Začínáme s balíkem knihoven VizLib

ZAČÍNÁME PROGRAMOVAT UŽIVATELSKÉ ROZHRANÍ PRO TERMINÁLY FIRMY SOFCON

Příručka uživatele a programátora

***SofCon*®spol. s r.o.**
Střešovická 49
162 00 Praha 6
tel/fax: +420 220 180 454
E-mail: sofcon@sofcon.cz
www: http://www.sofcon.cz

Verze dokumentu 00.99

Datum vydání: 10.09.2004
Datum posledního uložení dokumentu: 10.09.2004
(Datum vydání a posledního uložení dokumentu musí být stejné)

Obsah :

1. O dokumentu

1.1. Revize dokumentu

Verze dokumentu	Verze SW	Autor	Datum vydání	Popis změn
00.90	01.XX	Bin	20.02.2004	Pre-release
00.99	01.XX	Bin	10.09.2004	Pre-release 2

1.2. Účel dokumentu

Tento dokument slouží jako návod k programování za použití programového balíku vizualizačních knihoven VizLib používaného k vizualizacím na terminálech firmy SofCon s.r.o.

1.3. Rozsah platnosti

Určen pro programátory a uživatele programového vybavení SofCon.

1.4. Související dokumenty

Pro čtení tohoto dokumentu není potřeba číst žádný další manuál, ale je potřeba se orientovat v používání programového vybavení SofCon.

Tato učebnice obsahuje odkazy na refenční příručky jednotlivých jednotek vizualizačních knihoven (Controls, GrCtrls, IoDdrv, Bitmaps, Fonts a dalších).

Popis formátu verze knihovny a souvisejících funkcí je popsán v manuálu LibVer.

2. Termíny a definice

Používané termíny a definice jsou popsány v samostatném dokumentu Termíny a definice. Nově vytvořené používané termíny a definice jsou voleny tak, aby byly dostatečně výstižné pro programátory PC, kteří již přišli do styku (alespoň jako uživatelé) s nějakým (grafickým) vizualizačním prostředím (MS-Windows, TurboVision, X-Windows …)

3. Velmi jednoduchý program

3.1. Jednoduchá aplikace

Pro seznámení zkusíme velice jednoduchou aplikaci. Je určena k běhu na počítači PC. (Ke zdárnému spuštění programu je třeba mít nainstalován ovladač myši.) Zdrojový kód hlavního programu vypadá takto:

```
uses
  objects,
  iodrv, PCDrv,
  controls, grctrls,
  PCMCus,
  Fonts, Foss16;

var
  R  : TRect;
  ap : PApplication;
  pg : PPage;
  st : PStaticText;
begin
  ap := new ( PApplication, init(
          new( PInputDriver, init(
            new( PPCKeybDriver, init ),
            new( PPCMouseDriver, init )
          )),
          new( PVgaMonoDriver, init( 640, 480 )),
          nil
        ));
  ap^.Customize( ccApplication );
  ap^.SetFont( fidSS16 );

    R.Assign( 0, 0, 320, 240 );
    pg := new( PPage, Init( R ));
    pg^.Customize( ccPage );

      R.Assign( 1, 1, 319, 18 );
      st := new( PStaticText, init( R, 'Ahoj' ));
      st^.Customize( ccStaticText );
      pg^.Insert( st );

    ap^.Insert( pg );

  ap^.Run;
  dispose( ap, done );
end.
```

Běžící program se ukončuje klávesovou zkratkou Alt-X.
Tento soubor je dodáván v knihovnách v adresáři VIZ\Examples\EX01\
Můžete se pokusit jej přeložit a spustit, mělo by se vám to podařit.

3.2. „Alenka v říši divů“

Je jisté, že výše uvedený příklad prozatím není zcela srozumitelný a vzhledem k tomu, že jen napíše na obrazovku jeden text, může se zdát příliš složitý. Postupně jak budete pronikat do tajů knihoven, bude vám vše čím dál tím více jasné.

4. Objekty vizualizačních knihoven

Knihovny jsou rozděleny do několika jednotek.

- První skupinu tvoří obecné ovladače zařízení – objekty poskytující metody pro vstup a výstup, případně další pomocné jednotky (např. pro kreslení bitmap a textu – jednotky IoDrv, Fonts, Bitmaps) – těmi se zabývat nebudeme, neboť to jsou jen spodní vrstvy.
- Druhou skupinou jsou specifické ovladače konkrétních terminálů fy. SofCon s.r.o. (Term10, Touch11, Touch33, PC, …) – to jsou konkrétní implementace spodních vrstev.
- Třetí skupinou je univerzální objektová stavebnice. Jedná se o soubor událostmi řízených rozšiřitelných grafických komponent (jednotky Controls a GrCtrls). S nimi se v tomto návodu seznámíte.

4.1. Komponenty

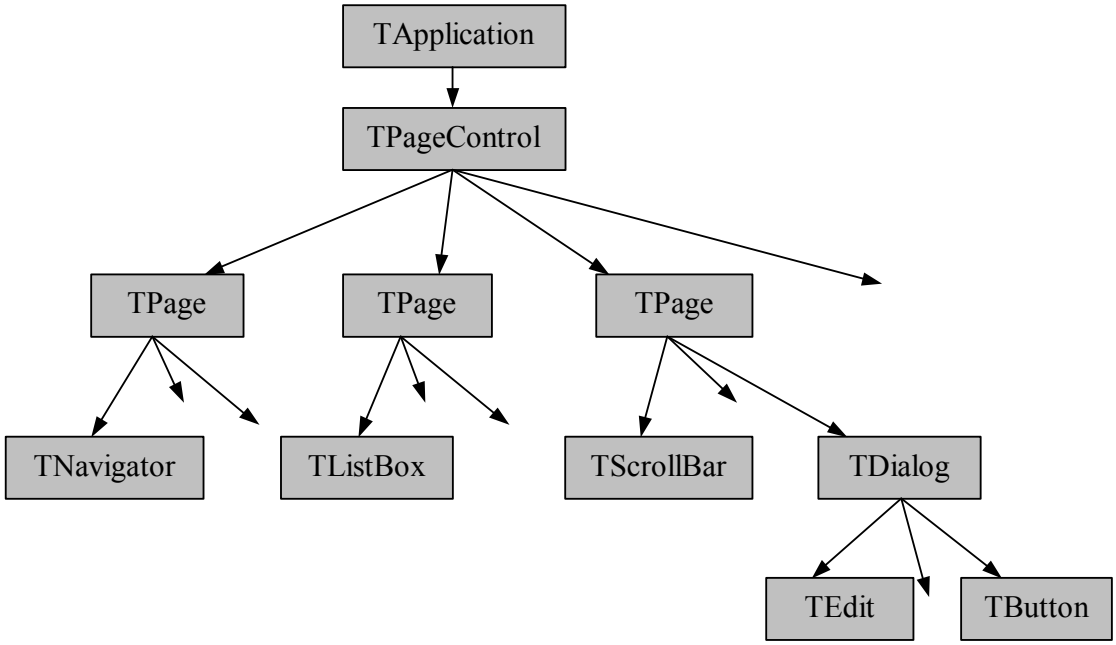
Komponenta je základní stavební prvek vizualizačního systému. Komponentou je například editační řádka, posuvná lišta, tlačítko apod. Obecně lze říct, že komponenta je instance třídy, která popisuje chování definované obdélníkové oblasti na displeji. Komponenta předepisuje co a jak bude v dané oblasti vykresleno a jak bude daná oblast reagovat na vstupní události přicházející od uživatele.
Všechny komponenty vycházejí z abstraktní třídy **TControl**. Třída **TControl** definuje standardní chování komponent a poskytuje rozhraní pro snadný návrh komponent a snadnou manipulaci s komponentami z uživatelské aplikace.
Velmi jednoduché komponenty, které budou často používány v demonstračních příkladech jsou statický text (**TStaticText**), či tlačítko (**TButton**), případně editační okénko (**TEdit**).

4.2. Skupiny komponent

Komponenty nemohou existovat izolovaně. Vždy jsou vloženy do seznamu vlastníka. Vlastník komponent je speciální komponenta – skupina – **TGroup** (odvozená z komponenty **TControl)**.
Třída **TGroup** a její potomci obsahují seznam komponent do nich vložených. Umožňují vkládání komponent do skupiny (metodou **Insert**) a odstraňování komponent ze skupiny (metodou **Delete**).
Ze třídy **TGroup** jsou odvozeny např. komponenty typu okno (**TWindow**), stránka (**TPage**), správce stránek (**TPageControl**) nebo třída zastřešující celou aplikaci (**TApplication**).
Pro demonstrační účely bude jako skupinová komponenta nejčastěji využívána stránka (**TPage**), případně okno (**TWindow**).

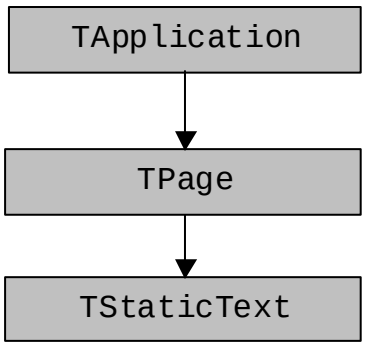
4.3. Hierarchie stránek, správce stránek, strom komponent

Display nebývá dostatečně veliký na zobrazení všech možných/potřebných informací. Z tohoto důvodu vznikla potřeba rozdělení celé vizualizace na více „stránek“ (**TPage**) a systém přeskakování mezi těmito „stránkami“. Stránky jsou vytvořeny a vloženy do jednoho objektu – „správce stránek“ (**TPageControl**). Tento správce stránek je odpovědný za to, kterou stránku v daném okamžiku na terminálu vidíme. Stránky obsahují jednotlivé komponenty. Správce stránek je vložen do aplikace (**TApplication**). Typický příklad relace „vlastnictví“ je zobrazen na následujícím obrázku.



Kořenem stromu je nutně vždy aplikace (**TApplication**). Do této aplikace bývá vložen správce stránek(**TPageControl**). Správce stránek obsahuje jednotlivé stránky(**TPage**) a tyto stránky obsahují jednotlivé komponenty/skupiny komponent.

4.4. Strom (vlastnictví) komponent v příkladu EX01

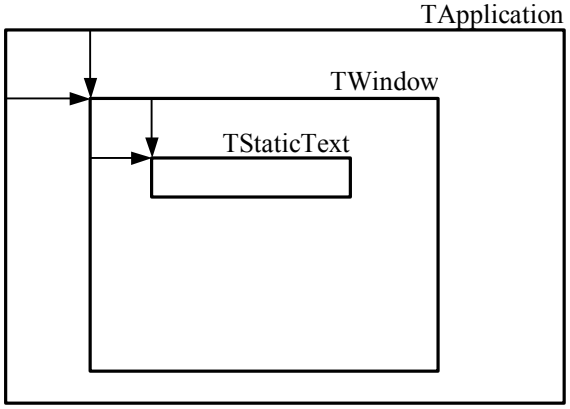


Z obrázku je zřejmé, že se jedná o velmi jednoduchý demonstrační příklad

4.5. Vytvoření komponent

4.5.1. Umístění a velikost komponent

Každá (i skupinová) komponenta (s výjimkou aplikace) zabírá určité obdelníkové místo. Tato oblast je zadávána jako první parametr konstruktoru komponenty (tento první parametr je vždy typu **TRect**). Veškeré polohy jsou udávány vůči levému hornímu rohu vlastníka, do něhož budou vloženy.



Většina obdelníkových údajů je uváděna tak, že levý a horní okraj zahrnují, zatímco pravý a dolní okraj nezahrnují. Vytvoříme-li tedy obdelník pomocí:
`R.Assign( 0, 0, 10, 5 );` bude zabírat `10 x 5` pixelů z bodu `[0,0]` do bodu `[9,4]`.

4.5.2. Vytvoření komponenty

Instanci komponenty vytvoříme rozšířeným voláním funkce **`new`** o jméno typu a jméno konstruktoru včetně potřebných parametrů konstruktoru.

Příklad:
```
st := new( PStaticText, Init( R, 'Nejaky text' ));
```

4.5.3. Nastavení předdefinovaných vlastností komponenty

Po vytvoření jsou vlastnosti komponenty na implicitních hodnotách. Tyto hodnoty ovšem je obvykle potřeba změnit. Přizpůsobení používanému terminálu, případně další přizpůsobení provádí metoda TControl.`Customize( ccXXXX );`.

