieranie formularza do wizualizacji z programu głównego (acnWizualizacja)

Procedura ta będzie otwierała dodatkowy formularz, przy pomocy metody formularza Form->Show.

programowanie zdarzenia OnExecute akcji

```
procedure TForm1.acnWizualizacjaExecute(Sender: TObject);
begin
    Form2.Show;
end;
```

ustawienie własności

```
Caption:    Wizualizacja
ImageIndex:    0
ShortCut:    Ctrl+W
```

Wyświetlenie okna z informacją o programie (dodatkowy element w menu głównym) (acnOprogramie)

Do wypisania informacji o programie można wykorzystać standardowe okno dialogowe o nazwie *About Box*, żeby je utworzyć należy stworzyć nowy formularz w istniejącym projekcie

File>New>Other...>Delphi Projects->About Box

Ponieważ nowy formularz jest tworzony automatycznie z modulem, który go opisuje, moduł ten o musimy dodać do spisu modułów w module związanym z naszym głównym formularzem (analogicznie pod liniijką dodającą moduł związany z formularzem do wizualizacji)

```
uses
    ..., Unit3;
```

Okno dialogowe będzie miało nazwę *AboutBox*. Obsługa akcji będzie następująca

programowanie zdarzenia OnExecute akcji

```
procedure TForm1.acnOprogramieExecute(Sender: TObject);
```

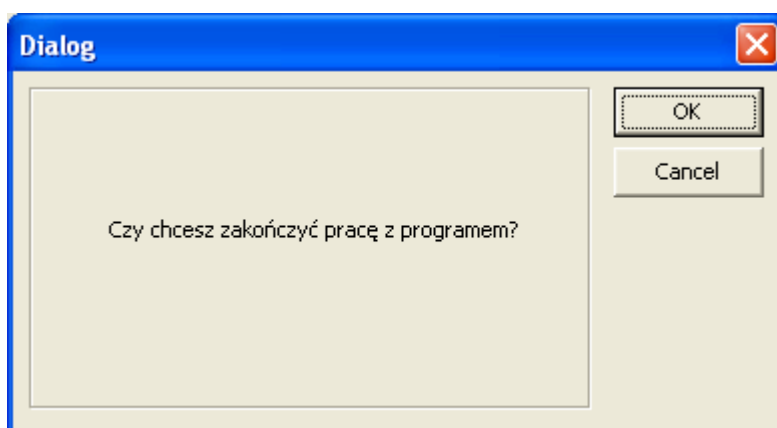
```
begin
    AboutBox.ProductName.Caption:='program Baza danych';
    AboutBox.Show();
end;
```

ustawienie własności akcji

```
Caption:    O programie
ImageIndex: 1
ShortCut:   Ctrl+A
```

Zakończenie aplikacji z wykorzystaniem okienka dialogowego (acnKoniec)

Nie zamykając projektu dodajemy do programu okno dialogowe (nowy formularz) wybierając opcje *File->New->Other->Dialogs* i wybierając okno *Standard Dialog* (np. *Standard Dialog (Horizontal)*).



Rysunek 7.8. Standardowe okno dialogowe do zakończenia pracy z programem

Następnie należy w pliku związanym z tym oknem dialogowym (np. OKCANCL2.pas) sprawdzić, jak nazywa się obiekt odpowiadający za okno dialogowe (np. OKRightDlg). Aby umożliwić wywoływanie okienka dialogowego z programu głównego, oraz pobierać z niego informacje, należy dołączyć jego plik nagłówkowy na początku programu związanego z głównym formularzem (np. Unit1.pas), przy pomocy dyrektywy:

```
uses
    ..., OKCANCL2;
```

Akcje należy tak zaprogramować, aby wywołane okno dialogowe było w trybie modalnym (*ShowModal*), oczekujące na naciśnięcie któregoś przycisku. Okno dialogowe będzie zwracało wartość modalną *ModalResult* informującą, który przycisk został wciśnięty. Akcję związaną z zamykaniem aplikacji przez okno modalne należy obsłużyć wystąpieniem zdarzenia *OnExecute*

```
procedure TForm1.acnKoniecExecute(Sender: TObject);
begin
    OKRightDlg.ShowModal;
    if (OKRightDlg.ModalResult=mrOk) then
        //zamkniecie formularza glownego
        self.Close();
end;
```

ustawienie własności

Caption: Koniec
ImageIndex: 2
ShortCut: Ctrl+Q

Skojarzenie komponentów z akcjami

Po zakończeniu projektowania akcji należy skojarzyć je z odpowiednimi komponentami poprzez ustawienie własności `Action` wybranego komponentu na daną akcję. W ten sposób zostaną przekazane ich własności bez potrzeby projektowania obsługi zdarzeń dla każdego komponentu oddzielnie.

