Programování uživatelského rozhraní

Tato kapitola popisuje, jak pomocí standardních procedur a funkcí, objektových metod a vlastností ovládat chování formulářů, menu, sestav a dalších prvků tvořících uživatelské rozhraní databázové aplikace.

Programování versus akce

K integrací formulářů a dalších objektů do uživatelského rozhraní lze využít také *akcí* způsobem popsaným v kapitole o vytváření aplikace. Nástroje, které popíšeme na tomto místě, jsou flexibilnější a poskytují vyšší stupeň kontroly nad vizuálními objekty. Jejich využití přitom často nevyžaduje skutečné programování, v mnoha případech stačí prostě zavolat vhodnou funkci nebo provést přiřazení hodnoty do vlastnosti.

Alternativy

Pro úplnost dodejme, že s obsahem databáze ve **WinBase602** lze pracovat i bez použití jejích vizuálních objektů například některou z těchto cest:

- volat z programu v externím jazyce pouze funkce serveru pro případnou komunikaci s uživatelem využít ***Windows*** API;
- využít objekty pro přístup do databází přes rozhraní ODBC nabízené např. ve vývojových prostředích ***Borland Delphi***, ***MS Visual C*** nebo ***Visual Basic***, v dotazovacích systémech jako ***Crystal Report***, ***MS Query*** apod.;
- vstupovat do databáze po Internetu pomocí WWW prohlížeče přes rozhraní CGI.

Těmto způsobům práce s **WinBase602** jsou věnovány jiné kapitoly manuálu.

Principy řízení uživatelského rozhraní

Databázové aplikace, jejich uživatelské rozhraní je vytvořeno nástroji **WinBase602**, lze podle způsobu naprogramování tohoto rozhraní rozdělit do 3 druhů:

Aplikace řízena menu nebo formulářem

Je-li startovním objektem aplikace menu nebo formulář, pak se tento objekt při spuštění aplikace otevře a další činnost aplikace je řízena tím, co dělá uživatel. Tím, že uživatel například provede příkaz z menu nebo stiskne tlačítko ve formuláři, vyvolá další akce, například otevírání jiných formulářů, přenosy dat, tisk sestav, změnu menu atd. Uzavřením startovního objektu aplikace skončí.

Pokud aplikace obsahuje program, pak pouze jako projekt, tedy proto, aby v něm byly deklarovány proměnné a naprogramovány procedury a funkce, které budou z formulářů a menu zavolány.

Aplikace řízena programem s pevnou posloupností akcí

Tento druh aplikace obsahuje program, jehož struktura odpovídá posloupnosti akcí, které má aplikace provést. Program může například napřed provést inicializaci, pak si vyžádat od uživatele vstupní údaje, na jejich základě provést změny v databázi nebo vypsat některá data a pak se rozhodnout, zda bude žádat další údaje nebo skončí. Během práce program otevírá tzv. *modální* formuláře, tedy formuláře, která pozastaví jeho běh a umožní uživateli zadávat nebo prohlížet si data, a po zavření každého takového formuláře pokračuje v činnosti.

Menu a nemodální formuláře se za běhu takového programu nedají použít - na obrazovce by se sice otevřely, ale nezastavily by běh programu, a proto by s nimi uživatel nemohl pracovat.

Tento typ programu se hodí pro ty aplikace, které předpokládají pevnou posloupnost akcí. Hodí se také, chcete-li uživateli poskytnout omezenou svobodu tím, že mu z jednoho modálního formuláře umožníte otevřít některé z dalších modálních formulářů a takto volit postup práce až do uzavření výchozího formuláře. Nehodí se pro situace, kdy uživatel potřebuje volně přecházet mezi současně otevřenými formuláři nebo vybírat si další činnost z menu.

Aplikace řízena programem se smyčkou zpráv

Pružné interaktivní aplikace se nejčastěji budují s využitím konstrukce, které se říká SMYČKA ZPRÁV. Smyčka zpráv je cyklus, v němž program čeká na zprávu od uživatele, a když přijde, pak na ní předepsaným způsobem reaguje. Jednotlivým zprávám jsou přiřazeny reakce programu, mezi nimiž je jedna, která způsobí opuštění smyčky a ukončení programu.

Typická smyčka zpráv

Aplikační program začíná pracovat v situaci, kdy je na obrazovce prázdné hlavní okno bez menu. Zpravidla nejprve provede potřebné inicializační akce a pak umístí na obrazovku výchozí ovládací objekty - menu nebo formuláře. Poté vstoupí do smyčky zpráv, v níž pomocí funkce **`Get_ext_message`** získává další čekající zprávu a zpracovává ji. Ve smyčce setrvává do té doby, dokud si v reakci na vhodnou zprávu nenastaví příznak ukončení běhu. Typická struktura programu vypadá takto:

```
CONST zpráva_1 = 1001;
      zpráva_2 = 1002;
      zpráva_konec = 1999;
VAR konec : Boolean;
    msg : integer;  handle : window_id;
.........................................
BEGIN
  { inicializace }
  .......................................
```

```
  { otevření výchozího menu nebo formuláře: }
  Main_menu('*hlavní_mn');
  ......................................
  konec := FALSE;
  WHILE NOT konec AND
        Get_ext_message(msg, handle, nil) DO
  BEGIN
      CASE msg OF
          zpráva_1: ...zpracování zprávy 1 ...
          zpráva_2: ...zpracování zprávy 2 ...
          ..................................
          zpráva_konec: konec := TRUE
      END
  END;
  { úklid }
  ......................................
END.
```

Funkce **Get_ext_message** čeká, dokud nepřijde další zpráva, a poté vrátí její číslo ve výstupním parametru msg. Během čekání zároveň doručuje běžné *služební* zprávy oknům na obrazovce a tím umožňuje uživateli s nimi interaktivně pracovat, provádět příkazy z menu a vůbec komunikovat s aplikací. Funkce **Get_ext_message** vrátí hodnotu FALSE, když dostane zprávu s číslem -1, čehož lze také využít k ukončení smyčky.

