Dialogi, Memo i Image

Modified – właściwość typu Boolean. Jest **automatycznie** ustawiana na True każdorazowo po dokonaniu zmian treści wpisanej w Memo.

Komponenty **Dialog**:

SaveDialog, OpenDialog:

Metoda: Execute - wyświetla okienko dialogowe zwraca informację logiczną o rezultacie dialogu z użytkownikiem

Właściwości: FileName - nazwa pliku ustalona w wyniki dialogu
Filter
DefaultExt - domyślne rozszerzenie (typ) pliku

Filter (przykład ustawienia):

Nazwa filtru	Rozszerzenie
Pliki źródłowe (*.pas)	*.pas;*.~pa
Wszystkie pliki (*.*)	*.*

Znak rozdzielający przy podawaniu wielu dopuszczalnych rozszerzeń – średnik ";"

Komponent Memo i typ TStrings

Komponent Memo służy do prezentacji i edycji informacji tekstowej uporządkowanej w wierszach. Własność komponentu o nazwie:

Text

zawiera główny zasób komponentu - edytowany tekst. Pozostałe własności (a także metody) komponentu służą do organizacji dostępu do tego głównego zasobu, np.:

Clear - kasowanie zawartości komponentu.

Modified – właściwość typu Boolean. Jest **automatycznie** ustawiana na True każdorazowo po dokonaniu zmian treści wpisanej w Memo.

Wśród pozostałych własności najważniejsza jest:

Lines - typu TStrings

Typ TStrings zawiera składowe - własności i metody, które stanowią jednolity interfejs do tekstu zorganizowanego w postaci listy łańcuchów. Typ ten nie zawiera jednak sam w sobie zasobu (tekstu), którym zarządza. Z tego względu zasadą jest, ze typ ten jest wykorzystywany tylko jako typ składowej innego obiektu (który to obiekt jest wyposażony bezpośrednio w zarządzany tekst).

Np. dla utworzenia własnego obiektu zawierającego kolekcje łańcuchów należy wykorzystywać typ TStringList - typ ten bardzo podobny do TStrings zawiera kolekcję łańcuchów.

Najważniejsze własności komponentu Lines:

Count - liczba łańcuchów (wierszy)
Strings - tablica odwzorowująca edytowany tekst w podziale na wiersze indeksowana od 0.

Memo1.Lines.Strings[1] i Memo1.Lines[1]

to dwa równoważne odwołania tak samo skutkujące.

Metody:

function **Add**(const S: String): Integer;

Wstawia S na koniec; zwraca indeks elementu po wstawieniu.

procedure **Insert**(I: Integer; Const S: Strind);

Wstawia S na miejsce I. Pozostałe wiersze są odpowiednio przesuwane.

Polecenia:

```
Memo1.Lines.Add('Nowy')  i
Memo1.Lines.Insert(-1, 'Nowy')
```

są równoważne; -1 może być zastąpione wartością Memo1.Lines.Count.

procedure **AddStrings**(S: TStrings); - dodaje listę łańcuchów.

procedure **Delete**(I: Integer) - usuwa i-ty wiersz

Np:

```
for i:=1 to 5 do
    Memo1.Lines.Delete(Memo1.Lines.Count-1)
```

usuwa 5 ostatnich wierszy tekstu.

```
for i:=3 to 5 do
    Memo1.Lines.Delete(i)
```

powyższy zapis jest błędny - w momencie usuwania wiersza o numerze 5 tego wiersza juz nie ma !!!!

procedure **Clear** - usuwa wszystko

procedure **Move**(I1, I2: Integer) - przesuwa łańcuch z I1 na I2

procedure **Exchange**(I1, I2: Integer) - zamienia miejscami

function **IndexOf**(const S: String):Integer - odszukuje wiersz zawierający tekst S i zwraca jego indeks (lub -1 gdy brak)

procedure LoadFromFile(Const FileName: String);

odczytuje plik o nazwie podanej w parametrze i ładuje wierszami. FileName może zawierać ścieżkę dostępu.

procedure **SaveToFile**(Const FileName: String);

zapisuje tekst na plik. Jeśli pliku nie ma, to jest tworzony.

Własna Lista Łańcuchów

jest tworzona z wykorzystaniem typu TStringList. Np.

```
Private
  Moja: TStringList;
```

Obiekt trzeba zainicjować (utworzyć). Np w momencie uruchomienia aplikacji tj. w czasie zdarzenia OnCreate dla formatki głównej:

```
Procedure TForm1.FormCreate(Sender: TObject);
Begin
Moja:=TStringsList.Create;
End;
```

Dodawanie wierszy tekstu można zorganizować np w zdarzeniu OnClick dla jakiegoś przycisku:

```
Procedure TForm1.Button1.Click(Sender: TObject);
Begin
Moja.Add(Edit1.Text);
End;
```

W powyższej procedurze wykorzystujemy własność Strings która jest typu TStrings.