W projekcie na formularzu głównym znajdują się dwa przyciski. Pierwszy z nich (`btnDane`) pozwala wyświetlić na odpowiednich etykietach określone przez użytkownika informacje o dodawanym czasopiśmie. Drugi (`btnDodaj`) pozwala dodać nowy element do zbioru czasopism które wyświetlane jest na drugim formularzu. Drugi przycisk jest blokowany tak, aby można było dodać nowe elementy tylko po potwierdzeniu ich przez pierwszy przycisk.

Na początku należy określić zmienne wykorzystywane w aplikacji. Z każdym formularzem są związane określone zmienne, które można dodać jako pola publiczne (`public`) odpowiedniego formularza. Przechowują one informacje o danym czasopiśmie, oraz informacje o ilości wszystkich czasopism w zbiorze.

Formularz główny (Form1, plik: Unit1.pas)

```
public {user declarations}
    czasopismo_tytul:AnsiString; //zmienna przechowująca tytuł czasopisma
    czasopismo_rodzaj:AnsiString; //zmienna przechowująca rodzaj czasopisma
    czasopismo_dostepnosc:AnsiString; //zmienna przechowująca informacje o
dostępności czasopisma
```

Formularz z wizualizacją (Form2, plik Unit2.pas)

```
public
{ Public declarations }
    iloscCzasopism:integer; //zmienna przechowująca informacje o ilości
czasopism
```

W aplikacji ważna jest procedura wypisująca dane na panelu (formularz główny). W tym celu należy obsłużyć zdarzenie `OnClick` przycisku `btnDane`

```
procedure TForm1.btnDaneClick(Sender: TObject);
begin
    //zapisanie informacji o aktualnym czasopiśmie
    self.czasopismo_tytul:='nazwa:'+self.edtTytul.Text;
    self.czasopismo_rodzaj:='rodzaj:'+self.cbRodzaj.Text;
    if (self.rgDostepnosc.ItemIndex=0) then
        self.czasopismo_dostepnosc:='dostepnosc:dostepny'
    else
        self.czasopismo_dostepnosc:='dostepnosc:niedostepny';

    //wyświetlenie informacji o danym czasopiśmie
    self.lblTytul.Caption:=self.czasopismo_tytul;
    self.lblRodzaj.Caption:=self.czasopismo_rodzaj;
    self.lblDostepnosc.Caption:=self.czasopismo_dostepnosc;
```

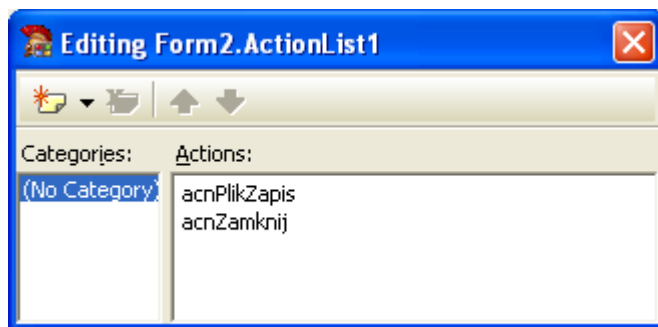
```
//Aktywacja przycisku btnDodaj
self.btnDodaj.Enabled:=true;
end;
```

Procedura dodająca dane do zbioru danych (formularz główny)

Przyjmując, że przycisk `btnDodaj` aktywuje się tylko na czas wpisywania kolejnego czasopisma do zbioru czasopism, należy na początku ustawić jego własność `Enabled` na `false`. Procedura obsługująca zdarzenie `onClick` tego przycisku jest następująca

```
procedure TForm1.btnDodajClick(Sender: TObject);
var s:AnsiString;
//s to zmienna przechowująca liniijkę do wpisania do memCzasopisma
begin
s:=self.czasopismo_tytul+', '+self.czasopismo_rodzaj+', '+self.czasopismo_dos
tepnosc;
//wpisanie liniijki do memCzasopisma na formularzu do wizualizacji
Form2.memCzasopisma.Lines.Add(s);
//wpisanie informacji o ilości czasopism na pasku stanu
//formularza do wizualizacji
Form2.StatusBar1.Panels.Items[1].Text:=IntToStr(Form2.memCzasopisma.Lines.C
ount);
//deaktywacja przycisku btnDodaj
self.btnDodaj.Enabled:=false;
end;
```