Činnost vzorového programu ukončí příchod zprávy zpráva_konec. Ostatní zprávy jsou zpracovávány v příslušné větvi příkazu CASE nebo jsou ignorovány.

Při použití smyčky zpráv není nutné, aby přes ní procházeny všechny operace, a zpravidla tomu tak není. Většina operací se v aplikací provádí tak, že formulář, menu nebo ovládací lišta přímo provedou potřebnou akci nebo zavolají vhodný podprogram.

Jaké zprávy dostává aplikační program?

Aplikační program dostává prostřednictvím funkce **Get_ext_message** zprávy vznikající trojím způsobem:

- Provedením *akce* „**Zaslat programu zprávu**“. Taková akce může být vyvolána např. příkazem z menu, stiskem tlačítka ve formuláři nebo na liště, provedením změny ve složce formuláře, otevřením nebo zavřením formuláře apod. Akce tohoto druhu se příliš často nepoužívají, protože místo posílání zpráv je jednodušší přímo zavolat vhodnou funkci programu.
- Provedením operace, které je přiřazena automaticky generovaná zpráva, např. otevřením textového editoru.
- Stiskem klávesy, které je přiřazena zpráva pomocí funkce **Register_key**.

Číslování zpráv

Zprávy použité v aplikaci (kromě zpráv automaticky generovaných) označuje její autor čísly v rozsahu od 1001 do 60000. Zpráva s číslem -1 má vyhrazený význam - je to žádost o ukončení aplikace. Tato zpráva sice běžící aplikaci sama neukončí, ale způsobí, že funkce **`Get_ext_message`** po jejím přijetí vrátí hodnotu FALSE, na níž může program reagovat ukončením své činnosti.

Čísla zpráv v rozsahu od 0 do 1000 a větší než 60000 jsou rezervována pro WinBase602 **a není dovoleno jich používat.**

Pro označení zpráv je výhodné použít konstant definovaných v programu ve vnitřním jazyce. Je-li tento program nastaven jako projekt (viz kapitola o programovacím prostředí), lze identifikátory zpráv využít v návrhu formulářů a menu.

Smyčka zpráv s funkcí `Peek_message`

Funkce **`Peek_message`** se od **`Get_ext_message`** liší tím, že pokud na vstupu žádná zpráva není, pak nečeká, skončí a vrátí FALSE. Hodí se k implementaci tzv. aktivního čekání, když program si zároveň dělá svou práci a zajišťuje funkčnost uživatelského rozhraní.

Kdy použít **`Peek_message`**

V některých situacích program potřebuje umožnit uživateli práci s formulářem pouze po určitou dobu a pak pokračovat ve výpočtu. Pokud okamžik, kdy má výpočet pokračovat, není ohlášen žádnou zprávou, pak je nutno místo funkce **`Get_ext_message`** použít **`Peek_message`**.

Příklad:

```
View_open('*form_x');
WHILE NOT ... podmínka ukončení ...
    DO Peek_message;
View_close('*form_x');
```

Tento úsek programu otevře formulář, umožní uživateli s ním pracovat do doby, dokud není splněna podmínka ukončení, pak formulář zavře a pokračuje.

Princip programování se smyčkou zpráv

Při programování aplikací se smyčkou zpráv musíte respektovat toto pravidlo:

Pokud má program reagovat na činnost uživatele, musí v cyklu volat funkce `Get_ext_message` **nebo** `Peek_message`**. Během provádění cyklu neobsahujícího opakované volání žádné z těchto funkcí počítač na činnost uživatel nereaguje.**

Dobu, po kterou není k dispozici žádná zpráva určená programu, program tráví uvnitř funkce **`Get_ext_message`**, kde vyřizuje služební zprávy a tím umožňuje uživateli normální práci s menu a okny na obrazovce.

Pokud při práci s formulářem nebo menu uživatel vyvolá nějaké akce (např. stiskne tlačítko, které volá proceduru), pak provádění těchto akcí se *vnoří* do volání funkce **`Get_ext_message`** nebo **`Peek_message`** – proběhne uvnitř ní.

Automaticky generované zprávy

Během práce uživatele s formulářem je automaticky generována řada zpráv s pevně určenými čísly dle níže uvedené tabulky. Při příjmu těchto zpráv funkce **`Get_ext_message`** vrátí v parametru *handle* hodnotu handle okna, které zprávu vyslalo.

Číslo zprávy	Událost
1	Formulář byl otevřen
2	Formulář byl uzavřen
3	Uživatel přešel na nový záznam
4	Uživatel změnil obsah složky
5	Vybraný záznam se překresluje
6	Celý formulář se překresluje
7	Formulář přechází k subkurzoru (QBE)
8	Formulář se vrací k superkurzoru
9	Otevřen závislý formulář
10	Uzavřen závislý formulář
11	Otevřen textový editor
12	Uzavřen textový editor
13	Otevřeno okno pro prohlížení obrázku
14	Uzavřeno okno pro prohlížení obrázku

Na tyto zprávy program reaguje pouze tehdy, pokud chce ošetřit určitou situaci, která je zprávou signalizována. Ostatní zprávy program ignoruje.