4.5.4. Vložení komponenty do skupiny

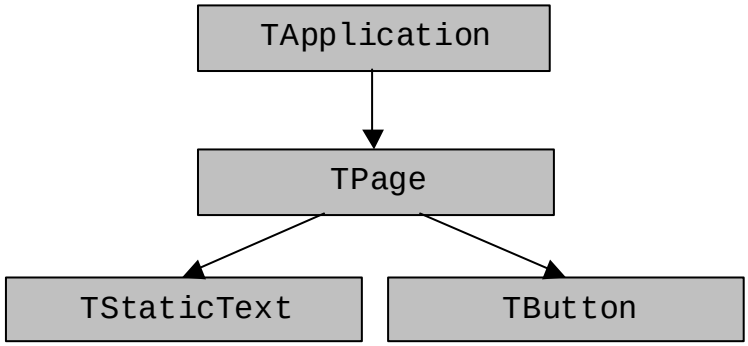
Komponenty jsou vkládány do skupiny voláním metody:
```
TGroup.Insert( komponenta: PControl );
```

4.6. Přidání tlačítka – EX02.pas

Řekněme že chceme přidat do příkladu EX01 tlačítko někam zhruba doprpostřed stránky.
Provedeme:

1) Do lokálních proměnných hlavního programu přidáme
```
bt : PButton;
```
2) Do výkonného kódu za vytvoření a vložení statického textu přidáme následující čtyři řádky:
```
R.Assign( 120, 100, 200, 120 );
bt := new( PButton, Init( R, 'Stiskni', 0, 0 ));
bt^.Customize( ccButton );
pg^.Insert( bt );
```

Tento upravený soubor je dodáván jako příklad EX02.
Vzhledem k tomu, že jsme do stránky přidali další objekt, strom vlastnictví vypadá takto:



4.7. Vytvoření kořene hierarchie vlastnictví – komponenty typu TApplication

Na nejvyšším stupni je komponenta typu TApplication. Přestože se jedná o komponentu, má některé specifické vlastnosti od komponent odlišné. Její hlavní úlohou je „spravovat“ vstupní zařízení (klávesnici a polohovací zařízení) a výstupní zařízení (display).
Konstruktor této komponenty má tři parametry. Prvním z nich je ovladač vstupních zařízení, druhým je ovladač displeje. Třetím parametrem je ukazatel na datovou strukturu obsahující nastavení terminálu (zatím ponecháme na nil).
Vytvoření této nejvrchnější komponenty vypadá takto:

```
ap := new( PApplication, Init(
        new( PInputDriver, Init(
          new( PPCKeybDriver, Init ),
          new( PPCMouseDriver, Init )
        )),
        new( PVgaMonoDriver, Init( 640, 480 )),
        nil
      ));
```

Výše uvedený kus kódu je určen pro počítač PC s klávesnicí, myší a standardním VGA displayem. Pro jednotlivé terminály jsou vytvořeny ovladače vstupních zařízení a displejů (v souborech PCDrv, T10Drv, TPDrv, T11MDrv, T11CDrv, T33MDrv,…).

4.8. Nastavování dalších vlastností komponenty

Mezi vytvořením komponenty a vložením do skupiny je ten správný prostor pro nastavení dalších vlastností komponenty. Například, přejeme-li si mít text „Ahoj“ orámovaný, přidáme na příslušné místo (tedy za st^.Customize(ccStaticText)) tento řádek:

```
st^.SetFlags( stfBorder, 0 );
```

4.9. Fonty – psaní textu

Některé komponenty vypisují na display text. Z tohoto důvodu každá komponenta obsahuje položku font obsahující identifikátor jejího příslušného fontu. Identifikátory fontů jsou konstanty s prefixem fid_. Tyto konstanty určují, který font bude použit při vypisování textu.
Vyjímečné postavení mezi těmito konstantami má hodnota fidDefault (=0). Ta udává, že bude použit font vlastníka (a to opakovaně ve smyslu průchodu nahoru stromem vlastnictví). Z tohoto důvodu je vhodné přiřadit kořeni vlastnictví – komponentě **TApplication** – nejčastěji využívaný font (v příkladě EX01,EX02 je to fidSS16). Tím pádem veškeré komponenty, u nichž explicitně neurčíme font jiný, používají font komponenty **TApplication**.
Aby mohl být font používán, musí být v klauzuli uses uveden správce fontu – jednotka Fonts a soubor s příslušným fontem (v příkladu je to FoSS16).
Pro nastavení fontu je určena metoda TControl.SetFont(fidXXX); . V příkladě je nastaven pouze u aplikace.

4.10. Vykreslování komponent

Každá komponenta má metodu Paint. Tato metoda je odpovědná za vykreslení komponenty. Chceme-li však lehce modifikovat kreslení, není potřeba tvořit následníka, ale pro tyto účely byly do komponent dodány dvě procedurální proměnné – **TControl.BeforePaint** a **TControl.AfterPaint**. Bez nároku na smysluplnost předpokládejme, že vykreslené tlačítko z příkladu 2 chceme mít přeškrtnuté křížkem. V tomto případě vytvoříme proceduru, která po vykreslení tlačítka toto tlačítko „škrtne“. Tuto proceduru uložíme do proměnné AfterPaint a kdykoliv dojde k vykreslení tlačítka, bude též vykresleno škrtnutí. Škrtací procedura vypadá takto:

```
procedure skrtni( AControl: PControl; ACanvas: PCanvas ); far;
var
  r : TRect;
begin
  AControl^.GetExtent( r );
  with ACanvas^ do
  begin
    Pen.Color := clBlack;
    Pen.Style := psSolid;
```

```
    DrawLine( r.a.x, r.a.y, r.b.x, r.b.y);
    DrawLine( r.a.x, r.b.y, r.b.x, r.a.y);
  end;
end;
```

Nyní k tlačítku tuto proceduru přiřadíme – za `bt^.Customize( ccButton );` přidáme řádek:

```
  bt^.AfterPaint := skrtni;
```

Tento příklad je dodáván jako EX03.

4.10.1. Canvas

Veškeré vykreslování je prováděno na „malířské plátno“ – Canvas. To je v knihovnách implementováno jako objekt – **TCanvas**. Tento objekt má různé metody pro vykreslování jednotlivých grafických prvků – v příkladě je uvedena procedura DrawLine.
Canvas může vykreslovat třemi možnými způsoby.

1) čárovou grafiku
2) výplňovou grafiku
3) text.

Pro čárovou grafiku je třeba vhodně nastavit kreslící pero – TCanvas.Pen
Pro výplňovou grafiku je třeba vhodně nastavit kreslící štětec – TCanvas.Brush
Pro vykreslení textu je třeba správně nastavit atributy fontu – TCanvas.Font

Při libovolném vykreslování je možné dále nastavit, operace, která bude provedena mezi původní hodnotou, která byla na display, a novou hodnotou. Tato operace je uložena v proměnné TCanvas.ROP a může nabývat hodnot:

1) **ropCopy** = bude uvažována pouze nová hodnota
2) **ropOr** = bude proveden logický součet barvy nové a barvy staré
3) **ropAnd** = bude proveden logický součin barvy nové a barvy staré
4) **ropAndNot** = logický součin barvy nové a negace barvy staré
5) **ropXor** = non-ekvivalence

4.10.2. Univerzalita

Všimněme si, že kreslení škrtnutí není nijak závislé na komponentě, ke které je přiřazeno. Jinak řečeno – tato škrtací procedura je univerzální a lze použít na libovolné viditelné objekty. Zkuste ji třeba přiřadit též stránce (pg^.AfterPaint := skrtni;).

4.10.3. Individualita

Pokud má vykreslující procedura používat specifické vnitřní proměnné komponenty (tj. ty, které nezdědila od TControl), je potřeba parametr procedury AControl přetypovat na danou komponentu. Viz následující kód:

```
procedure skrtni( AControl: PControl; ACanvas: PCanvas ); far;
var
  r : TRect;
begin
  with PButton( AControl )^ do
  begin
    ... {např. if pressed then ...}
  end;
end;
```

4.10.4. Překreslení komponenty – metoda TControl.Repaint

Přestože každá komponenta má metodu Paint, pokud programátor má v záměru překreslit komponentu, zavolá metodu Repaint. To je nejen z důvodu schování kurzoru a ukazatele myši, ale i správného vyvolání procedur BeforePaint a AfterPaint, případně dalších akcí před překreslením a po nakreslení.

5. Události, událostmi řízené programování

5.1. Událost

Pod pojmem událost (event) máme na mysli jev, který je generován periferním zařízením (tedy například stisknutí klávesy, pohyb myši atp.) Tato událost je v programu použita na řízení jeho činnosti.
Událost je v programu zpracována v proměnné typu **TEvent**. Pro začátek stačí že budete vědět, že uvnitř recordu **TEvent** je položka **Code** znamenající význam události (jedna z konstant evXXX) a nějaké další parametry závislé na typu události. Událost je record (a nikoliv objekt) – tedy není sama o sobě schopna žádné činnosti.

5.2. Zdroje událostí, typy událostí

Možné zdroje událostí jsou:

1) Klávesnice
2) Polohovací zařízení (myš / dotykový panel)
3) Časovač(e)
4) Program

Události od klávesnice jsou zřejmé – stisk tlačítek (**Event.Code** = **evKeyDown**). Parametrem této události je kód stisknuté klávesy.
Myš a dotykový panel jsou v jedné kategorii. Možné události od myši jsou: pohyb (**evMouseMove**), stisk tlačítka (**evMouseDown**), uvolnění tlačítka (**evMouseUp**), dvojklik (**evMouseDbl**), držení (opakování) tlačítka myši (**evMouseRep**). Dotykový panel se chová stejně jako myš s tímto omezením: máme k dispozici pouze jedno tlačítko.
Třetím možným zdrojem událostí je časovač. Tento generuje pouze jedinou událost (**evTimer**) – vypršení časovače. Časovač se používá buď na děje jednorázové, nebo na děje opakované.

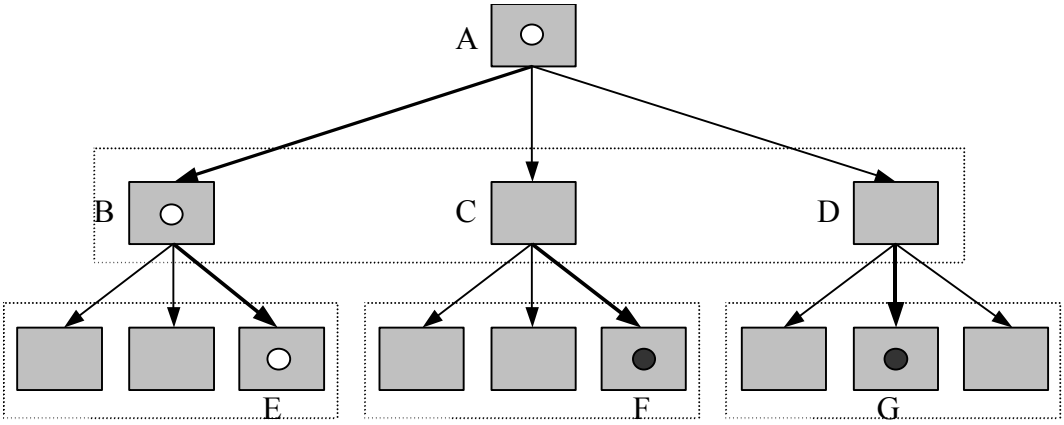
Program může vytvořit událost a vnutit jí vstupnímu ovladači. Tato „uměle vytvořená“ událost je zpracována standardními prostředky, takže je možné z programu simulovat libovolnou událost (stisk klávesy, pohyb myši, …).

Typy událostí:
Události dělíme na několik typů

- události poziční – události, které putují ke komponentě, na jejíž pozici událost vznikla (tedy události od polohovacího zařízení)
- události ohniskové – události které jsou předávány jedné zvolené komponentě ve skupině (předdefinováno - události od klávesnice)
- události ostatní (message a broadcast)
- prázdná událost – pokud vstupní zařízení nemají ve svých vstupních frontách žádnou událost, nebo je již událost zpracována, obsahuje záznam události kód **evNothing**.

5.3. Vybraná komponenta, komponenta v ohnisku

Z hlediska jistého druhu vstupních zařízení (do nichž patří např. klávesnice) je vhodné ve skupinové komponentě určit jednu komponentu, které budou události předávány. Takové komponentě se říká komponenta vybraná (selected). Ve skupině může být nejvýše jedna komponenta vybraná.
Kromě vybraných komponent ještě existují komponenty zaostřené (v ohnisku, focused). Zaostřená komponenta je taková vybraná (selected) komponenta, jejíž vlastník je též zaostřen. (ošklivá definice rekurzí).
Jinak řečeno: máme strom komponent. Kořenem tohoto stromu je proměnná typu **TApplication** (ta je zaostřena vždy) a půjdeme-li z tohoto kořene do vybrané (selected) komponenty, získáme komponentu nejen vybranou, ale i zaostřenou. Takto procházíme strom komponent do hloubky až do té doby, dokud nenajdeme komponentu, která už není skupinová, nebo komponentu, která nemá žádnou svoji komponentu vybranou. Větší jasno pomůže vnést následující obrázek:



Na obrázku výše je schématicky znázorněn strom komponent. Komponenta označená písmenem A na vrcholu stromu je vždy v ohnisku. Pouze jedna z komponent vložených do komponenty A může být vybraná. V tomto případě se jedná o komponentu B. V rámci komponenty B je vybraná komponenta označená písmenem E. **Komponenty A, B, E se nacházejí v ohnisku**. V rámci komponenty C resp. D

jsou vybrány komponenty F resp. G. Komponenty F, G se nenacházejí v ohnisku, protože jejich vlastníci, tj. komponenty C, D se nenacházejí v ohnisku.

Komponenta, která se nachází v ohnisku přednostně zpracována události od klávesnice. Obvykle je komponenta v ohnisku nějak graficky označena, tak aby bylo zřejmé, že stisk klávesy způsobí změnu právě v této komponentě.

U komponent lze tedy rozlišit tři možné stavy:

- Komponenta není vybraná
- Komponenta je vybraná
- Komponenta je vybraná a nachází se v ohnisku

Tyto tři stavy jsou odlišeny příznaky sfSelected a sfFocused v položce State třídy **TControl.**

5.4. Modální stav

Zobrazitelná komponenta si může v určitém okamžiku vyhradit vykonávání všech (vizualizačních) událostí pro sebe. V tom případě ji nazýváme modální komponentou (komponentou v modálním stavu, komponentou s vyhrazeným režimem). Takovou typickou komponentou je například dialogové okno. Je-li totiž aktivní, převezme řízení programu a dokud jej neuzavřeme, nefunguje žádné řízení klávesou ani myší nikde mimo něj. Po spuštění aplikace je v modálním stavu sama aplikace. Modální stav komponenty je navozen zavoláním metody Execute. Tato metoda není volána přímo, ale předání modálního stavu se provádí voláním metody TGroup.ExecControl. Modální stav je indikován příznakem sfModal v položce State třídy **TControl**.

