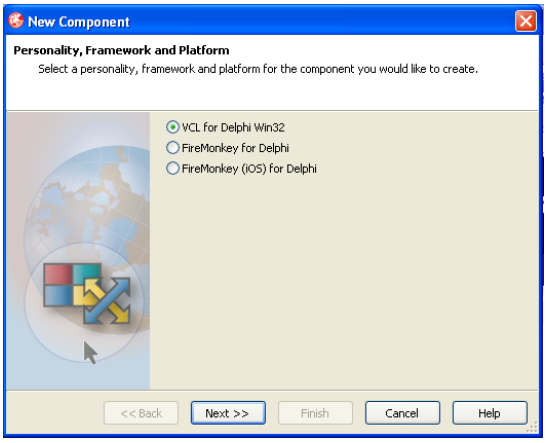
Tworzenie własnych komponentów

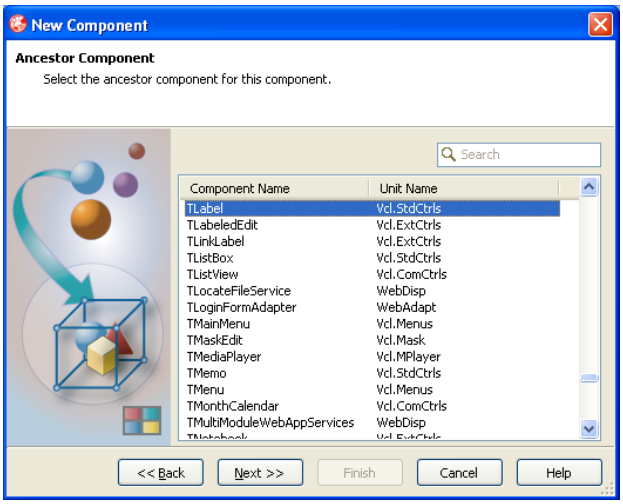
1. Tworzenie nowego komponentu

W tym celu należy wykorzystać menu *Component*. Interesujące są dwie opcje menu *New Component* i *Install Component*. Pierwsze polecenie służy do stworzenia nowego projektu komponentu, natomiast za pomocą drugiego możemy zainstalować już istniejący komponent. Po wybraniu tej opcji na palecie komponentów zostanie utworzona nowa ikonka i wówczas komponent będzie działał jak zwykły komponent VCL.

Aby utworzyć nowy komponent z menu *Component* wybierz *New Component*. Na ekranie zobaczysz okienko, jakie zostało przedstawione na poniższym rysunku:

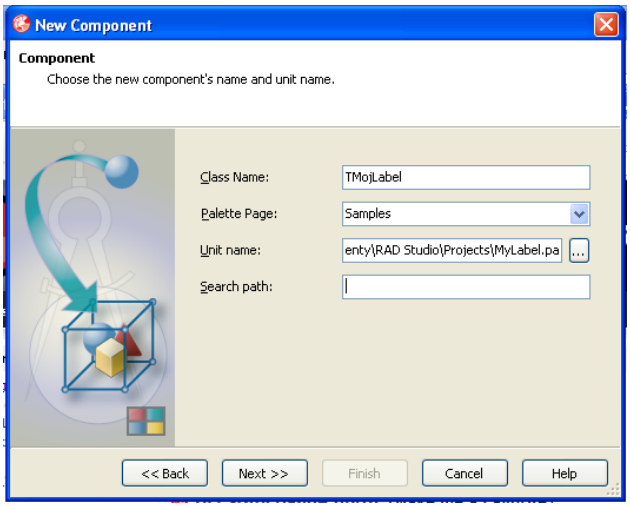


Należy dokonać wyboru „VCL for Delphi Win32” i kliknąć „Next”. Na ekranie zobaczysz okienko, jakie zostało przedstawione na poniższym rysunku:

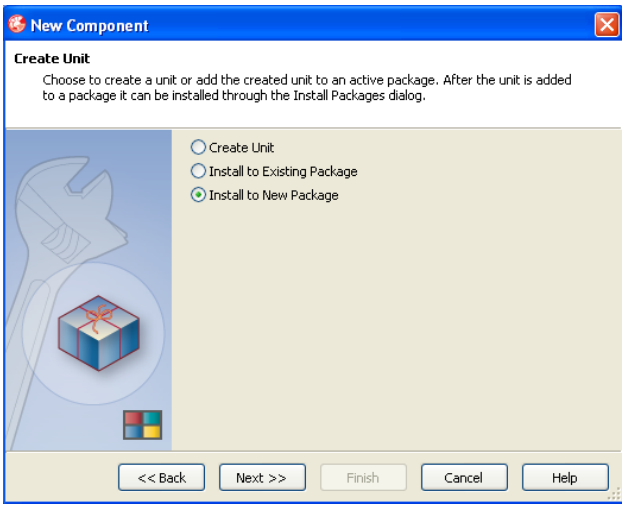


Z listy należy wybrać komponent, który będzie stanowił klasę bazową dla naszego nowego obiektu, np. klasę `TLabel`. Jeżeli chcesz tworzyć komponent od początku, wybierz pozycję `TComponent`. Jest to bazowa klasa dla wszystkich komponentów, zawierająca parę potrzebnych procedur i funkcji.