Příklad:

Chce-li program zachytit okamžik, kdy uživatel změní obsah některé složky ve formuláři ID, pak analyzuje docházející zprávy takto:

```
Get_ext_message(msg, handle, nil);
IF (handle = ID) AND (msg = 4)
    THEN ...    { reakce na změnu obsahu složky }
ELSE ...        { zpracování ostatních zpráv }
```

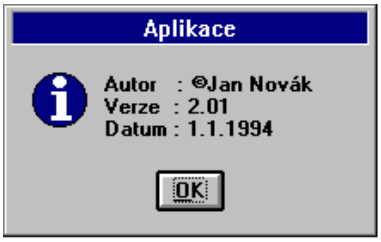
Vstup a výstup ve standardních oknech

Předávání informací mezi aplikací a uživatelem může probíhat dvěma cestami: buď prostřednictvím formulářů, anebo pomocí jednoduchých standardních oken.

Výstup zprávy

Pro výstup zprávy, kterou uživatel musí před pokračováním programu potvrdit, se hodí akce **Zobrazit okno se zprávou** nebo procedura **`Info_box`**. Otevře na obrazovce okno se zprávou a počká, dokud uživatel nestiskne potvrzující tlačítko. Text zprávy lze členit do řádek pomocí oddělovacího znaku s kódem 10 (LF).

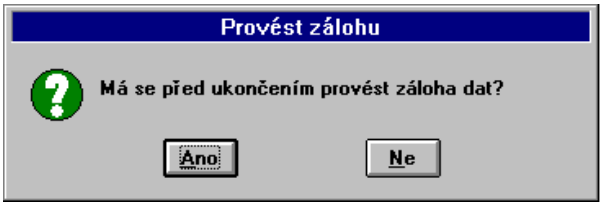
Info_box



Otázka typu ANO - NE

Pro položení otázky a získání odpovědi **ANO** nebo **NE** slouží funkce **`Yesno_box`**. Otevře okno s otázkou a dvěma tlačítky a dá uživateli možnost kladné nebo záporné odpovědi.

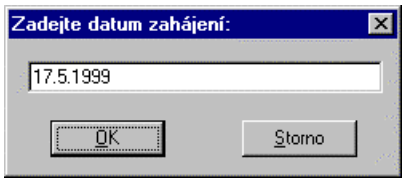
YesNo_box



Vstup jednoduchého údaje

Pro vstup údaje jednoduchého nestrukturovaného údaje (např. čísla, data, jména) slouží funkce **`Input_box`**. Umožní uživateli zadat v dialogovém okně řetězec znaků. Program si jej pak může konvertovat na vhodný typ.

Input_box



Potřebujete-li, aby uživatel zadal více než jeden údaj, pak použijte místo tohoto standardního okna formulář do proměnných projektu.

Informační řádek

Na informační řádek může program psát pomocí *akce* **Na stavový řádek vypsat zprávu** nebo procedur **`Set_status_text`** a **`Set_status_nums`**.

Informační řádek

Hlášení chyby

Zprávu o poslední chybě při provádění *databázové* operace podává procedura **`Signalize`**.

Ovládání formulářů

Základní jednotkou komunikace mezi uživatelem na jedné straně a programem nebo obsahem databáze na straně druhé je formulář. Existuje několik alternativních cest, jak programově manipulovat s formuláři. Tyto cesty lze kombinovat.

Tyto odstavce slouží jako vysvětlení principů práce s formuláři a obecný návod k jejich použití. Detailní popis jednotlivých funkcí, metod a vlastností naleznete v nápovědě.

Práce s formuláři pomocí standardních funkcí

WinBase602 nabízí řadu standardních funkcí vnitřního programovacího jazyka, které dovolují otevřít formulář, manipulovat s nim a s jeho obsahem a zavřít jej.

Každý otevřený formulář je identifikován číslem typu window_id**, kterému říkáme HANDLE formuláře.**

Funkce otevírající formulář vrací jeho handle. Funkce, které s formulářem manipulují, dostávají handle jako vstupní parametr. V příkazech, které jsou *součástí* formuláře, lze místo jeho handle použít symbol `!!`.

Pro tento způsob práce je charakteristické, že formulář *existuje* právě tehdy, když *je otevřeno jeho okno* na obrazovce.

Práce s formuláři deklarovanými jako objekty v programu

Objekty typu formulář deklarované v programu nabízejí nejsilnější a zároveň nejjemnější způsob práce. Formulář se v programu deklaruje takto:

```
VAR identifikátor_formuláře : FORM jméno_návrhu_formuláře;
```

Tato deklarace zavádí nový objekt označený *identifikátorem_formuláře*, vytvořený podle návrhu uloženého v databázi pod *jménem_návrhu_formuláře*.

Objekt typu formulář (formulářový objekt) lze deklarovat buď na úrovni programu nebo jako lokální v podprogramu, anebo jej lze vytvářet dynamicky procedurou **New**.

Vlastnosti a metody

Formulářový objekt lze ovládat pomocí jeho metod a vlastností. Vlastnost je hodnota sdružená s formulářem. Hodnotu vlastnosti lze přečíst a do vlastnosti lze také zapsat hodnotu novou. Metoda je podprogram sdružený s formulářem. Může mít parametry a může vracet hodnotu. Vlastnosti a metody se označují tak, že za identifikátorem formuláře se uvede tečka a jméno vlastnosti či metody.