5.5. Zpracování událostí

Komponenty zpracovávají události pomocí metod HandleEvent.

Každou událost vybírá ze vstupního zařízení modální komponenta. Touto komponentou je při spuštění programu komponenta typu **TApplication**.

Zjednodušený popis zpracování událostí:
Veškeré události jsou generovány ve vstupním ovladači (potomku **TInputDriver**). Vygenerovanou událost vybírá komponenta v modálním stavu.

1) Jedná-li se o událost časovače, je předána příslušné komponentě, jíž časovač náleží.
2) Jedná-li se o událost ohniskovou, je postupně předána nejhlouběji zanořené zaostřené komponentě.
3) Jedná-li se o událost poziční, je předána nejhlouběji zanořené právě viditelné komponentě, na níž k události od myši vznikla.
4) Jedná-li se o jinou událost pak:

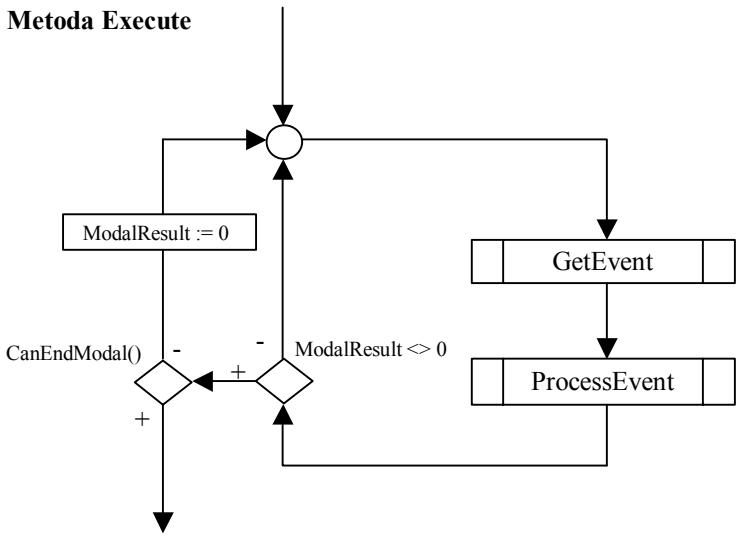
a) pokud se jedná o zprávu (evMessage), je tato zpráva přímo zaslána komponentě, které je určena (podobně jako časovač).
b) pokud se jedná o broadcast (evBroadcast), prochází tato zpráva celým stromem vlastnictví do hloubky. (což je samozřejmě pomalé)

5.6. Cyklus získání a zpracování událostí (modální stav)

Je-li komponenta v modálním stavu, je prováděna její metoda execute. Ta vypadá (většinou) takto:

```
function TGroup.Execute: Word;
var
  E        : TEvent;
begin
  repeat
    ModalResult := 0;
    repeat
      GetEvent( E, False );
      ProcessEvent( E );
    until ModalResult <> 0;
    if not CanEndModal( ModalResult ) then ModalResult := 0;
  until ModalResult <> 0;
  Execute := ModalResult;
end;
```

Metoda Execute



Metoda GetEvent je určena k získání události z vstupního zařízení.
Metoda ProcessEvent je určena k standardnímu vyvolání obsloužení události.

5.7. Standardní zpracování událostí – metoda HandleEvent

Každá komponenta má svou metodu HandleEvent. Ta standardně reaguje na události.

5.8. Události – upravování chování – metoda ProcessEvent (beforeHandle, afterHandle)

Chceme-li reagovat na nějakou událost, nepotřebujeme většinou vytvářet potomka dané komponenty. Zpracování (tj. metoda ProcessEvent) totiž nejprve zavolá uživatelskou obsluhu před zpracováním (procedurální proměnná **TControl.BeforeHandle**), poté standardně zpracuje událost (procedurou HandleEvent) a posléze zavolá uživatelskou obsluhu po zpracování (procedurální proměnná **TControl.AfterHandle**).
Tudíž máme možnost obsloužit událost před standardním obsloužením a též po obsloužení. Standardně tyto procedurální proměnné ukazují na NIL, a tak se žádná speciální obsluha nevolá.

Nadešel čas seznámit se s další primitivní grafickou komponentou. Je jí komponenta ProgressBar – ukazatel průběhu. Byla vybrána právě proto, že zobrazuje veličinu, která se dá měnit. Kromě standardních metod definuje některé nové, například TProgressBar.SetRange – nastavuje povolený rozsah zobrazované veličiny a TProgressBar.SetPosition – nastavuje aktuální hodnotu. Vytvoříme tedy ukazatel průběhu a dvě tlačítka, které budou stav průběhu zvětšovat a zmenšovat.
Do kódu tedy přidáme:

a) inicializovanou proměnnou jménem hodnota

```
const
  hodnota :byte = 27;
```

b) procedury obsluhující události tlačítek

```
procedure zmensi( AControl: PControl; var AEvent: TEvent ); far;
begin
  if AEvent.code = evMouseDown then
  begin
    if hodnota > 0 then dec( hodnota );
    pb^.SetPosition( hodnota );
  end;
end;

procedure zvetsi( AControl: PControl; var AEvent: TEvent ); far;
begin
  if (AEvent.code = evMouseDown) then
  begin
    if hodnota < 100 then inc( hodnota );
    pb^.SetPosition( hodnota );
  end;
end;
```

c) proceduru vykreslující procentuální stav průběhu

```
procedure pbpaint( AControl: PControl; ACanvas: PCanvas ); far;
var
  r : TRect;
  s : string;
begin
  with PProgressBar( AControl )^ do
  begin
    GetExtent( R );
    ACanvas^.ROP := ropXOR;
    str( hodnota, s );
    ACanvas^.DrawTextRect( R, s+'%', tfCenterX or tfCenterY );
  end;
end;
```

d) do hlavního programu přidáme kód vytvářející ukazatel průběhu a dvě tlačítka:

```
R.Assign( 80, 160, 240, 180 );
pb := new( PProgressBar, init( R ));
pb^.Customize( ccProgressBar );
pb^.SetRange( 0, 100 );
pb^.SetPosition( hodnota );
pb^.AfterPaint := pbPaint;
pg^.Insert( pb );

R.Assign( 55, 160, 75, 180 );
bt := new( PButton, init( R, '-', 0 ,0 ));
bt^.Customize( ccButton );
bt^.BeforeHandle := zmensi;
pg^.insert( bt );

R.Assign( 245, 160, 265, 180 );
bt := new( PButton, init( R, '+', 0, 0 ));
bt^.Customize( ccButton );
bt^.BeforeHandle := zvetsi;
pg^.insert( bt );
```

Tento příklad je dodáván v adresáři ex04. Po spuštění vám umožní zvětšovat a zmenšovat hodnotu pomocí tlačítek. Aktuální hodnota je zobrazována ukazatelem průběhu, do které je doprostřed dokreslen číselný údaj.

Zatím jsme vždy pracovali s myší. Představme si, že chceme reagovat též na stisk kláves. Klávesnice vždy pracuje s jednou komponentou (s tou, která je v ohnisku). Standardně je nastaveno, že přeskakování mezi komponentami je prováděno pomocí klávesy Tab (příp. Shift-Tab) a provedení akce stisknutím tlačítka Enter. Bylo by možné upravit procedury zvětši a zmenši na reakci na tlačítko Enter, pokud je komponenta vybraná – tedy nějak takhle:

```
procedure zmensi( AControl: PControl; var AEvent: TEvent ); far;
begin
  if (AEvent.code = evMouseDown) or
     ((AEvent.code = evKeyDown) and (AEvent.KeyCode = kbEnter)) then
  begin
    if hodnota > 0 then dec( hodnota );
    pb^.SetPosition( hodnota );
  end;
end;
```

Přestože výše uvedené je možné, existuje „elegantnější“ řešení odezev na události. Tím řešením jsou oznámení (notifikace).

6. Oznámení (notification) – aplikační přizpůsobování chování komponent

Pokud je nějaká událost zachycena a dochází k jejímu zpracování, je možné, že bude potřeba změnit stav aplikace a tím pádem i komponenty, případně více komponent. K tomuto úkolu slouží oznámení.

6.1. Co je to oznámení (notification)

Kromě událostí existují ve vizualizačních knihovnách oznámení. Jedná se o zprávy vysílané jednotlivými komponentami. Oznámení mohou vznikat jako následky událostí a slouží k aplikačnímu zpracování. Zdrojem oznámení je vždy komponenta. Všechny komponenty vyvolávají oznámení při změnách stavu, jako je například získání ohniska, opuštění ohniska, povolení/zakázání komponenty, zobrazení/skrytí komponenty, žádost komponenty o nápovědu, žádost o zpracování události časovače, ukončení modálního stavu a další. Některé „stavové“ komponenty (jako například posuvník – **TScrollBar**, táhlo – **TTrackBar**) vyvolávají oznámení o změně svého vnitřního stavu v důsledku uživatelského zásahu. Prohlížečky seznamů (druh komponent, do nichž patří **TListBox**, **TListView**) generují oznámení o zvolení jedné položky ze seznamu, či oznámení požadavku na zobrazovaná data. Tlačítka vyvolávají oznámení o svém stisku (ať už myší, klávesou, či klávesovou zkratkou).
Oznámení jsou uloženy v záznamu (**TNotification = record**), který obsahuje zdrojovou komponentu oznámení (položka TNotification.Control), kód oznámení (položka TNotification.Code, obsahující konstantu s prefixem nm_) a další parametry specifické danému kódu.
Vizualizační prostředí bylo navrženo tak, aby programátor nepoužíval události, ale reagoval/obsluhoval oznámení.

6.2. Vyvolávání oznámení

Veškeré oznámení jsou vyvolávány metodou **TControl.NotifyEx**. (příp. TControl.Notify) Každá komponenta, která vyvolá oznámení je začne zpracovávat. Ke standardnímu zpracování je určena metoda HandleNotification. Tato metoda je v předkovi prázdná (TControl.HandleNotification) a potomci ji předefinovávají k chování svému.

6.3. Šíření oznámení, BeforeNotify, AfterNotify, ClearNotification

Stejně jako je možné pomocí procedurálních proměnných BeforeHandle a AfterHandle obsluhovat události, je vhodné pomocí obdobných procedurálních proměnných BeforeNotify a AfterNotify definovat chování aplikace v závislosti na změnách stavu komponent.
Každá komponenta, která dostane oznámení provede toto:

1) (pokud je definována,) zavolá procedurální proměnou BeforeNotify
2) zavolá svou metodu HandleNotification
3) (pokud je definována,) zavolá procedurální proměnnou AfterNotify
4) pokud ani jeden z výše uvedených bodů nepožádal o zastavení šíření oznámení, je toto posláno vlastníkovi komponenty.

Směr šíření oznámení je tedy od komponenty k vlastníkovi (tedy přesně opačný, než měly události).
Pokud některá komponenta oznámení obslouží a chce jej označit za obsloužené – tedy nechce povolit další zpracování tohoto oznámení, zavolá (globální) proceduru ClearNotification(oznámení); - tím požádá o ukončení zpracování oznámení a toto není dále zpracováváno.

Bylo by možné každé komponentě přidělit aplikačně závislou proceduru, která by vykonávala požadovanou činnost, ale to bychom měli velmi veliký počet (povětšinou krátkých) procedur. Druhý extrém by byl vytvořit pouze jednu jedinou notifikační proceduru, přidělit jí kořeni hierarchie vlastnictví – komponentě **TApplication**, a veškeré notifikace obsluhovat pouze v ní. Úkolem programátora je najít vhodný kompromis mezi těmito dvěma extrémy. Jako vhodné se jeví přiřadit notifikační procedury jednotlivým stránkám (**TPage**).
Aby jedna procedura mohla obsluhovat oznámení od více komponent, je třeba tyto komponenty nějak odlišit. K tomu je určen identifikátor komponenty.

6.4. Identifikátory komponent cidXXX

Každá komponenta má svůj identifikátor (typu Word). Ten je uložen v položce `TControl.Id`. Není ovšem potřeba, aby každé komponentě byl unikátní identifikátor přidělen. Jedná-li se o standardní komponentu s implicitním chováním, není třeba jí identifikátor přidělovat.
Doporučení:

1) identifikátory nadefinovat jako konstanty a pro jméno používat prefix `cid_`
2) správci stránek (**TPageControl**) přidělit identifikátor č. 1. (cidPageCtrl = 1)
3) každé stránce přidělit identifikátor, neboť podle něj správce stránek přepíná jednotlivé stránky, identifikátorům stránek přidělovat „kulatá“ čísla (např. 100, 200, 300 …, nebo $100, $200, $300 …)
4) identifikátory komponent ve skupině přidělovat vzestupně od čísla stránky + 1.