Przed zakończeniem aplikacji trzeba pamiętać o skasowaniu wygenerowanego obiektu Moja. Np w czasie obsługi zdarzenia OnDestroy dla formatki głównej:

```
Procedure TForm1.FormDestroy(Sender: TObject);
begin
Moja.SaveToFile('Wiersze.Txt');
Moja.Free;
End;
```

Inne własności w typie TStringList to

Sort - metoda porządkująca wiersze (łańcuchy) alfabetycznie

Sorted - własność sprawiająca ze metoda Add dodaje wiersze na koniec zgodnie z uporządkowaniem.

Obiekty typu TStringList posiadają (generują) tez zdarzenie:

OnChange - następuje z chwila zmiany obiektu

dodatkowo zdarzenie OnChanged w chwili rozpoczynania zmiany.

Z obsługą tego zdarzenia jest pewien problem. Obiekt typu TStringList, nie jest komponentem, jest tworzony i obsługiwany całkowicie w tzw. Run-Time, a jego składowe oczywiście nie występują w Inspektorze obiektów. Tak wiec obsługę zdarzenia trzeba zorganizować inaczej. Należy skorzystać z faktu, ze zdarzenia są definiowane jak specyficzne własności np:

property OnChange: TNotifyEvent;

gdzie: TNotifyEvent = procedure (Sender: TObject) of object;

A więc zdarzenie jest własnością typu proceduralnego.
Obsługa tych zdarzeń nie będzie tu opisana.

Komponent Image

Właściwości komponentu **Image** (ale też komponentu **Form**):

- **Width** - szerokość komponentu,
- **Height** - wysokość komponentu,
- **Canvas** - zawartość komponentu. Właściwość typu obiektowego **TCanvas** grupująca właściwości m.in.:

  - **Brush** - właściwość typu obiektowego **TBrush** grupująca cechy tła komponentu m.in. cechę **Color**,

  - **Pen** - właściwość typu obiektowego **TPen** grupująca cechy linii kreślonych na komponencie,

  oraz metody m.in.:

  - Procedure **MoveTo**(X, Y: Integer); - umieszczenie kursora graficznego w punkcie X, Y
  - Procedure **LineTo**(X, Y: Integer); - wykreślenie linii do punktu X, Y.

Przykładowa procedura rysowania ramki:

procedure TForm1.FormCreate(Sender: TObject);
begin

```
with Image1, Image1.Canvas do
  begin
  LineTo(Width-1, 0);
  LineTo(Width-1, Height-1);
  LineTo(0, Height-1);
  LineTo(0, 0);
  end;
end;
```

Układ współrzędnych obiektu Canvas:

Właściwości **Pen** oraz **Brush** można wykorzystać do ustalenia kolorów kreślonych linii i tła:

```
Pen.Color:= clRed;
Brush.Color:=clBlue;
```

Kasowanie wykreślanych linii poprzez wypełnianie obszaru tłem.
Kolor tła jest wykorzystywany w czasie wypełniania prostokątnych obszarów metodą:

```
procedure FillRect(const Rect: TRect);
```

Parametr Rect jest typu rekordowego:

```
type
        TPoint = record
                  X: Longint;
                  Y: Longint;
                  end;

        TRect = record
                 case Integer of
                   0: (Left, Top, Right, Bottom: Integer);
                   1: (TopLeft, BottomRight: TPoint);
                 end;
```

Przykład zastosowania metody:

```
        Image1.Canvas.FillRect(Rect(0,0,100,100));
```

Powyższe spowoduje namalowanie kwadratu 0 wierzchołkach: (0, 0) i (100, 100) w bieżącym kolorze tła.

Standardowa funkcja **Rect:**

function Rect(ALeft, ATop, ARight, ABottom: Integer): TRect;

pozwala zamienić 4 parametry całkowite na odpowiedni rekord.

W przypadku kolorowania całego obszaru Canvas można wykorzystać właściwość ClientRect.

Właściwość ClientRect komponentu Image zwraca rekord zawierający wymiary komponentu.

Nowa postać procedury rysowania ramek:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
with Image1, Image1.Canvas do
  begin
  Pen.Color:= clRed;
  Brush.Color:=clBlue;
  FillRect( ClientRect);
  LineTo(Width-1, 0);
  LineTo(Width-1, Height-1);
  LineTo(0, Height-1);
  LineTo(0, 0);
  end;
end;
```