Existence a otevření

Při práci s objekty typu formulář je oddělena *existence* formuláře od jeho *otevření* v okně na obrazovce. V existujícím objektu lze nastavit různé vlastnosti, například rozměry okna, vybraný záznam atd., a až pak jej otevřít. Formulář lze zavřít tedy odstranit z obrazovky, aniž by se ztratily jeho nastavené vlastnosti, a později jej opět otevřít. Při ukončení programu tyto objekty nezanikají a jejich okna se neuzavírají. Objekt definitivně zaniká až při zavření projektu (pro objekty deklarované v hlavním program), při opuštění podprogramu (pro objekty deklarované jako lokální v podprogramu) nebo při provedení procedury **Dispose** (pro dynamicky alokované objekty).

Nespecifikované formuláře

Formulářové objekty nelze mezi sebou přiřazovat, do podprogramu je lze předávat pouze referencí. Pokud deklarujete referenční parametr podprogramu nebo ukazatel typu formulářový objekt, pak lze jméno návrhu formuláře z deklarace vypustit. Ukazateli nebo referenčnímu parametru deklarovanému bez jména návrhu formuláře lze přiřadit libovolný formulář, ukazateli nebo referenčnímu parametru deklarovanému se jménem návrhu formuláře lze přiřadit pouze formulář se stejným návrhem.

Práce s nedeklarovanými formulářovými objekty

S formulářovými objekty lze pracovat i zjednodušeným způsobem, bez jejich deklarace v programu. V takové případě objekt nemá vlastní identifikátor a místo něj se používá jméno, pod nimž je navržený formulář uložen v databázi jakou součást aplikace.

Tato cesta nevyžaduje deklarování objektů, má však jistá omezení:

- nelze současně otevřít více instancí formuláře podle stejného návrhu;
- nelze manipulovat s formulářem předtím, než je zobrazen na obrazovce, protože jakmile se pokusíte odkázat na neotevřený formulář, automaticky se otevře.

Odkazování na sebe sama

Příkazy, které jsou součástí návrhu formuláře, mohou odkazovat na tento formulář pomocí klíčového slova **THISFORM**. Toto slovo nahrazuje identifikátor objektu nebo jméno návrhu a hraje podobnou roli, jako symbol **!!** při manipulaci prostřednictvím standardních funkcí.

Otevření formuláře na obrazovce

K otevření formuláře slouží metoda **Open**. Je-li v aplikaci uložen navržený formulář **Evid_list**, pak v programu můžete deklarovat objekt:

```
VAR evid_list_form : FORM Evid_list;
```

a otevřít jej příkazem **evid_list_form.Open(nil)**. Jednodušeji lze stejný formulář otevřít i bez deklarace objektu příkazem **Evid_list.Open(nil)**.

Formulář se dá na obrazovce otevřít také řadou *akcí* nebo *funkcí* podle toho, k jakému účelu je otevírán. Při otevírání formuláře pomocí funkce lze jeho vzhled a chování ovlivnit předáním různých (níže popsaných) příznaků v parametru *flags*. Jednotlivé příznaky lze libovolně kombinovat tím, že jako parametr předáte jejich *součet*.

Jméno akce	Jméno funkce	Výsledek
Otevřít (modální) formulář, Na stejný záznam otevřít (modální) formulář	**Open_view**, případně s příznaky MODAL_VIEW nebo PARENT_CURSOR	Otevření normálního formuláře
Otevřít relační formulář	**Relate_record**	Otevření relačního formuláře
Otevřít vázací formulář	**Bind_records**	Otevření vázacího formuláře
	Select_records	Výběr záznamů
	View_open	Otevření normálního formuláře (zastaralé)

Funkce **Relate_record** a **Bind_records** se volají pouze z formuláře, který chce otevřít další formulář za účelem editování relace mezi záznamy (viz kapitola o vytváření aplikace). Nevolají se nikdy z programu.

Dynamické vytvoření návrhu formuláře

Kromě formulářů, jejichž návrh je uložen v databázi jako součást aplikace, lze otevírat i formuláře, jejichž návrh byl vytvořen dynamicky a v databázi není.

Text návrhu formuláře lze předat jako první parametr všem funkcím, které otevírají formulář, například **Open_view**. Alternativně lze v programu deklarovat formulářový objekt bez uvedení jména návrhu a text návrhu mu dynamicky přiřadit voláním metody **Set_source**. Tyto možnosti se však příliš běžně nepoužívají, protože struktura zdrojového textu formuláře není dokumentována.

Výjimkou je standardní formulář zobrazující obsah tabulky, odpovědi na dotaz nebo kurzoru otevřeného v programu. Jeho zdrojový text lze zapsat jednoduše takto:

pro tabulky `'DEFAULT` *`jméno_tabulky`* `TABLEVIEW'`

pro pevný kurzor	'DEFAULT *jméno_kurzoru* CURSORVIEW'
pro proměnný kurzor	'DEFAULT *číslo_otevřeného_kurzoru*'

Otevření formuláře s takovýmto zdrojovým textem odpovídá provedení akce **Otevřít standardní formulář do tabulky** a **Otevřít standardní formulář do dotazu**.