Přidělení identifikátoru komponentě se provádí následovně:

```
const
  cidMyText = 101;
...
var
  R : TRect;
  St : PStaticText;
begin
...
  R.Assign( 5,20,105,40 );
  St := new( PStaticText,init( R, 'Nějaký text'));
  St^.Id := cidMyText;
...
```

Využití těchto identifikátorů je právě v notifikačních procedurách. Typ funkce pro obsluhování oznámení TNotificationHandler je:

```
TNotificationHandler = procedure( var ANotification: TNotification );
```

6.5. Struktura notifikační procedury

Obecná struktura notifikační procedury je:

```
procedure MyNotify( var N: TNotification ); far;
begin
  case N.Control^.Id of
    cidXXX: begin
              case N.Code of
                nmAAAA: begin
                          ...
                        end;
```

```
                nmBBBB: begin
                          ...
                        end;
              end;
            end;
    cidYYY: begin
              case N.Code of
                nmCCCC: begin
                          ...
                        end;
                nmDDDD: begin
                          ...
                        end;
              end;
            end;
    cidZZZ: begin
              case N.Code of
                nmEEEE: begin
                          ...
                        end;
                nmFFFF: begin
                          ...
                        end;
              end;
            end;
    ...
  end;
end;
```

Jedná se tedy o rozeskoky dle identifikátorů komponent a ve zpracování jednotlivých komponent jsou rozeskoky dle jednotlivých kódu notifikace.

(Samozřejmě výše uvedené není dogma, je možné místo druhého vnořeného **case** použít **if**, zvláště pak v případech, kdy obsluhujeme od jedné komponenty pouze jeden kód oznámení.)

6.6. Příklad

Naším cílem nyní bude vytvořit stejné chování, jako v předešlém případě, ovšem nikoliv za použití událostí, ale s použitím notifikací.

Modifikujeme zdrojový příklad takto:

1) Přidáme definice konstant cid tlačítek

```
const
  cidZmensi = 101;
  cidZvetsi = 102;
```

2) Smažeme procedury zvetsi a zmensi
3) Vytvoříme proceduru MyNotify:

```
procedure MyNotify( var N: TNotification ); far;
begin
  case N.Control^.Id of
    cidZmensi: begin
                 if N.Code = nmClick then
                 begin
                   if hodnota > 0 then dec( hodnota );
                   pb^.SetPosition( hodnota );
                   ClearNotification( N );
```

```
                end;
              end;
    cidZvetsi: begin
                if N.Code = nmClick then
                begin
                  if hodnota < 100 then inc( hodnota );
                  pb^.SetPosition( hodnota );
                  ClearNotification( N );
                end;
              end;
  end;
end;
```

4) Stránce přidělíme tuto notifikační proceduru: za pg^.Customize(ccPage); přidáme:

```
pg^.AfterNotify := MyNotify;
```

5) Tlačítkům přidělíme po vytvoření identifikátory:

Tlačítku zmenši mezi vytvořením a vložením do stránky přidáme řádek

```
bt^.Id := cidZmensi;
```

Tlačítku zvětši mezi vytvořením a vložením do stránky přidáme řádek

```
bt^.Id := cidZvetsi;
```

6) Oběma tlačítkům povolíme volby ofSelectable a ofShowFocus: (= upravování vlastností, aby se projevilo, je třeba volat až **po** metodě Customize)

```
bt^.SetOptions( ofSelectable or ofShowFocus, 0 );
```

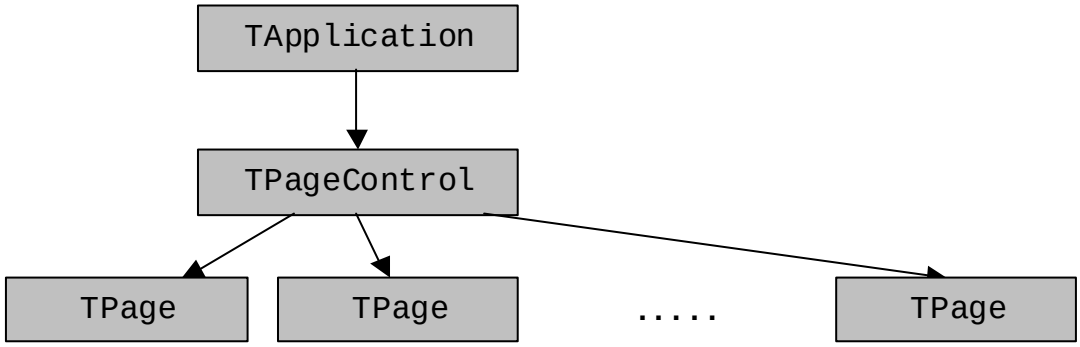
7) příklad přeložíme a spustíme.

(Tento příklad je dodáván jako EX05.)

7. Správce stránek – TPageControl

Máme-li více stránek než pouze jednu, je zapotřebí mezi těmito stránkami nějak přeskakovat. K tomuto přeskakování slouží správce stránek. Jedná se o skupinovou komponentu, která je zařazena mezi aplikaci (**TApplication**) a jednotlivé stránky (**TPage**).

Správce stránek smí obsahovat pouze komponenty typu stránka (**TPage**). Nejvyšší tři vrstvy hierarchie vlastnictví se správcem stránek tedy vypadají zhruba takto:



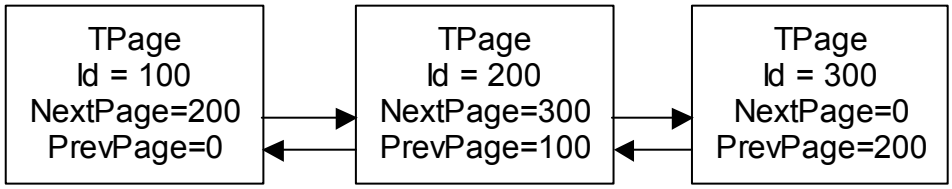
Protože stránky jsou vždy stejně veliké (tak aby zabíraly celý zobrazovací prostor displeje), je zobrazovaná velikost správce stránek stejná, jako velikost jednotlivých stránek a pro vytvoření stránek se využívá volání metody:

```
TPageControl.NewPage( Id: word );
```

Tato metoda zajistí vytvoření stránky příslušné velikosti a přiřadí stránce identifikátor Id.

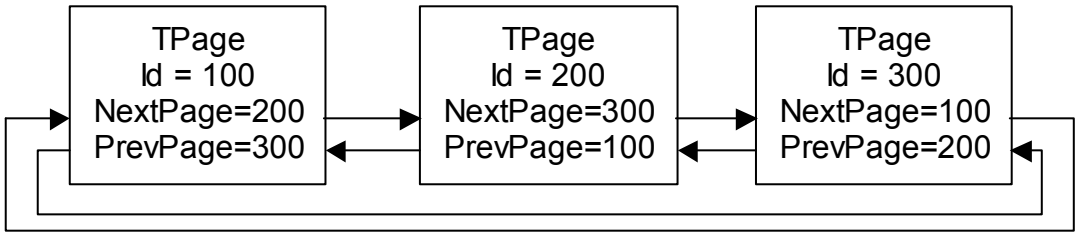
Pro přepínání mezi stránkami slouží metoda TPageControl.GotoPage(Id: word); , která zobrazí stránku určenou identifikátorem id.

Každá stránka obsahuje identifikátor svého předchůdce (TPage.PrevPage) a identifikátor svého následníka (TPage.NextPage). Pokud tyto hodnoty nastavíme, vytvoříme seznam stránek mezi sebou provázaných. Například chceme-li 3 stránky zařadit do lineárního seznamu, nastavíme identifikátory předchůdců a následníků takto:



Hodnotu 0 je vhodné nechat nedefinovanou, čímž bude znamenat něco jako odkaz **NIL**.

Chceme-li tytéž tři stránky zařadit do cyklického seznamu, nastavíme identifikátory předchůdců a následníků takto:



Pro skoky na následující resp. předchozí stránku slouží metoda TPageControl.GotoNextPage; resp. TPageControl.GotoPrevPage; .

Kromě přepínání stránek správce též nabízí volání stránek a návraty ze stránek. Máme-li zájem zavolat stránku, vyvoláme metodu TPageControl.CallPage(id); . Pro zpětný návrat slouží metoda TPageControl.RegurnPage.

(Vnitřní implementace není zásobník, nýbrž pouze každá stránka má zpětný odkaz na stránku, z níž byla vyvolána. To je jisté omezení, které je třeba uvažovat, při návrhu aplikační hierarchie obrazovek – stránek.)

8. Strom (význačných) komponent

Přestože všechny komponenty jsou začleněny do hierarchie vlastnictví, odkazy na některé komponenty je vhodné si uchovávat v proměnných. K tomuto účelu slouží (v aplikaci) deklarovaný záznam takovéhoto typu:

```
{-------------------------------------------------------}
{ Odkazy na instance komponent ve stromu aplikace       }
{-------------------------------------------------------}

var
  Tree :
  record
    PageCtrl : PPageControl;

    MainPage : record         { Hlavni stranka         }
      Title   : PStaticText;  { Titulek stranky        }
      Navig   : PNavigator;   { Navigator stranky      }
      ...
    end;

    OtherPage : record        { Dalsi stranka          }
      Title   : PStaticText;  { Titulek stranky        }
      Navig   : PNavigator;   { Navigator stranky      }
      ...
    end;

    ...

  end;
```

Vytváříme-li komponenty, používáme proměnné z této struktury a můžeme je kdekoliv v programu využít.
Ukazatel na hlavní Aplikaci (proměnnou typu PApplication) je uložen v globální proměnné Controls.Application.

Doporučení: Po vytvoření aplikace (**TApplication**) nejdříve vytvoříme správce stránek a pak jednotlivé stránky vytváříme voláním metody `Tree.PageCtrl^.NewPage( id );`

```
...
  ap := new( PApplication, Init (…));
  ap^.Customize( ccApplication );

  R.Assign( ... );
  Tree.PageCtrl := new( PPageControl, Init( R ));
  Tree.PageCtrl^.Customize( ccPageControl );

  Page := Tree.PageCtrl^.NewPage( cidMainPage );
  Page^.Customize( ccPage );
  Page^.NextPage := cidOtherPage;
  ...
  Tree.PageCtrl^.Insert( Page );
```

```
  Page := Tree.PageCtrl^.NewPage( cidOtherPage );
  Page^.Customize( ccPage );
  Page^.PrevPage := cidMainPage;
  ...
  Tree.PageCtrl^.Insert( Page );
...
```

9. Skelet aplikace

Vizualizační knihovny jsou dodávány s předpřipravenými velmi jednoduchými aplikacemi, které mají sloužit jako základ skutečných použitelných aplikací. Tyto jednoduché aplikace jsou nazývány skelety. Skelety se nachází v adresáři VizLIB\SKELETY\ V současné době je k dispozici skelet pro Term10, Touch11, Touch33, počítač PC a pro terminál Term90.

Aplikace (kořen stromu) je vytvořena v souboru uMenuIni. V tomto kořeni jsou též odkazy na vstupní a výstupní zařízení.
Většinou jako jediná (skupinová) komponenta je do aplikace vložen správce stránek (**TPageControl**), který je určen k přepínání stránek mezi sebou. Správce stránek a jeho podstrom je vytvořen procedurou CreateAppPages v souboru uMenuDef.pas.

Cílem této kapitoly je uvedení do provozu již předpřipraveného skeletu aplikace.
Předpoklady:

1) Počítač s os. DOS či Windows
2) Polohovací zařízení včetně instalovaného ovladače do příslušněh o.s.
3) Nainstalován programovací jazyk Borland Pascal 7.
4) Standardní knihovny fy. SofCon s.r.o.
5) Vizualizační knihovny fy. SofCon s.r.o. (tento manuál je jejich součástí).

Postup:

1) Zvolíme si skelet, pro který terminál má být aplikace napsána [T11, T10, T33].
2) Zkopírujeme adresář (VIZ\SKELETY\) (vybraný_skelet) na vhodné místo.
3) Přejmenujeme lde nového názvu aplikace
4) V adresáři VerPC spustíme Borland Pascal
5) Nastavíme prostředí - cesty ke knihovnám a primární soubor (app.pas)
6) Přeložíme program
7) Spustíme program

Pokud je vše v pořádku, můžeme vidět na obrazovce simulátor obrazovky/displeje vybraného terminálu. Aplikaci můžeme ukončit stisknutím Alt-X .

9.1. Podmínky překladu (soubor SETS.INC)

Tyto skelety (stejně jako většina programů pocházejících z fy. SofCon s.r.o.) obsahují podmíněné překlady. Definování direktiv podmíněných překladů je v souboru SETS.INC a prakticky všechny aplikačně závislé zdrojové soubory obsahují na začátku `{$I SETS.INC}`.
Pro první spuštění je vhodné tento soubor nikterak neupravovat.

9.1.1. DOS/MCP

Direktiva DOS určuje, že program je spouštěn na počítači typu PC.
Direktiva MCP určuje, že program je překládán a nahráván do řídícího systému fy. SofCon.
V adresáři VerPC by měla být definována direktiva DOS a v adresáři VerMC by měla být definována direktiva MCP.

9.1.2. COLOR

Direktiva COLOR určuje, zdali je použit barevný terminál. V případě T10 nebo T33 necháme nedefinováno.