Kliknij „Next”, a na ekranie zobaczysz okienko, jakie zostało przedstawione na poniższym rysunku:



W polu *Class Name* należy wpisać nazwę nowego komponentu. Pamiętaj, że wedle obowiązującej zasady należy poprzedzić nazwę literą T. Pole *Palette Page* określa, na jakiej palecie zostanie zainstalowany komponent. *Unit file name* to ścieżka do katalogu, w którym zostaną zapisane nowe pliki z komponentem. Ostatnie pole określa ścieżkę do katalogu, gdzie znajdować się będą potrzebne do kompilacji pliki komponentu. ${DELPHI} oznacza domyślną ścieżkę do pliku programu Delphi.
Kliknij „Next”, a na ekranie zobaczysz okienko, jakie zostało przedstawione na poniższym rysunku:



Wybierz opcję „Install to New Package”. Komponent znajdujący się w pakiecie będzie od razu gotowy do instalacji (począwszy od wersji 2005, Delphi instaluje komponenty tylko z pakietów).
Po wykonaniu powyższych operacji w edytorze kodu pojawi się następujący kod:

```
unit MyLabel;

interface

uses
  System.SysUtils, System.Classes, Vcl.Controls, Vcl.StdCtrls;
```

```
type
  TMojLabel = class(TLabel)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
  end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('Samples', [TMojLabel]);
end;

end.
```

Kluczowe znaczenie ma procedura `Register`. Następuje w niej zarejestrowanie komponentu realizowane przez polecenie `RegisterComponents`. Pierwszy parametr oznacza nazwę palety. Drugi parametr to nazwy komponentów do zainstalowania. Muszą one być wpisane w nawiasie kwadratowym, gdyż w tym wypadku występują jako elementy tablicy.

2. Konstruktory i destruktory:

Komponenty, podobnie jak zwykłe klasy, posiadają konstruktory oraz destruktory. Jeśli ich nie zadeklarujemy to komponent będzie posiadał domyślny konstruktor z klasy bazowej.

```
public
    constructor Create(AOwner : TComponent); override;
    destructor Destroy; override;
```

Zarówno konstruktor, jak i destruktor są opatrzone klauzulą `override`, co znaczy, że są przedefiniowane.
W konstruktorze możemy zainicjować właściwości wartościami domyślnymi. W destruktorze należy zwolnić pamięć, jeśli została przydzielona w konstruktorze.

3. Właściwości

Właściwości komponentów służą do przechowywania wartości różnych typów. Jednak jeśli chcemy, aby właściwości były widoczne w Inspektorze Obiektów, należy użyć kolejnej sekcji w klasie, sekcji `published`:

```
property CzasMigania: Integer read FCzasMigania write UstawCzas
default 200;
```

Właściwość należy oznaczyć słowem kluczowym `property`. Zwykło się także definiować zmienne pomocnicze, których deklaracje umieszcza się w sekcji `private`. Nazwy tych zmiennych umieszczane są po słowach kluczowych `read i write`. W ten sposób łatwo

utworzyć jakąś właściwość tylko do odczytu (pozostawiając jedynie klauzulę `read`) lub tylko do zapisu (klauzula `write`), albo też zarówno do zapisu, jak i do odczytu (zarówno `read`, jak i `write`).

Powyższy kod oznacza, że właściwość `CzasMigania` będzie typu `Integer`, a jej odczytanie będzie równoważne z odczytaniem danych ze zmiennej `FCzasMigania`. Natomiast w przypadku, gdy użytkownik zechce przypisać jakąś wartość właściwości `CzasMigania`, wywoła procedurę `UstawCzas`. Nowa wartość zostanie przekazana do procedury `UstawCzas` w parametrze `AMiganie`.

4. Definiowanie własnych zdarzeń

Zdarzenia służą do zaprogramowania określonej reakcji w momencie zajścia jakiegoś wydarzenia (np. kliknięcia myszą). Podczas projektowania własnych komponentów często trzeba będzie przypisywać w kodzie programu jakieś zdarzenie, tzw. procedurę zdarzeniową. Procedura zdarzeniowa musi mieć określone parametry, zależne od rodzaju zdarzenia. Przykładowo procedura zdarzeniowa dla zdarzenia `OnClick` musi mieć parametr `Sender` typu `TObject`.
Czasem może przytrafić się sytuacja, gdy w naszym komponencie będzie wymagane zdarzenie zawierające więcej niż jeden parametr, będzie to umożliwiało użytkownikowi przekazanie jakichś funkcji. W takim wypadku konieczne stanie się zadeklarowanie dodatkowo nowego typu zdarzenia:

```
type
  TMyEvent = procedure(Sender: TObject; X : Integer) of object;
```

Taka składnia, tj. umieszczenie frazy `of object` na końcu jest obowiązkowa. Teraz oprócz zwykłego parametru `Sender` zdarzenie będzie posiadać także parametr *X*.

5. Przykład:

Ogólny zarys klasy:

```
type
  TMojLabel = class(TLabel)
  private
    { Private declarations }
    FMiganie : Boolean;
    FCzasMigania : Integer;
    FTimer : TTimer;
  protected
    { Protected declarations }
    procedure MiganieWlaczone(AMiganie: Boolean);
    procedure OnTimer(Sender: TObject);
    procedure UstawCzas(AMiganie: Integer);
  public
    { Public declarations }
    //ovveride oznacza, ze sa przedefiniowane
    constructor Create(AOwner: TComponent);override;
    destructor Destroy;override;
  published
    { Published declarations }
```

```
  {jesli chcemy, zeby wlasciwosci byly widoczne w Inspektorze Obiektow}
  property Miganie: Boolean read FMiganie write MiganieWlaczone default
    true;
  property CzasMigania: Integer read FCzasMigania write UstawCzas default
    200;
end;
```