Příklad:

```
VAR
  c : cursor;
  s : string[20];
  id : window_id;

BEGIN
  Open_view('DEFAULT TAB1 TABLEVIEW',no_redir,MODAL_VIEW,0,0,nil);
  Open_view('DEFAULT CURS CURSORVIEW',no_redir,MODAL_VIEW,0,0,nil);
  IF NOT Open_sql_cursor(c,'select * from TAB1 order by A')
  THEN BEGIN
    s :='DEFAULT '+int2str(c);
    Open_view(s,no_redir,MODAL_VIEW,0,0,nil);
    Close_cursor(c);
  END;
END.
```

Nastavení zdroje dat pro formulář

Jméno zdroje dat je součástí návrhu formuláře. Formuláři lze však přiřadit jiný zdroj dat a tím dosáhnout například zpřístupnění záznamů z odpovědi na dynamicky vytvořený dotaz.

Funkce **Open_view**, **Print_view**, **Relate_record**, **Bind_records** a **Select_records** umožňují pomocí parametru *base* zadat číslo otevřeného kurzoru, který bude zdrojem dat. Pokud se má použít zdroj dat uvedený v návrhu, pak se jako parametr *base* uvede konstanta NO_REDIR.

Zdroj dat lze zjišťovat a měnit také v existujícím formuláři, viz dále.

Varianty chování formuláře na obrazovce

Formulář lze otevřít na obrazovce třemi způsoby: jako normální okno uvnitř hlavního okna aplikace (MDI-child), jako plovoucí okno (popup, modeless) nebo jako modální okno blokující zbytek aplikace (modal).

Způsob otevření formuláře je popsán ve vlastnostech jeho návrhu. Mimo to lze otevření normálního formuláře v plovoucí nebo modální podobě vynutit pomocí příznaku

MODELESS_VIEW nebo MODAL_VIEW předaného jako parametr funkci otevírající formulář, případně voláním akce **Otevřít modální formulář**.

Funkce, akce nebo metoda, která otevřela modální formulář, neskončí, dokud tento formulář není zavřen. Proto taková funkce či metoda nikdy nevrací handle tohoto formuláře (byl by již neplatný), ale hodnotu 1 při úspěchu a 0 při chybě.

Příklad:

```
Open_view("*NOVA_KNIHA",NO_REDIR,MODAL_VIEW,0,0,id_nk);
```

Omezení uživatelovy manipulace s formulářem

Rozsah dovolených manipulací s formulářem je součástí jeho návrhu, dá se však dodatečně změnit. Při otevírání formuláře lze pomocí příznaků zakázat některé činnosti, které jsou v návrhu povoleny.

- `NO_EDIT` - v formuláři je znemožněna editace
- `NO_INSERT` - v formuláři je znemožněno vkládání nových záznamů
- `NO_DELETE` - v formuláři je znemožněno rušení záznamů
- `NO_MOVE` - v formuláři je znemožněno přecházení ze záznamu na záznam.

Je-li povoleno vkládání nových záznamů, pak nedoporučujeme zakazovat editaci, jinak by se do nově vložených záznamů nedalo nic zapsat.

V formulářích, které vedou do kurzoru, se automaticky uplatní omezení na vkládání, rušení a přepisování podle charakteru kurzoru.

Ve formulářových objektech lze omezení manipulací dynamicky zjišťovat a měnit pomocí vlastností **`Editable`**, **`Insertable`**, **`Deletable`** a **`Movable`**.

Zrušené záznamy ve formuláři

Otevřete-li formulář s příznakem `DEL_RECS`, budou v něm obsaženy i zrušené záznamy.

Formulář do tabulky

Pokud formulář otevřený s tímto příznakem vede do tabulky, pak zrušené i nezrušené záznamy vypadají zcela stejně, a jedinou cestou, jak je odlišit, je využít sloupce DELETED, který je součástí každé tabulky. Tento sloupec může mít ve formuláři vlastní složku nebo například může ovlivňovat viditelnost nebo aktivitu jiných složek - třeba viditelnost nápisu „*Tento záznam je zrušený*“.

Formulář do odpovědi na dotaz

Pokud formulář vede do dotazu, pak se při konstrukci odpovědi použijí pouze nezrušené záznamy z tabulky resp. z tabulek, bez ohledu na přítomnost parametru DEL_RECS. Pokud však během práce s formulářem některý záznam zrušíte, pak:

- není-li použit parametr `DEL_RECS`, zrušený záznam zmizí z formuláře
- je-li použit parametr `DEL_RECS`, zrušený záznam sice zůstane v formuláři, ale bude mít ve svých složkách nápis **`Zrušeno`**.

Použití parametru DEL_RECS poněkud zrychluje práci z formulářem vedoucím do zdroje dat obsahujícího velmi mnoho záznamů. Doporučujeme využít tento parametr

zejména tehdy, pokud program otevře formulář na určitém záznamu a nedává uživateli možnost přejít na záznam jiný. Pak je vypouštění zrušených záznamů zcela zbytečné.

Ostatní příznaky

Další příznaky, které lze také předat otevíranému formuláři, jsou:

- `AUTO_CURSOR` - způsobí, že při zavírání formuláře se automaticky zavře také kurzor předaný jako zdroj dat pomocí parametru *base*; kurzor se uzavře i v případě, že se formulář nepodaří otevřít;
- `COUNT_RECS` - při otevření formuláře se spočítají všechny jeho záznamy; tím je zajištěna správná poloha táhla na pravém rámu formuláře a vypsání celkového počtu záznamů ve formuláři na stavové řádce; otevření formuláře může trvat delší dobu;
- `PARENT_CURSOR` - formulář použije kurzor a vyrovnávací paměti formuláře, předaného jako parametr *hParent,* a nastaví se záznam, který je v něm vybrán (k trvalé synchronizaci obou formulářů nedojde). Viz detailní popis v kapitole o vytváření aplikací.
- `QUERY_VIEW` - otevře formulář sloužící pro položení QBE dotazu vztahujícího se k formuláři, jehož handle je předán jako parametr *hParent*.