9.1.3. Ladění aplikací na PC – direktivy UseVESA a TermEMU

Pro emulaci displejů terminálů lze využít tři různé emulační prostředky:

VGA	V knihovně PCDrv je definován ovladač standardní VGA karty v monochromatickém režimu (TMonoVGADriver). Tento ovladač umožňuje emulaci terminálů s monochromatickými displeji až do rozlišení 640x480 bodů. Ovladač pracuje bez omezení na všech platformách.
VESA	V knihovně PCDrv je definován ovladač pro standardní SVGA karty s rozšířením VBE (což jsou dnes prakticky všechny grafické karty). Tento ovladač umožňuje emulaci terminálů s monochromatickými i barevnými displeji. Rozlišení lze nastavit od 320x240 až do 1024x768 bodů. Ovladač pracuje bez omezení v DOSu a ve Windows 95/98/ME. Ve Windows XP je nutné ovladač vyzkoušet.
Emulátor terminálu	V knihovně TEDrv je definován ovladač emulátoru terminálu (TTEDisplayDrv). Tento ovladač využívá pro zobrazování aplikaci emulátoru terminálu (TermEmu) běžící na pozadí v okně. Ovladač umožňuje emulaci terminálů s monochromatickými i barevnými displeji. Rozlišení lze nastavit libovolné. Ovladač lze použít pouze ve Windows 2000 a Windows XP.

	DOS	Windows 95/98/ME	Windows 2000	Windows XP	Protect Mode
VGA	●	●	●	●	●
VESA	●	●		●[1)]	
Emulátor terminálu			●	●	●

1) Nutno vyzkoušet

9.1.3.1. Režim VGA

Vlastnosti:

a. monochromatické zobrazování
b. pouze celoobrazovkový režim

c. funkční na většině platforem

Direktivy překladu:

V souboru sets.inc necháme nedefinováno USEVESA a necháme nedefinováno TERMEMU.

```
...
{ define USEVESA}
{ define TERMEMU}
...
```

9.1.3.2. Režim VESA

Vlastnosti:

a. možnost i barevného zobrazování
b. pouze celoobrazovkový režim
c. nefunkční na os. Microsoft Windows 2000

Direktivy překladu:

V souboru sets.inc definujeme USEVESA a necháme nedefinováno TERMEMU.

```
...
{$define USEVESA}
{ define TERMEMU}
...
```

9.1.3.3. Emulátor terminálu

Vlastnosti:

a. možnost i barevného zobrazování
b. zobrazování je prováděno do samostatného okna systému Windows
c. funkční po nainstalování ovladačů na Windows 2000, Windows XP

Direktivy překladu:

V souboru sets.inc definujeme TERMEMU (na stavu direktivy USEVESA v tomto přízadě nezáleží).

```
...
{ define USEVESA}
{$define TERMEMU}
...
```

9.2. Soubor app.pas

V tomto souboru je inicializován systém reálného času ReTOS a jediná činnost zde uvedená je spuštění vizualizace a čekání na ukončení. Spuštění vizualizace je provedeno procedurou StartMenuProcess(…) .

Tento soubor je předurčen k tomu být hlavním souborem aplikace. Je možné/vhodné jej přejmenovat dle názvu aplikace.

Využívá knihovní jednotky: Kernel, Tick, uMenuIni.

Jednotka Kernel je uvedena z důvodu použití operačního systému reálného času ReTOS.

Jednotka Tick je použita z důvodu zrychlení systémového časovače na 10 ms. Máme-li dotykový panel (tedy Touch11 nebo Touch 33), je třeba periodicky vyvolávat

tikající proceduru tohoto panelu. Perioda tohoto tikání je maximálně 10 ms. (na terminálu Term10 není potřeba).

9.3. Soubor UMenuIni.pas

Tento soubor je jednotka (unit) obsahující v interface sekci pouze jednu proceduru, a to StartMenuProcess(…).

Tato procedura nejprve inicializuje vizualizační prostředí (voláním InitMenu), poté spustí vizualizační proces(start(…);exit;) a v něm spustí vlastní vizualizaci (voláním RunMenu) a po ukončení uvolní proměnné z paměti (voláním DoneMenu).

Vzhledem k tomu, že se jedná o událostmi řízené programování, provádí RunMenu pouze spuštění klasické smyčky – „Vezmi událost" a „Zpracuj událost".

Tedy hlavní těžiště práce spočívá v nastavení vizualizačního prostředí tak, aby se chovalo dle předpokladů. Toto nastavení je prováděno v proceduře InitMenu.

Procedura InitMenu provádí toto:

1) inicializuje příslušná vstupní a výstupní zařízení
2) vytvoří hlavní komponentu (typu **TApplication**)
3) zavolá proceduru CreateAppPages, čímž nakonfiguruje
    a. CO bude zobrazováno a zároveň
    b. JAK se to bude chovat.

Procedura CreateAppPages je definována v souboru UMenuDef.pas

V implementation sekci jsou definovány i další užitečné konstanty a procedury.
Konstanta `cMenuStatPri` udává statickou prioritu procesu menu v OS ReTOS.
Konstatnta `cMenuDynPri` udává dynamickou prioritu procesu.
Konstanta `cMenuWaitCount` udává počet waitů jádra při nečinnosti uživatele.
Konstanta `cMenuStackSize` udává velikost zásobníku vyhrazeného vizualizačnímu procesu.
Procedura `AppIdle` je volána z vizualizačních knihoven při nečinnosti uživatele. Standardně je nastaveno čekání cMenuWaitCount tiků jádra.

9.4. Soubor uMenuDef.pas

9.4.1. Procedura CreateAppPages – vytvoření správce stránek

(*spusťte si vývojové prostředí Borland pascal a podívejte se na tuto proceduru*)
Tato procedura vytvoří správce stránek (typu **TPageControl**), zavolá procedury pro vytvoření jednotlivých stránek – CreateMainPage, CreateOtherPage a CreateTxxSetupPages. Dále nastaví některé své vlastnosti a nakonec vloží správce stránek do aplikace.
Procedury CreateXXXPage vytvářejí jednotlivé stránky aplikace.

Každá stránka zabírá kompletně celou obrazovku terminálu.

9.4.2. Procedura CreateMainPage – vytvoření hlavní stránky

(*podívejte se na tuto proceduru*)
Vzhledem k tomu, že stránky jsou stejně veliké – a to přesně stejně jako správce stránek, je vytváření instance stránky (TPage) provedeno voláním metody správce stránek `TPageControl.NewPage( id );` , který vytvoří stránku přesně tak velikou, jak má být.

9.4.3. Příklad

Cíl: přidání textu „Pokusny text“ na hlavní stránku na nějakou pozici.

Ve vývojovém prostředí Borland pascalu otevřeme soubor uMenuDef.pas, nalezneme proceduru CreateMainPage a tu budeme editovat.

Postup:

1) Do deklarace lokálních proměnných (procedury CreateMainPage) přidáme
   ```
   St : PStaticText;
   ```
2) Zatím v těle této vytvářecí procedury jsou pouze titulek (title) a navigátor (navig). Za ně přidáme tyto řádky:
   ```
   R.Assign( 30, 30, 130, 50 );
   St := new( PStaticText, Init( R, 'Pokusny text'));
   St^.Customize( ccStaticText );
   Page^.Insert( St );
   ```
3) Spustíme program.

První řádka určuje pozici, na níž má být text vykreslen.
Druhá řádka vytvoři instanci proměnné obsahující příslušný text s implicitními hodnotami.
Třetí řádka přizpůsobí vlastnosti komponenty. (bude vysvětleno dále)
Čtvrtá řádka vloží tento text do stránky.

Veškeré další příklady jsou založeny na těchto skeletech, tudíž je měněna jen jednotka uMenuDef.pas.

10. Vlastnosti komponent a jejich nastavování

10.1. Stavy komponenty

Každá komponenta má proměnnou state. Tato proměnná udává, v jakém stavu se komponenta nachází. Možné příznaky jsou:

1) Viditelnost komponenty: `sfVisible`
2) Vybranost komponenty: `sfSelected`

3) Ohniskovost komponenty: `sfFocused`
4) Zakázání komponenty: `sfDisabled`
5) Modální stav: `sfModal`
6) Viditelnost kurzoru: `sfCaretVis`
7) Příznak aktivního okna: `sfActive`
8) (Vnitřní) příznak zobrazovanosti: `sfExposed`
9) (Vnitřní) příznak překrytí: `sfOverlapped`

Ke změnám jednotlivých příznaků stavové proměnné je určena metoda `TControl.SetState( AState: word; AEnable: boolean)`, kde `AState` je konstanta `sfXXX` a `AEnable` určuje, zdali se má příznak nastavit, nebo odstranit. Většina stavů je měněna automaticky a pokud potřebujeme stav nějak ručně změnit, je vhodnější použít speciálně k tomu určené funkce.

10.1.1. Příznak sfVisible – Zobrazení a skrytí komponenty

Za běhu aplikace lze libovolnou komponentu skrýt a opětovně zobrazit. K tomu slouží dvě metody:

```
procedure TControl.Show;
procedure TControl.Hide;
```

Metoda **Hide** skryje komponentu. Pokud komponenta je vlastník jiných komponent, pak jsou skryty zároveň i všechny komponenty, které jsou umístěny v této komponentě. Metoda **Show** zobrazí komponentu, která byla dříve skryta pomocí metody **Hide**.

Metody **Show** a **Hide** zároveň generují oznámení nmShow a nmHide.

Tyto dvě metody jsou demonstrovány v příkladě ex10\1\ (Main page) V horní třetině na hlavní obrazovce jsou dvě tlačítka, a jeden statický text. Jedno tlačítko statický text skryje (Hide), druhé je ukáže (Show).

10.1.2. Příznak sfDisable – Zakázání a povolení komponenty

Za běhu aplikace lze libovolnou komponentu zakázat a opětovně povolit. Tzv. zakázaná komponenta nemůže přijímat žádné vstupní události, takže se vůči uživateli chová, jako kdyby byla nefunkční. Komponenta může zakázaný stav odlišit od povoleného např. jiným zobrazením komponenty. Pro zákaz a povolení komponenty se používají dvě metody:

```
procedure TControl.Enable;
procedure TControl.Disable;
```

Metoda **Enable** komponentu povolí a metoda **Disable** komponentu zakáže.

Metody **Enable** a **Disable** zároveň generují oznámení nmEnable a nmDisable.

Tyto dvě metody jsou demonstrovány v příkladě ex10\1\ (Main page). Na hlavní obrazovce zhruba uprostřed jsou dvě tlačítka – Enable a Disable a vpravo vedle nich je další tlačítko čítající počet svých stisků. Cílem je ukázat, že zakázané tlačítko nečítá stisky, neboť nedostává události. Pro zvýraznění, zdali je tlačítko povoleno či zakázáno, je při zákazu tlačítko škrtnuto.

10.1.3. Příznaky sfSelected a sfFocused – změna ohniska

U komponent lze rozlišit tři možné stavy:

- Komponenta není vybraná
- Komponenta je vybraná
- Komponenta je vybraná a nachází se v ohnisku

Tyto tři stavy jsou odlišeny příznaky sfSelected a sfFocused v položce State třídy **TControl**.

K přepnutí ohniska nebo výběru komponenty lze použít následující metody.

```
function  TControl.Focus: Boolean;
procedure TControl.Select;
function  TGroup.FocusNext( AForwards: Boolean ): Boolean;
procedure TGroup.SelectNext( AForwards: Boolean);
```

Metoda **Select** provede výběr komponenty v rámci skupiny. Pokud je vlastník komponenty v ohnisku, dostane se do ohniska i tato komponenta.

Metoda **Focus** provede výběr komponenty v rámci skupiny a zároveň rekurzivně zavolá metodu **Focus** vlastníka komponenty. V podstatě dojde k výběru všech komponent na cestě ke kořeni stromu komponent. V případě, že se podařilo všechny komponenty na cestě ke kořeni stromu vybrat, vrátí metoda hodnotu True. V opačném případě, kdy např. jedna z komponent nepovolí přepnutí, metoda vrací hodnotu False.

Metoda **SelectNext** provádí výběr následující nebo předchozí komponenty v seznamu vlastníka a metoda **FocusNext** přepíná ohnisko na následující nebo předchozí komponentu vlastníka.

Tyto metody je možné ručně volat z aplikace, jinak **jsou obvykle automaticky volány** např. při stisku klávesy pro přesun na následující příp. předchozí komponentu, při přepnutí stránky nebo okna apod.

Komponenty při získání ohniska generují notifikaci **nmEnter** a při ztrátě ohniska notifikaci **nmExit**.

10.1.4. Příznak modálního stavu sfModal

V aplikaci je vždy právě jedna komponenta v modálním stavu. Po spuštění je to přímo **TApplication**.

Pokud je komponenta v modálním stavu, přivlastňuje si veškeré poziční a ohniskové události. Pro uvedení komponenty do modálního stavu slouží metoda TControl.Execute. Samostatné volání této metody se nedoporučuje. Pro přepnutí komponenty do modálního stavu se využívá metoda:

```
TGroup.ExecControl( komponenta: PControl );
```

Ukončení modálního stavu se provádí nastavením TControl.ModalResult na hodnotu jinou, než mrNone.

10.1.5. Zobrazování kurzoru – sfCaretVis

Pro příznak `sfCaretVis` jsou to TControl.ShowCaret a TControl.HideCaret.

10.1.6. sfActive, sfExposed, sfOverlapped

sfActive je příznak právě aktivního okna. Bližší specifikiace – viz referenční příručka. Příznaky sfExposed a sfOverlapped jsou pouze vnitřní implementační příznaky.