Następnie zostaną zaprojektowane akcje dla formularza wizualizacji. Akcje zapisuje się w komponencie `ActionList`. Należy kliknąć na komponent `ActionList` i utworzyć akcje obsługujące funkcje programu (np. o nazwie `acnWizualizacja`). Zaletą akcji jest możliwość definiowania dodatkowych własności które będą przejmowane przez komponenty: napis - `Caption`, skrót klawiszowy - `Shortcut`, numer obrazka - `ImageIndex`. Z wykorzystaniem Edytora Akcji komponentu `ActionList` (dwukrotne kliknięcie na komponent otwiera edytor akcji) utworzyć następujące akcje.



Rysunek 7.7. Edytor Akcji (ActionList)

W przypadku projektowanej aplikacji należy utworzyć 2 akcje dla formularza wizualizacji:

- `acnPlikZapis` - Procedura do zapisywania informacji o czasopismach do pliku tekstowego
- `acnZamknij` - Procedura zamykająca formularz

Procedura do zapisywania informacji o czasopismach do pliku tekstowego (`acnPlikZapis`)

Procedura ta jest wywoływana z menu głównego formularza do wizualizacji poprzez opcję "Plik -> Zapisz do pliku" i poprzez przycisk na pasku szybkiego dostępu. Wykorzystuje ona okienko dialogowe do wyboru pliku do

zapisu (komponent SaveDialog). Ponieważ procedura ta wywoływana jest przez dwa komponenty można utworzyć dla niej odpowiednią akcję. Postać tej procedury jest następująca:

```
procedure TForm2.acnPlikZapisExecute(Sender: TObject);
begin
  //wybór pliku do zapisu i zapis do pliku
  if (self.SaveDialog1.Execute()) then
    self.memCzasopisma.Lines.SaveToFile(self.SaveDialog1.FileName);
  //wyświetlenie informacji o pliku w którym zapisano dane w pasku stanu
  self.StatusBar1.Panels.Items[0].Text:=self.SaveDialog1.FileName;
end;
```

ustawienie własności

```
Caption: Zapisz do pliku
ImageIndex: 0
Shortcut: Ctrl+S
```

Dodatkowo warto ustawić odpowiedni filtr okna SaveDialog, który pozwoli wybrać tylko pliki typu tekstowego, z rozszerzeniem .txt (patrz rozdział: **Okno dialogowe do pracy z plikami**)

Procedura zamykająca formularz do wizualizacji (acnZamknij)

Procedura zamykająca formularz do wizualizacji wywoływana jest przez opcję menu głównego tego formularza, przez przycisk szybkiego dostępu i przez przycisk na formularzu. Procedura może być zrealizowana w postaci akcji. Ma ona następującą postać

```
procedure TForm2.acnZamknijExecute(Sender: TObject);
begin
  self.Close;
end;
```

Należy zwrócić uwagę że self odnosi się w tym przypadku do formularza do wizualizacji a nie całej aplikacji

ustawienie własności

```
Caption: Zamknij
ImageIndex: 1
Shortcut: Ctrl+E
```

Procedura do czyszczenia informacji o czasopismach (formularz do wizualizacji)

Procedura czyszcząca komponent memCzasopisma w którym zapisywany jest zbiór czasopism wywoływana jest z menu kontekstowego (komponent PopUpMenu skojarzony z komponentem memCzasopisma) jako opcja "czyszczenie pola edycyjnego".

```
procedure TForm2.Czyszczeniepolaedycyjnego1Click(Sender: TObject);
begin
  self.memCzasopisma.Clear();
  self.StatusBar1.Panels.Items[1].Text:='0';
end;
```

Skojarzenie komponentów z akcjami

Po zakończeniu projektowania akcji należy skojarzyć je z odpowiednimi komponentami poprzez ustawienie własności Action wybranego komponentu na daną akcję. W ten sposób zostaną przekazane ich własności bez potrzeby projektowania obsługi zdarzeń dla każdego komponentu oddzielnie.