Závislá okna

Pokud při otevírání okna B předáte otevírající funkci jako parametr *hParent* handle formuláře A, pak se formulář B bude stále nacházet nad formulářem A a dokud bude otevřen, bude blokovat změny v A.

Příklad:

Formulář Edit_kniha se otevře nad formulářem s handle rovno idk.

```
Open_view('*EDIT_KNIHA',cx,MODAL_VIEW,-irec,idk,id_ek);
```

Manipulace s otevřeným formulářem

Polohu formuláře na obrazovce a jeho rozměry definují vlastnosti **`X`**, **`Y`**, **`Width`** a **`Height`**. Nadpis okna je ve vlastnosti **`Caption`**. Handle okna vrací metoda **`Handle`**.

Příklad:

Pokud máte deklarovaný formulářový objekt **`evid_list_form`**, pak pomocí příkazů:

```
evid_list_form.Caption := 'Evidenční list';
evid_list_form.Height := 2 * evid_list_form.Height;
```

mu přiřadíte nadpis „Evidenční list“ a zdvojnásobíte jeho výšku proti návrhu. Tyto operace lze provést před otevřením formuláře na obrazovce nebo po něm.

Zavření

Formulář se zavírá *akcí* **Zavřít formulář** nebo *funkcí* **`Close_view`** (příp. **`View_close`**) nebo metodou formuláře **`Close`**. Všechny otevřené formuláře lze současně zavřít funkcí **`Close_all_views`**.

Pomocí funkce nebo metody **Pick_window** lze aktivovat a vysunout do popředí vybraný formulář. Funkce **Active_view** vrací handle toho formuláře, který je právě aktivní.

Dva formuláře lze uvést do synchronizovaného stavu (ukazují tentýž záznam) pomocí funkce **Register_rec_syn**.

Zdroj dat

Číslo kurzoru, který je zdrojem dat, a nastavení příznaků v otevřeném formuláři zjišťuje funkce **Get_fcursor** a nastavuje funkce **Set_fcursor**. Číslo kurzoru lze také číst a měnit pomocí metody **Cursnum**.

Metoda **Help** vyvolá nápovědu definovanou pro formulář nebo jeho vybranou složku.

Dotazy

Pro otevření formuláře na kladení QBE dotazu resp. uspořádání slouží metody **OpenQBE** resp. **OpenSort**. Zadaný dotaz lze potvrdit voláním metody **AcceptQuery**. Je-li obsah formuláře ovlivněn položeným dotazem, zde dotaz odstranit metodou **CancelQuery**. Funkce nebo metoda **QBE_state** zjistí, zda formulář právě slouží ke kladení QBE dotazu nebo zadání uspořádání. Využívá se zejména v podmínkách viditelnosti a aktivity.

Příklad:

Program zjistí číslo kurzoru (zdroje dat), číslo aktuálního záznamu a otevře nový formulář do tohoto zdroje dat a na tento záznam:

```
if Get_fcursor(idc,cx,NIL) then
    if Get_view_pos(idc,irec,erec) then
        Open_view('*EDIT_CTEN',cx,MODAL_VIEW,-irec,0,id_ec)
```

Sledování, zda je formulář otevřen

Funkce, které otevírají formulář, stejně jako metoda **Open**, umožňují zadat adresu proměnné, do níž se při otevření formuláře zapíše jeho handle a při uzavření formuláře nula. Díky tomu program může snadno rozpoznat okamžik, kdy je formulář uživatelem uzavřen.

Příklad:

Do proměnné **id_vc** je předán handle formuláře:

```
Open_view('*VybrCtenar',NO_REDIR,0,0,0,id_vc);
repeat Peek_message until id_vc=0;
```

Pokud je proměnná **id_vc**, která byla předána tímto způsobem do formuláře, lokální v podprogramu, pak podprogram nesmí skončit dříve, dokud formulář nebude uzavřen. Jinak by při uzavírání formuláře došlo k přepsání již neexistující proměnné, což může přivodit vážnou chybu aplikace.

Zjistit, zde určitý formulář je otevřen, lze také pomocí metody **Is_open** nebo funkce **View_state**. Využívají se například v podmínkách aktivity nebo zatržení příkazů menu otevírajících formulář.

Způsob číslování záznamů ve formulářích

Záznamy ve formulářích jsou vzestupně očíslovány dvojím způsobem: absolutně a interně. Některé funkce vyžadují předání absolutního a jiné interního čísla záznamu.

Absolutní číslování

První číslování je nezávislé na formulářích - každý záznam má svoje absolutní číslo ve zdroji dat, kam patří. ABSOLUTNÍ (EXTERNÍ) ČÍSLO ZÁZNAMU v tabulce je neměnné po celou dobu existence záznamu s výjimkou zkompaktnění tabulky. Absolutní číslo záznamu v kurzoru je záznamu přiděleno v okamžiku otevření kurzoru. Absolutní (externí) čísla záznamů se používají při přímém čtení a zápisu dat do tabulek a kurzorů.

Interní číslování

Jakmile otevřete formulář, pak záznamy obsažené v formuláři se dají očíslovat tak, jak se ve formuláři objevují: první shora má číslo 0, druhý 1 a tak dále. Tomu říkáme VNITŘNÍ (INTERNÍ) ČÍSLOVÁNÍ ZÁZNAMŮ ve formuláři. Pokud ve formuláři do tabulky zrušíte nějaký záznam, pak záznam zmizí a vnitřní čísla všech záznamů následujících za ním se zmenší o jedničku.