10.2. Nastavení (option) komponenty

Každá komponenta má proměnnou option. Tato udává některé nastavení komponent. Příznaky jsou:

1) Vybratelnost: `ofSelectable`
2) Přesun do popředí při výběru: `ofTopSelect`
3) Zpracování události před komponentou v ohnisku: `ofPreProcess`
4) Zpracování události po komponentě v ohnisku: `ofPostProcess`
5) Reakce na první klik: `ofFirstClick`
6) Zakázání přepnutí ohniska: o`fValidate`
7) Sdílení palety `ofSharedPalette`
8) Vykreslování pozadí: `ofBackground`
9) Vykreslování komponenty metodou paint: `ofPaintControl`
10) (Vnitřní) příznak: `ofFlickSafe`
11) Zobrazování ohniska komponentou: `ofShowFocus`

K nastavování jednotlivých nastavení (option) slouží metoda:

```
TControl.SetOptions( set, reset );
```

přičemž do položky set uvedeme jednotlivé příznaky, které chceme nastavit a do položky reset jednotlivé příznaky, které chceme zrušit.
Množiny set a reset udáváme jako jednotlivé příznaky oddělené buďto operátorem aritmetického součtu (+), nebo operátorem logického součtu (or), neboť se jedná o vzájemně nekolidující bity na wordu.

10.2.1. Schopnost komponenty býti vybrána – příznak ofSelectable

Tento příznak určuje, zdali při předávání ohniska je možné tuto komponentu vybrat/zaostřit.

10.2.2. Přesun do popředí při výběru – příznak ofTopSelect

Máme-li několik překrývajících-se komponent, přičemž tyto komponenty jsou zvolitelné `((TControl.Options and ofSelectable) <> 0)`, tak je možné přiřadit jednotlivým komponentám příznak ofTopSelect - „přenést do popředí při výběru“.

Příklad demonstrující tento příznak je EX10\1\ (Other page). Zde jsou přes sebe čtyři statické texty (s rámečky), přičemž je explicitně (ručním dokreslováním) zobrazován fokus tečkovanou čarou a těmto čtyřem textům je nastaven i příznak ofSelectable i příznak ofTopSelect. Z tohoto příkladu je dobře vidět, že označení komponenty s příznakem ofTopSelect přenese komponentu do popředí.

10.2.3. Zpracování ohniskových událostí nezaostřenou komponentou – příznaky ofPreProcess a ofPostProcess

Pokud máme zájem zpracovávat ohniskové události (zatím jen události od klávesnice) též komponentou, která není v ohnisku, nastavíme jeden z těchto příznaků. Použití je většinou u navigátoru, který obsluhuje reakce na „hot keys“. Více o zpracování ohniskových událostí nezaostřenými komponentami je v referenčním manuálu.

10.2.4. ofFirstClick

Kliknutí na nevybranou vybratelnou komponentu způsobí její vybrání. Pokud si zároveň přejeme, aby tento první klik též provedl akci, nastavíme příznak ofFirstClick.

10.2.5. ofValidate

Standardně uživatel může přepínat fokus komponent tak jak se mu zlíbí. Pokud máme komponentu, která si má fokus (v některých případech) udržet, nastavíme této komponentě příznak ofValidate a v notifikační proceduře obsloužíme nmCanExit (pokud chceme zakázat opuštění, nastavíme N.Accept = False;)

10.2.6. ofSharedPalette

Vnitřní příznak, používaný při nastavování barev. Pokud nastavujete barvy pomocí TControl.SetColor, je tento příznak zpracováván automaticky. Pokud používáte TControl.SetPalette, musíte si dát na tento příznak pozor.

10.2.7. ofBackGround

Pokud komponenta má nastaven tento příznak, tak před vykreslením (před zavoláním BeforePaint, Paint, AfterPaint) bude přemazána barvou svého pozadí.

10.2.8. ofPaintControl

Pokud komponentá má tento příznak nastaven, pak je při standardním šíření událostí vyvolávána její metoda Paint. Doporučení – neměnit.

10.2.9. ofFlickSafe

Vnitřní implementační příznak rychlého překreslení komponenty bez probliknutí.

10.2.10. ofShowFocus

Na terminálech bez polohovacího zařízení, kde máme k dispozici pouze klávesnici je potřeba rozlišovat s kterou komponentou právě pracujeme. Je to právě ta komponenta, která je v ohnisku. Aby se tato komponenta lišila od stejných komponent, které v ohnisku nejsou, umožňují některé komponenty tuto ohniskovost zobrazovat.

Příklad demonstrující tento příznak je EX10\1\ (Other page). V pravé části obrazovky jsou dvě tlačítka. Vrchní má příznak ofShowFocus nastaven, spodní jej má nenastaven. Zkuste si je proklikat.

10.3. Maska událostí

Každá komponenta obsahuje položku EventMask, která povoluje či zakazuje reagovat na jednotlivé typy událostí. Implicitně po vytvoření všechny komponenty mohou reagovat na všechny události (tedy položka EventMask obsahuje $FFFF). Chceme-li například zakázat statickému textu (komponentě) reagovat na dvojklik myši, provedeme:

```
st^.EventMask := st^.EventMask and not evMouseDbl;
```

10.4. Vlastnosti specifické pro jednotlivé komponenty

Většina komponent má své vlastní vlastnosti. Tyto jsou nastavovány metodami specifickými pro uvedený druh komponent. Například statický text pro změnu svého textu má metodu TStaticText.SetText(text);
Většina komponent má položku flags. Tato obsahuje typově závislé příznaky. Tyto jsou nastavovány pomocí metody SetFlags(set, reset);
Jednotlivé příznaky mají třípísmenný prefix XYfAAAA, kde XY jsou identifikátory typu komponenty (TStaticText má st, TButton má bt, atp) a AAAA je anglické jméno

vlastnosti. Většina komponent má příznak XYfBorder, který znamená, že komponenta má kolem sebe vykreslit rámeček. Další příznaky jsou v referenčním manuálu.

10.5. Přizpůsobení vybranému terminálu

Komponenta vytvořená voláním konstruktoru má nastaveny implicitní parametry. Vzhledem k různosti ovládání terminálů je vhodné volat procedury nastavující parametry pro daný terminál.
Term10 – má klávesnici, nemá myš, je černobílý
TouchXX – nemá klávesnici, má TouchPanel, Touch11 může být barevný
PC – má klávesnici a většinou i myš, může být černobílý i barevný
…

Každá komponenta má svou základní třídu nastavení. Identifikátor této třídy nastavení je složen z prefixu cc_ (**c**omponent **c**lass) a jména typu komponenty bez úvodního T (takže např. ccPage, ccButton, ccApplication, ccStaticText…).

Každá komponenta má metodu Customize. Tato metoda je určena právě k nastavení parametrů příšlušné komponenty dle identifikátoru třídy nastavení. Tato metoda Customize postupně volá konfigurační procedury. Standardně každý terminál má svojí konfigurační proceduru. Tyto procedury jsou v souborech XXXcus.pas a pro jejich použití je třeba je uvést v klauzuli uses.
Jednotné volání této metody má tvar:

```
TControl.Customize( ccXXXX );
```

11. Knihovní komponenty

Detailní popis komponent je v referenčním manuálu. Zde je jen všeobecný přehled.

11.1. TStaticText

Velmi jednoduchou komponentou je statický text. Je určen k zobrazování textu.
Bývá zvykem v aplikacích, že na horním okraji stránky je (statickým textem) vykreslen titulek.
Tato komponenta umožňuje do obdelníkové oblasti určeným fontem vypsat jeden či více řádků textu. Tento text je možno zarovnat a odsadit (v horizontálním i vertikálním směru).
Kromě základní třídy nastavení komponenty ccStaticText existuje ještě třída ccStaticTextTitle, která je určená k vykreslování titulku stránky.

Příklad je v EX10\2\. (main page) Jedná se o jednoduchý statický text, který zobrazuje hodnotu, která je každým kliknutím zvětšena.

11.2. TNavigator

Velmi často používanou komponentou je navigátor. Jedná se o horizontální obdelník s jednotlivými panely, které zobrazují příkazy/funkce, možné v právě aktuálním kontextu (aktuální obrazovce) provést. Typicky na terminálech s klávesnicemi zobrazují „hot keys“. Na terminálech s polohovacím zařízením na stisknutí reagují provedením příslušné činnosti. Bývá zvykem tento navigátor umisťovat na spodní okraj obrazovek. Chování této komponenty je uzpůsobeno tomu, aby zpracovávala „hot keys“ (má nastaven příznak ofPostProcess).

11.3. TButton

Klasickým prvkem je tlačítko. Zde umožňuje provést předdefinovanou akci (stisk klávesy klávesnice, ukončení modálního stavu, skok na určitou stránku, zavolání určité stránky, návrat ze stránky, skok na následující stránku, skok na předcházející stránku), nebo programátorem definovanou akci. (Programátor tuto akci definuje do notifikační procedury.)
Je možné nastavit, zde se jedná o tlačítko, či o přepínač.

Příklad A ukazující uživatelsky definovanou akci je dodáván v Ex10\1\ (Main page). Tlačítko uprostřed obrazovky čítá počet svých stisknutí obsloužením své notifikace nmClick.
Příklad B simulující stisk klávesy Alt-X je dodáván v Ex10\2\. (Main page). Tlačítko vlevo dole nad navigátorem vnutí aplikaci stisknutí klávesy Alt-X, čímž na PC ukončí program.
Příklad ukazující ukončení modálního stavu bude uveden u příkladu s dialogovým oknem.

11.4. TEdit

Komponentou pro zadávání znakových údajů je TEdit. Jedná se o komponentu, která zobrazuje jednořádkový text, který je možno pomocí klávesnice editovat.

V adresáři EX10\2 (Other page) je stránka, ve které jsou pod sebou tři editační komponenty. První z nich je standardní TEdit se zarovnáním doleva. Druhý má zarovnání doprava a zároveň pokud text je delší než aby se vešel do zobrazitelné části komponenty, objeví se na stranách značky toto signalizující. Třetí TEdit ukazuje, jak je možné použít zástupný znak při editaci – zde hvězdička – což je výhodné například pro zadávání hesel.

11.5. TImage

Pro zobrazení bitmapy (obrázku) slouží komponenta TImage. Bitmapy se do vizualizačních obrazovek dostanou pomocí BmpCreatoru – aplikace pod windows, která umožňuje z několika bitmap vytvořit programovou jednotku (unit), která jde použít k vykreslování.

11.6. TScrollBar

Další klasickou komponentou je posuvník (TScrollBar). Většinou slouží k rolování výřezu z nějaké velké struktury, která se nevejde na celou obrazovku. Může být s tlačitky na okraji (pro polohovací zařízení) nebo bez nich (pouze pro ukázání aktuálního výřezu).
Je možné určit, zda bude velikost posuvníku konstantní, či proporcionální vzhledem k počtu zobrazených položek.

Příklad na použití posuvníku bude u komponenty TListBox.

11.7. TTrackBar

Ne moc častým grafickým prvkem je táhlo (TTrackBar). Více než dlouhý popis řekne obrázek:

Táhlo slouží k nastavování ordinálních hodnot ve vymezeném rozsahu.

11.8. TProgressBar

Další ze standardních grafických komponent je ukazatel průběhu (TProgressBar).
Jedná se o obdelník, který se z jednoho konce postupně zaplňuje ke druhému konci.

Jeho použití již bylo ukázáno v jednoduchém příkladu EX04. Vzhledem k velmi častému požadavku zobrazení též hodnoty uprostřed tohoto ukazatele průběhu, je tamtéž uvedena procedura číselnou hodnotu vykreslující.

11.9. TUpDown

Tato komponenta je „dvoutlačítko“ – tedy jedna komponenta generující dvoustavový výstup (typu [přidat, ubrat]).

11.10. TListBox

Komponentou pro výpis (nestrukturovaného) seznamu položek je ListBox. Umožňuje vypisovat na obrazovku do jednoho či více sloupců položky seznamu (nejčastěji pole řetězců). Dále umožňuje zvolení (vybrání) jedné položky. K této komponentě je možné přidružit posuvník (TScrollBar).

Jako příklad se podívejte na EX10\2\ (Third page). Můžete vybírat položky z prohlížečky seznamu, přičemž právě aktivní prvek je vypisován v prvním statickém

textu pod prohlížečkou a vybraný prvek je vypisován v druhém statickém textu pod prohlížečkou (vybrání se provede pomocí double-clicku, nebo klávesy Enter).

Zde se nově objevuje nový typ notifikace – kód **nmGetData**. Tento kód generuje komponenta, pokud žádá informaci pro překreslení. Pozor! Notifikační procedura obsluhující nmGetData uloží do oznámení adresu řetězce, který komponenta požadovala. (`N.Text := @S;`) Protože se jedná o adresu (proměnné **S**), je třeba zajistit, aby tato (adresa) byla platná i po opuštění zpracovávací notifikační procedury – tedy (**S**) nemůže být lokální proměnná (na zásobníku), musí se jednat buďto o globální proměnnou, nebo o lokální inicializovanou proměnnou (tedy v datovém segmentu). (Vysvětlení: pokud by se jednalo o lokální proměnnou na zásobníku, byla by tato zneplatněna odchodem z notifikační procedury a kterýkoliv jiný proces či přerušení by mohlo tento řetězec přepsat.)

11.11. TListView

Kopmonentou pro výpis strukturovaného seznamu položek je ListView. Je určen k vypisování jednotlivých položek záznamů do více sloupců, přičemž jeden záznam zabírá právě jeden řádek. Tuto komponentu je možné vytvořit se záhlavím, či bez něho, se svislou, vodorovnou či oběma mřížkami, jednotlivé údaje ve sloupcích je možné horizontálně zarovnávat.