Změna číslování po vložení záznamu

Vložením nového záznamu do formuláře se změní pořadová (interní) čísla těch záznamů, před nimiž se nový záznam objeví (to se může stát pouze ve formuláři do tabulky; ve formuláři do kurzoru se přidává vždy na konec). Absolutní čísla záznamů se tím nezmění. Jak pořadová tak i absolutní čísla záznamů ve formuláři rostou směrem od začátku formuláře k jeho konci.

Programové ovládání obsahu formuláře

Před čtením tohoto odstavce doporučujeme prostudovat popis cesty dat mezi formulářem a databází v kapitole *Vytváření aplikací* v knize *WinBase602 – Příručka uživatele*.

Pokud formulář obsahuje více než jeden záznam, pak se při použití metod a vlastností složek vždy pracuje se složkami ve *vybraném* záznamu. Před manipulací se složkou je proto třeba vybrat záznam, například přiřazením do vlastnosti Curpos.

Nový záznam lze vložit metodou **`Insert`**, běžný záznam zrušit metodou **`DelRec`**. Zrušit všechny záznamy (s potvrzovacím dotazem) umožňuje metoda **`DelAsk`**.

Výběr záznamu a složky ve formuláři

Vlastnost **`Curpos`** obsahuje interní číslo vybraného záznamu ve formuláři, vlastnost **`Curextrec`** externí číslo vybraného záznamu. Obě čísla se dají přečíst také funkcí **`Get_view_pos`** a nastavit funkcemi **`Set_int_pos`** a **`Set_ext_pos`**. Uvnitř formuláře lze na absolutní číslo běžného záznamu odkazovat symbolem `@`, na interní číslo symbolem `@@`.

V obecném formuláři vlastnost **CurItem** obsahuje číslo vybrané složky a umožňuje tedy vybrat složku na základě čísla. Číslo vybrané složky vrací také funkce **Current_item**. Vybrat složku lze také pomocí metody složky **SetFocus**.

Pohyb po záznamech ve formuláři umožňují metody **FirstRec**, **LastRec**, **PrevRec**, **NextRect**, **PrevPage** a **NextPage**. K pohybu po složkách v rámci běžného záznamu slouží metody **FirstItem**, **LastItem**, **PrevTab** a **NextTab**. Na nejbližší vyšší resp. nižší složku se lze dostat metodami **UpItem** a **DownItem**.

Každá složka obecného formuláře (kromě čáry a rámu) je z hlediska ***Windows*** oknem a má svůj *handle*. Handle složky v aktuálním záznamu lze zjistit pomocí metody složky **Handle**.

Přepisování obsahu a ovládání složek

Přepisování obsahu složek formuláře je způsobem, jak může program paralelně ovlivnit obsah databáze i obsah formuláře. Zapisovat hodnoty do databáze lze také přímo, bez účasti formuláře, způsobem popsaným v samostatné kapitole. Přepis hodnot složek se využívá zejména tehdy, pokud je třeba ve formuláři přednastavit některé hodnoty nebo pokud se některé hodnoty mají zkontrolovat a dopočíst na základě jiných uživatelem zadaných hodnot.

Metody a vlastnosti složek

Jednotlivé složky formuláře jsou označeny jménem uvedeným v návrhu formuláře. Složky mají své metody a vlastnosti, z nichž některé jsou společné zatímco jiné jsou specifické po konkrétní druh složky (třeba combo). Při dokazování na vlastnosti a metody složky uvedete identifikátor formuláře, tečku, jméno složky, tečku a jméno vlastnosti či metody.

Obsah složek lze přepisovat pomocí jejich vlastností **Text** a **Value**. Obě vlastnosti se vztahují ke stejným datům, ale jsou rozdílného typu. Zatímco **Text** obsahuje textový zápis hodnoty tak, jak se objevuje na obrazovce, **Value** je hodnota stejného typu, jako ve zdroji dat. V překládaném combu obsahuje vlastnost **Text** text napsaný na obrazovce, zatímco vlastnost **Value** obsahuje hodnotu zapisovanou do databáze. Nápisy a tlačítka mají vlastnost **Text**, ale nikoli **Value**.

Příklad:

Nechť ve formulářovém objektu **evid_list_form** je tlačítko jménem STORNO a dvě editační pole typu datum se jmény ZAHÁJENÍ a UKONČENÍ. Pak příkazy:

```
evid_list_form.UKONČENÍ.Value:=evid_list_form.ZAHÁJENÍ.Value+20;
evid_list_form.STORNO.Text := 'Konec';
```

nastaví datum UKONČENÍ na 20 dnů po datu ZAHÁJENÍ a změní nápis na tlačítku STORNO na „Konec“. Jméno vlastnosti **Value** (a tečku před nim) lze vypustit a psát:

```
evid_list_form.UKONČENÍ := evid_list_form.ZAHÁJENÍ + 20;
```

Metoda **IsNull** zjišťuje, zda ve složce je hodnota NULL, metoda **SetNull** zapisuje NULL do složky. Hodnotě NULL odpovídá prázdný řetězec ve vlastnosti **Text**.

Pro čtení nebo přepis hodnoty lze také využít funkcí **Set_item_value** a **Get_item_value**. spolupracujících s proměnnou typu **Untyped**. Přečíst nebo přepsat text ve složce obecného formuláře lze také pomocí zastaralých funkcí **Get_view_item** resp. **Set_view_item**, takto provedená změna se však nezapisuje do databáze a proto má velmi omezené použití.