Příklad je v adresáři EX10\3\ (Main page). Jedná se o zobrazení pole položek, přičemž do této stránky je přidělána editace jednotlivých položek pole. Editace je provedena vytvořením komponenty TEdit, vložením do stránky na příslušné místo tak, aby přesně překrývala vybranou položku, spuštěním modálního stavu tohoto editačního řádku a při kladné odpovědi (mrOK) je hodnota přepsána do datového pole. Tato editace je zbytečně složitá, pro zjednodušení existují funkce InLineEdit, příp. InLineEditEx, které budou probrány později.

11.12. TListEdit

TListEdit je upravená komponenta ListView. Úprava spočívá ve specializaci na editaci fyzikálních/technologických parametrů - na 2-3 sloupce, a to název parametru, skutečnou hodnotu (a jednotku). Dále tato komponenta umožňuje skutečnou hodnotu měnit/editovat. (ve spojení s validátory).

Příklad použití této komponenty je u validátorů.

11.13. TPaintBox

TPaintBox je prázdná komponenta jejíž veškeré chování a vykreslování je ponecháno na programátorovi.

11.14. TMenuBar

MenuBar je horizontální „řádkové“ menu. Programátor definuje jednotlivé položky vykonávající akce, nebo položky pod-menu s ucelenými skupinami příkazů.
Jednotlivé položky mohou samy vykonávat akce: skok na stránku, zavolání stránky, návrat ze stránky, ukončení modálního stavu, nebo vyvolat notifikaci, kterou si programátor obslouží jak bude právě potřebovat.

Příklad je dodáván v adresáři EX10\3\ (Other page). K vyvolání MenuBar slouží buďto kliknutí do oblasti menu, nebo stisknutí klávesy vkMenu (na PC je to F10). Jednotlivé příkazy pouze zkopírují text vybrané položky do StaticTextu.

11.15. TMenuBox

MenuBox je vertikální „sloupcové“ menu. Programátor nevytváří instance tohoto typu, neboť ty jsou samy dynamicky vytvářeny při procházení ostatními druhy menu (TMenuBar, TMenuPopUp).

11.16. TMenuPopUp

MenuPopUp je „vyskakovací sloupcové“ menu. Možnosti jsou stejné jako u TMenuBar, ale standardně je toto menu skryté a na vyvolání je použita metoda TMenuPopup.Popup(X, Y); - přičemž hodnoty X a Y udávají souřadnice, kde se má levý horní roh menu objevit. Jakmile je toto menu vyvoláno, je možné vybrat nějakou položku – menu se skryje a provede se akce, nebo nevybrat žádnou položku, „click-nout vedle“, čímž se menu taktéž skryje a žádnou akci neprovede.

Příklad je dodáván v adresáři EX10\3\ (Other page). K vyvolání MenuPopup slouží poslední položka z MenuBar. Chování příkazů menu je totožné jako u MenuBar – text vybrané položky je vypsán komponentou TStaticText.

11.17. TMenuOverlay

MenuOverlay je speciální typ hierarchického stromového menu, kde toto menu má přesně udanou pozici, jednotlivé položky menu jsou v jednom sloupci pod sebou, a v jednom okamžiku se zobrazuje seznam potomků jednoho menu (jedna úroveň menu). Výběrem jednotlivé položky je vyvolána definovaná akce, výběrem položky obsahující podmenu (vytvořené pomocí NewSubMenu(…)) je toto podmenu zobrazeno touto komponentou (dostaneme se o jednu úroveň níže), výběrem položky nadmenu (vytvořené pomocí NewMenuItem(…) s příznakem akce mifReturn) se dostáváme o jednu úroveň výše.
Pokud je počet položek úrovně menu větší než je počet zobrazených řádek, pak se lze zobrazeným výřezem menu rolovat nahoru a dolu. K tomuto TMenuOverlay je možné přidružit posuvník

Ukázka použití tohoto MenuOverlay je v adresáři – EX10\4\ (Main page). Je zde možné si vyzkoušet chování kláves vkEnter a vkEsc.

11.18. TKeyPad

Komponentou pro zadávání údajů na display, pokud nemáme klávesnici k dispozici je komponenta TKeyPad. Jedná se o „klávesnici na obrazovce“.
Je možné si vytvářet vlastní klávesnice. Několik základních typů velikosti vhodné pro Touch11 již je připraveno, stačí v programu použít jednotku T11MKpad. Předdefinované klávesnice vytváříme tak, že zavoláme metodu customize s jednou z těchto tříd nastavení: [ccKeypadAlphaEn, ccKeypadAlphaCz, ccKeypadAlphaEnCz, ccKeyPadNumHex, ccKeyPadNumDec, ccKeyPadNumReal, ccKeyPadNumSmallReal, ccKeyPadNumSpec, ccKeyPadOkCancel].

Příklad použití TKeyPad bude uveden u Dialogu (TDialog).

11.19. TFrame

Tato komponenta je určena k vykreslení rámečku. Programátor tuto komponentu (většinou) nevytváří, neboť je implicitně používaná jako součást oken.

12. Knihovní skupinové komponenty (TGroup)

Skupina komponent je následník komponenty (**TControl**), obohacený o schopnost vlastnit komponenty. Přidává hlavně metody Insert a Delete pro vkládání a vybírání komponent a další metody pro práci se seznamem svých komponent.

12.1. TWindow

Window je jednoduchá skupinová komponenta, která obsahuje nadpis okna, rám a další vložené komponenty.

12.2. TDialog

Dialog je následník okna, který je určen k běhu v modálním stavu. Jedná se o komponentu, která většinou nezabírá celou obrazovku, ale přisvojuje si vykonávání ohniskových a pozičních událostí jen pro sebe.

Příklad obsahující dialog je v EX10\4\ (Other page). Na kliknutí na text zobrazující teplotu se objeví dialogové okno žádající změnu teploty. V tomto dialogovém okně můžeme zřetelně vidět rám, klávesnici (TKeyPad) a tlačítko „zpět“ provádějící ukončení editace a návrat bez potvrzení změny hodnoty.
Dialogová okna jsou zhruba na stejné úrovni jako stránky – tedy mají svou vytvářecí proceduru (v příkladu je to CreateEditDialog)

12.3. Stránky (TPage)

Stránka je abstrakce jedné obrazovky terminálu. Jedná se o skupinovou komponentu, která (většinou) zabírá celou viditelnou plochu terminálu a umožňuje vytvořit lineární (jednorozměrný dvousměrný) seznam stránek a dále pak obsahuje záznam, z které obrazovky byla vyvolána.

12.4. Správce stránek (TPageControl)

Správce stránek je skupinová komponenta obsahující jednotlivé stránky (a pouze stránky), schopná práce se stránkami (vytváření, skoky mezi stránkami, volání a návraty jednotlivých stránek).

12.5. Aplikace (TApplication)

Aplikace je nejvrchnější (skupinovou) komponentou určenou k (tranzitivnímu) vlastnistvní všech zobrazovatelných komponent a k obhospodařování vstupního a výstupního zařízení.

13. Časovače komponent - rozhýbání

Třetím možným zdrojem událostí (první je klávesnice, druhý je polohovací zařízení) je časovač. Jedná se o prostředek ke generování událostí **evTimer**.

13.1. Práce s časovači komponent

13.1.1. Naplánování časovače

Komponenta má metodu:

```
function TControl.ScheduleTimer( AId: Integer; ATime:
LongInt; AOptions : Word): Boolean;
```

Tato metoda slouží k alokování a naplánování časovačů.
Prvním parametrem metody je identifikační číslo časovače.
Druhý parametr udává délku periody v milisekundách.
Třetí parametr určuje, zdali se bude jednat o děj jednorázový(toOneShot), či opakovaný (toPeriodic).

13.1.2. Zrušení časovače

Komponenta má metodu:

```
procedure TControl.CancelTimer( AId: Integer );
```
Tato metoda zruší naplánování a dealokuje časovač.

13.1.3. Obsloužení zásahu časovače

Pokud časovač dosáhne naplánovaného času, vyvolá událost evTimer. Tato událost je standardně předána přímo komponentě, které časovač náleží. Tato komponenta vyvolá oznámení s kódem TNotification.Code = nmTimer, přičemž v položce TNotification.TimerId je číslo časovače.

13.2. Jednorázová změna

Pro některé komponenty je vhodné „naplánovat“ jejich změnu. Například vytváříme-li aplikaci, bývá první stránka nějaké logo, které sice je líbivé pro oko, ale nepřináší žádnou užitnou hodnotu a proto řádově po několika vteřinách chceme z této stránky přejít na stránku další. K tomuto úkolu byl stvořen časovač a tak jej použijeme. Vytvoříme uživatelskou proceduru obsluhující notifikaci, která při vstupu do stránky naplánuje časovač a při vypršení časovače přepne stránku na nějakou jinou.

Vhodný zápis části kódu programu tedy je:

```
procedure FirstPageNotify( var N: Tnotification );
begin
  case N.Control^.Id of
    cidFirstPage:
      begin
        case N.Code of
          nmEnter:
            begin
              N.Control^.ScheduleTimer( 1, 5000, toOneShot );
              ClearNotification( N );
            end;
          nmTimer:
            begin
              Tree.PageCtrl^.GotoPage( cidMainPage );
              ClearNotification( N );
            end;
          nmExit:
            begin
              N.Control^.CancelTimer( 1 );
              ClearNotification( N );
            end;
        end;
      end;
  end;
end;
```

13.3. Opakovaný časovač

Kromě jednorázového časovače je též možné použít časovač opakovaný. Zobrazujeme-li například nějaký technologický proces, kde se mění, měří a zobrazují

veličiny, bývá potřeba tyto veličiny periodicky aktualizovat. Předpokládejme, že zobrazované veličiny jsou v proměnných velA a velB, přičemž komponenty toto zobrazující jsou význačné komponenty Tree.MainPage.stA a Tree.MainPage.stB. Notifikační procedura hlavní stránky pak vypadá takto:

```
procedure MainNotify( var N: TNotification ); far;
begin
  case N.Control^.Id of
    cidMainPage:
      begin
        case N.Code of
          nmEnter:
            begin
              N.Control^.ScheduleTimer( 1, 500, toPeriodic );
              ClearNotification( N );
            end;
          nmTimer:
            begin
              Tree.MainPage.stA^.SetText( velA );
              Tree.MainPage.stB^.SetText( velB );
              ClearNotification( N );
            end;
          nmExit:
            begin
              N.Control^.CancelTimer( 1 );
              ClearNotification( N );
            end;
        end;
      end;
  end;
end;
```

13.4. Další informace k časovačům

Každá komponenta má své časovače. Tyto jsou v rámci jedné komponenty číslovány svými identifikátory – číslem typu Integer. Tedy komponenta A může mít časovač číslo 7, komponenta B taktéž časovač číslo 7 a tyto dva časovače jsou vzájemně nezávislé (tedy nejedná se o tentýž časovač).
Je třeba si dát pozor na to, že maximální počet v jednom okamžiku běžících časovačů je omezen na 16 !
Vzhledem k tomuto omezení (na 16 časovačů) je vhodné přiřadit časovače jednotlivým stránkám, přičemž při nmEnter tyto časovače naplánujeme a při nmExit je dealokujeme.

14. Kontroly

14.1. Nepovolení opuštění komponenty (změna ohniska)

Pokud je komponenta v ohnisku a nechce dovolit přechod ohniska na jinou komponentu, může tomu zabránit. Provádí se to takto: této komponentě nastavíme příznak ofValidate. Nyní při pokusu o změnu ohniska vysílá komponenta oznámení nmCanExit. Položka TNotification.Accept je nastavena na hodnotu **true**. Toto

oznámení je standardně šířeno a jakmile dorazí zpět vyvolávající komponentě, je testována tato položka TNotification.Accept. Má-li stále hodnotu **true**, dochází k přepnutí ohniska. Nastavil-li však někdo tuto položku na **false**, je přepnutí zakázáno.

14.2. Nepovolení ukončení modálního stavu

Pokud je komponenta v modálním stavu a nechce dovolit ukončení tohoto modálního stavu, může tomu zabránit. Provádí se to takto: pokud je vyvolán požadavek na ukončení modálního stavu, je vyvoláno oznámení- nmCanEndModal, přičemž položka TNotification.Accept je nastavena na hodnotu **true**. Toto oznámení je standardně šířeno a jakmile dorazí zpět vyvolávající komponentě, je testována tato položka TNotification.Accept. Má-li stále hodnotu **true**, dochází k ukončení modálního stavu. Nastavil-li však někdo tuto položku na **false**, je ukončení zakázáno.

14.3. Validátory

Validátory jsou objekty, které umožňují komfortní práci se změnami hodnot proměnných. Jsou definovány v knihovně Valids. Abstraktní předek TValidator umožňuje v zásadě 3 činnosti:

1) převod hodnoty proměnné na textovou reprezentaci
2) převod textové reprezentace na hodnotu proměnné
3) kontrola (validace) textové hodnoty

Tyto validátory nemají nic společného s již uvedeným příznakem ofValidate v příznakové položce TControl.Options.

14.3.1. Typy validátorů

Z abstraktního předku TValidator jsou odvozeny tři podtypy:

1) Validátor ordinálních hodnot (TOrdinalValidator)
   Z ordinálního validátoru jsou odvozeny:
   a. Numerické validátory (Byte, ShortInt, Word, Integer, Longint)
   b. Výčtové validátory (ByteEnum, WordEnum)
   c. Booleovský validátor
2) Validátor textových řetězců (TTextValidator)
   a. Validátor řetězců (TStringValidator)
   b. Validátor pole znaků (TCharArrayValidator)
3) Validátor reálných čísel (TRealValidator)

14.3.2. Příklad použití validátoru

Řekněme, že chceme editovat celé kladné číslo v desítkové soustavě od 1 do 100. Jako nejmenší použitelný vhodný datový typ je Byte a tak použijeme TByteValidator.

```
var
  hodnota: byte;
```

```
  V: PByteValidator;
  text: string;
  result : integer;
...
begin
  hodnota := 17;
...
  V := new( PByteValidator, Init( @hodnota, nvfRadixDec, 1, 100 ));
...
```

Nyní máme proměnnou hodnota chráněnou validátorem. Pokud budeme k této proměnné přistupovat pouze přes validátor pomocí metody TransferText, máme zajištěnu konzistenci a validitu hodnoty.
Pro vyčtení textové reprezentace z validátoru použijeme:

```
  result := V^.TransferText( @text, 5, vmLoad );
```

Pro validaci textové reprezentace řetězce použijeme:

```
  result := V^.TransferText( @text, 5, vmValidate );
```

Pro převod textu na vnitřní hodnotu validátoru použijeme:

```
  result := V^.TransferText( @text, 5, vmStore );
```

(při převodu textu na vnitřní hodnotu se nejprve provede validace, a je-li úspěšná, je proměnná modifikována)

14.3.3. Ordinální validátor

Ordinální validátor je rozšíření abstraktního TValidator o metodu TransferOrdinal. Tato metoda poskytuje možnost vmLoad, vmValidate a vmStore nikoliv na text, ale na parametr kompatibilní s LongIntem.

14.3.4. Spolupráce komponent a validátorů

Komponenta obsahuje položku validator. Této položce můžeme přiřadit konkrétní instanci validátoru a používat metody pro vzájemnou spolupráci komponenty a validátoru.
Pro spolupráci je určena metoda TControl.Transfer(AMode: Integer) : Integer; , která jako parametr přijímá jeden příkaz (vmLoad, vmStore nebo vmValidate) a výsledkem je konstanta s prefixem ve_ oznamující úspěch/příčinu neúspěchu provedení operace.
Pokud si přejeme načíst hodnotu z proměnné přes validátor do komponenty, provedeme: TControl.Transfer(vmLoad); . Pokud naopak máme zájem obsah/vnitřní stav komponenty přenést do proměnné, zavoláme: TControl.Transfer(vmStore); . Druhá spolupracující metoda je TControl.Validate : boolean; . Tato metoda vyvolává validaci hodnoty a vrací úspěch/neúspěch.

14.3.5. Roztřídění jednotlivých typů validátorů

14.3.5.1. Číselné validátory

Do této oblasti spadají validátory, které spolupracují s číselnými hodnotami. Jedná se o numerické validátory (TByteValidator, TShortIntValidator, TWordValidator, TIntegerValidator, TLongIntValidator) a o validátor datového typu real.

Zjednodušený konstruktor požaduje čtyři parametry: adresu datové proměnné, příznakový byte, minimální povolenou a maximální povolenou hodnotu.
Rozšířený konstruktor umožňuje používat tyto čísla jako čísla s pevnou řádovou čárkou a určitým počtem desetinných míst, takže konstruktor vyžaduje šest parametrů – kromě 4 standardních ještě počet desetinných míst a měřítko. Počet desetinných míst je zcela zřejmý. Měřítko je (celé) číslo, které udává, kolikrát je skutečná hodnota menší než jeho vnitřní reprezentace ve zvoleném datovém typu.

Validátor datového typu real nemá zjednodušený konstruktor.

14.3.5.2. Výčtové validátory

Standardně jsou podporovány dva výčtové typy – na Byte a na Wordu.
Konstruktor má celkem 3 parametry: první je adresa proměnné, druhý je počet položek výčtového typu a třetím parametrem je (aplikačně definovaná) funkce provádějící převod čísla na příslušný řetězec. Metoda TransferText výčtových validátorů implementuje pouze vmLoad. Na vmStore a vmValidate vrací veNotImpl. Spolupráce s komponentami je prováděna pomocí metody TransferOrdinal.

14.3.5.3. Booleovský validátor

Konstruktor booleovského validátoru má tři parametry. Prvním z nich je klasicky adresa datové proměné (typu boolean) a další dva jsou řetězce pro hodnotu **false** a pro hodnotu **true**. Metoda TransferTExt implementuje pouze vmLoad. Na vmStore a vmValidate vrací veNotImpl. Spolupráce s komponentami je prováděna pomocí metody TransferOrdinal.

14.3.5.4. Textové validátory

Další skupinou jsou textové validátory, které umožňují pracovat s datovými typy string a array of char.

14.3.6. Příklad

Příklad ukazující použití validátorů současně s komponentou TListEdit je dodáván v EX10\3\ (third page). Numerické validátory dovolují editovat hodnotu, řádek s výčtovým validátorem reaguje cyklickým nastavením následující hodnoty výčtového typu.

15. Editace in-line

Pokud máme zájem editovat nějakou položku na určitém místě, je možné zavolat editaci in-line. Tato editace je automatické vytvoření editační komponenty (TEdit) na

zvoleném místě přičemž pokud je editace potvrzena, je nová hodnota (přes validátor) předána do proměnné.
Pro editaci in-line existují tyto dvě funkce: InLineEdit a InLineEditEx. Příklad použití je v adresáři EX10\4\ (Third page).

16. Kurzor (caret)

I v těchto knihovnách existuje (textový) kurzor. V drtivé většině případů jej potkáte pouze u komponenty TEdit – při editaci textu.
Komponenta obsahuje položky TControl.Caret : TPoint; a TControl.CaretSize: TPoint; . Dále pak existuje stavový příznak sfCaretVis, který nastaven zobrazuje též nadefinovaný kurzor. Zobrazení a skrytí kurzoru se provádí metodami TControl.ShowCaret a TControl.HideCaret.
Aby byl kurzor vidět, je potřeba, aby komponenta byla v ohnisku.

17. Standardní dialogová okna komponenty TApplication – metoda MessageBox

Kořen vlastnictví – komponenta TApplication má metodu MessageBox. Tato metoda je určena k vyvolávání standardních dialogových oken. Deklarace této metody je následující:

```
function TApplication.MessageBox( ABounds: TRect;
  const ATitle, AText: TRString; AFlags : Word ): Word;
```

Prvním parametrem je umístění dialogového okna, druhý parametr je titulek okna, třetí parametr je vnitřní zobrazovaný text a čtvrtý parametr je zamýšlen jako pole příznaků určujících vzhled/funkčnost dialogového okna.

V tomto okamžiku (únor 2004) zatím toto funguje jen u terminálu Term10, přičemž příznakový word není využit a je zobrazeno pouze dialogové okno s titulkem a vnitřním textem (bez tlačítek). Zde je ovšem návod, jak si vytvořit svá vlastní dialogová okna.

Komponenta TApplication obsahuje procedurální proměnnou MessageBoxFunc deklarovanou takto:

```
MessageBoxFunc : function( ABounds: TRect;
  const ATitle, AText: TRString; AFlags : Word ): Word;
```

(tedy stejně jako TApplication.MessageBox). Pokud do této proměnné uložíme funkci, která vrací ukazatel na TCustomDialog, máme vyhráno. Na ukázku je zde uvedena funkce definovaná pro Term10:

```
function T10MessageBoxFunc( const ABounds: TRect;
  const ATitle, AText: string; AFlags: Word ): PCustomDialog; far;
var
  Dlg  : PDialog;
  Text : PStaticText;
  R    : TRect;
begin
  Dlg := New( PDialog, Init( ABounds, ATitle ) );
  Dlg^.Customize( ccDialog );
  Dlg^.GetExtent( R );
  Inc( R.A.Y, 16 );
```

```
  Dec( R.B.Y, 4 );
  Inc( R.A.X, 4 );
  Dec( R.B.X, 4 );
  Text := New( PStaticText, Init( R, AText ) );
  Text^.Customize( ccStaticText );
  Text^.TextFlags := tfLeft or tfTop or tfMultiline;
  Dlg^.Insert( Text );
  T10MessageBoxFunc := Dlg;
end;
```

Protože jsou tyto std. dialogová okna terminálově závislá, budou deklarovány v customizačních jednotkách XXXcus.pas pro jednotlivé terminály.

18. Mapování kláves

Vstupní ovladač klávesnice vytváří události TEvent.Code = evKeyDown. Dále vyplní příslušnou položku TEvent.KeyCode (jednou z konstant kbXXX) a TEvent.CharCode (pokud se jedná o standardní ASCII znak). Vzhledem k odlišnosti klávesnic terminálů a z důvodu snadné přenositelnosti/znovupoužitelnosti však aplikace nereagují přímo na konstanty kbXXX, nýbrž na tzv. virtuální klávesy. To jsou konstanty s prefixem vkXXX, které jsou uloženy v TEvent.VirtKey.

Každá komponenta obsahuje položku KeyMapper. Jedná se o funkcionální proměnnou převádějící jednotlivé kláveskové kódy kbXXX na vkXXX.

Metoda ProcessEvent zpracovávající události provede při stisku tlačítka toto: prochází od kořene (tapplication) až k zaostřené komponentě a ve všech komponentách zavolá tuto funkcionální proměnnou. Pokud tedy máme zájem předefinovat klávesy pro jednu komponentu, či stránku, vytvoříme mapovací funkci a uložíme jí do příslušné položky KeyMapper zvolené komponenty. Standardně komponenta TApplication má v této položce uložen DefaultKeyMapper vypadající takto:

```
function DefaultKeyMapper( KeyCode: Word ): Word; far;
begin
  case KeyCode of
    kbEnter      : DefaultKeyMapper := vkEnter;
    kbEsc        : DefaultKeyMapper := vkEsc;
    kbTab        : DefaultKeyMapper := vkNext;
    kbShiftTab   : DefaultKeyMapper := vkPrev;
    kbLeft       : DefaultKeyMapper := vkLeft;
    kbRight      : DefaultKeyMapper := vkRight;
    kbUp         : DefaultKeyMapper := vkUp;
    kbDown       : DefaultKeyMapper := vkDown;
    kbInsert     : DefaultKeyMapper := vkInsert;
    kbDelete     : DefaultKeyMapper := vkDelete;
    kbBackSpace  : DefaultKeyMapper := vkBackSpace;
    kbHome       : DefaultKeyMapper := vkHome;
    kbEnd        : DefaultKeyMapper := vkEnd;
    kbPageUp     : DefaultKeyMapper := vkPageUp;
    kbPageDown   : DefaultKeyMapper := vkPageDown;
    kbF1         : DefaultKeyMapper := vkHelp;
    kbAltX       : DefaultKeyMapper := vkAppExit;
    kbF6         : DefaultKeyMapper := vkNextWindow;
    kbShiftF6    : DefaultKeyMapper := vkPrevWIndow;
    kbClear      : DefaultKeyMapper := vkClear;
    kbF10        : DefaultKeyMapper := vkMenu;
  else
```

```
    DefaultKeyMapper := 0;
  end;
end;
```

19. Více komponent vložených do TApplication

Standardně je v aplikaci vložen pouze správce stránek a ten je zaostřen a obhospodařuje jednotlivé stránky. Pokud však začneme používat dialogová okna a tyto vyvolávat ať už pomocí Application^.ExecControl(dialog_okno); , či Application^.MessageBox(…); , dochází k přechodu ohniska a modálního stavu na toto nové dialogové okno. Tímto krokem je ovšem odejmuto ohnisko správci stránek – tím pádem i jednotlivým stránkám a komponentám v nich a tím přestávají tikat časovače (ty správně nadefinované u stránek na nmEnter a nmExit). To znamená, že komponenty se nepřekreslují i přes to, že je můžeme vidět (neboť ne všechny dialogy nutně zabírají celou plochu displaye).

Pokud nutně potřebujeme prohlížet všechny události, které přicházejí od vstupního ovladače (například stisk tlačítka „stop“ na klávesnici terminálu), máme k tomu procedurální položku TApplication.OnEvent. Ta má úplně stejný typ jako již známé procedury BeforeHandle a AfterHandle.

20. Další drobnosti, které by se mohly hodit

20.1. Kam s watchdogem procesu vizualizace

(doporučení:) Zpracování událostí je prováděno smyčkou GetEvent – ProcessEvent. Pokud vyvolání GetEvent vrátí evNothing – žádná událost, nedochází ke zpracování – tedy nevolají se metody handleEvent, ale vyvolá se „místo nich“ procedurální proměnná TApplication.OnIdle. Tato procedura tvoří první vhodné místo pro občerstvení WatchDogu. Pokud však bychom měli velký počet událostí, které bychom téměř nestačili zpracovávat, procedura OnIdle by volána nebyla a tak je ještě zapotřebí další občerstvení WatchDogu při příchodu libovolné neprázdné události – což je procedurální proměnná TApplication.OnEvent.
Tedy občerstvování watchdogu procesu vizualizace je vhodné vložit do TApplication.OnIdle a zároveň TApplication.OnEvent.

20.2. Jazykové mutace a resource stringy

Zatím jsme všude používali obyčejné textové řetězce typu string. Tyto vizualizační knihovny umožňují i více jazykových mutací, dokonce i za běhu přepínaných. Pokud potřebujete právě toto (jazykovou lokalizaci), podívejte se na dokumentaci od jednotky RString.

20.3. Paleta – barvy komponent

Zatím veškeré ukázky byly provozovatelné na monochromatickém display. Tyto knihovny však poskytují možnost využití barev. Pokud máte zájem využívat barvy, podívejte se do manuálu ke knihovně CONTROLS na odstavec 4.5.1 – Paleta komponenty.