Způsob zobrazení složek lze řídit pomocí vlastností **Format** a **Precision**.

Specifika složek

Stav označovacích čtverců a přepínačů se řídí vlastností **Checked**. V editačním poli, editovatelném combu a ve formátovaném textu lze pozici kurzoru nebo zvýrazněný úsek textu nastavit metodou **Selection**. Pro práci se složkou Podpis se používají metody **Signature_check**, **Signature_sign** a **Signature_state**. Ve složce Seznam lze zjistit nebo nastavit vybraný řádek pomocí vlastnosti **SelIndex**. Nabídku ve složce Combo lze aktualizovat metodou **ResetCombo**. Ve složce Záložky lze nastavit nebo zjistit číslo vybrané stránky pomocí vlastnosti **Page**. Nad složkou obsahující datum lze otevřít měsíční kalendář pomocí metody **Calendar**.

Hodnoty proměnné velikosti

Složky pracující s hodnotami proměnné délky dovolují zjistit nebo přepsat aktuální délku hodnoty pomocí vlastnosti **Length**, načíst obsah složky ze souboru pomocí metody **ReadFile**, zapsat obsah složky do souboru pomocí metody **WriteFile**. Obsah složek Raster, Text a OLE lze otevřít v samostatném okně pomocí metody **Open**.

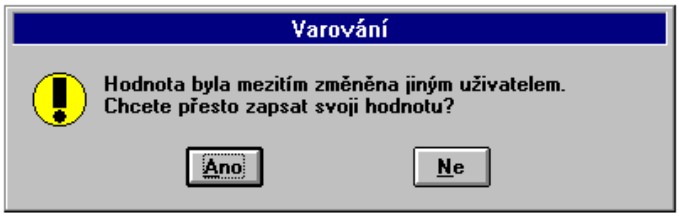
Pokud složka obsahuje hodnotu proměnné velikosti, pak pomocí vlastnosti **Value** lze s ní pracovat také po částech. Zápis **Value[offset,size]** označuje úsek hodnoty délky *size* počínaje bajtem *offset*. Zápis **Value#** označuje celkovou délku hodnoty.

Zápis změn do databáze a aktualizace obsahu formuláře

Po zapsání nové hodnoty do formuláře se záznamovou synchronizací je nutno pomocí funkce nebo metody **Commit_view** explicitně zapsat změny do databáze. Jinak by se snadno mohly ztratit. Změny provedené programem nebo uživatelem ve formuláři lze odvolat funkcí nebo metodou **Roll_back_view**.

Pokud byl obsah databáze změněn jinak než prostřednictvím formuláře, pak obsah formuláře lze aktualizovat funkci nebo metodu **Reset_view** s příznakem `RESET_CACHE`. Obsah jednotlivých složek formuláře ve vybraném záznamu lze aktualizovat metodou **Refresh**.

Příklad: v záznamově synchronizovaném formuláři je tlačítko volající proceduru, která v databázi přepíše některou hodnotu zobrazenou ve formuláři. V záznamu provedete nějaké změny a stisknete toto tlačítko. Jím vyvolaná změna se ve formuláři neprojeví. Při přechodu na jiný záznam nebo zavření formuláře se však může objevit varování signalizující, že se hodnota za dobu otevření formuláře změnila:



Aby tato matoucí situace nenastala, je nutno po změně hodnoty programem zavolat na složku metodu **Refresh** nebo na formulář funkci nebo metodu **Reset_views** příznakem RESET_CACHE, která uvede do souladu obsah formuláře s hodnotami v databázi.

Přepočty čísel záznamů

Kurzor a tabulka

Absolutní číslo záznamu ve formuláři do editovatelného kurzoru lze převést na absolutní čísla záznamů z tabulek, z nichž daný záznam vznikl, pomocí funkce serveru **Translate**.

Příklad:

```
function Exists(kod:string[8];var cisloVTab:integer) : boolean;
{************************************}
// zjistí, existuje-li kniha s kodem kod
// jestliže ano, vrátí v cisloVTab cislo záznamu z tabulky Knihy
var
  ck : cursor;
  podm2 : string[100];
  pocet : integer;
begin
  podm2 := "select * from KNIHY where KOD="""+kod+""""";
  if Open_sql_cursor(ck,podm2) then Exit;
  Rec_cnt(ck,pocet);
  if pocet = 1 then begin
    Translate(ck,0,0,cisloVTab);
    Exists := true              {OK}
  end
  else begin
    Info_box("Upozornění","Zadaný kód nebyl nalezen!");
    b := true;
    Exists := false;
  end;
  Close_cursor(ck);
end;
```

Subkurzor a superkurzor

Předpokládejme, že jeden kurzor je subkurzorem jiného kurzoru (budeme mu říkat superkurzor). Absolutní číslo záznamu v subkurzoru pak lze převést na absolutní číslo téhož záznamu v superkurzoru pomocí funkce **Super_recnum**. To je potřeba udělat např. v situaci, když uživatel stiskem tlačítka žádá provedení akce se záznamem, na němž se právě nachází. Pokud totiž předtím položil dotaz (např. QBE setřídění), pak jeho formulář

vede do subkurzoru. Program musí pomocí funkce **Get_fcursor** získat od formuláře jeho kurzor, pomocí funkce **Get_view_pos** získat číslo záznamu v subkurzoru a funkcí **Super_recnum** je přepočíst na číslo záznamu v superkurzoru (anebo provést požadovanou akci s pomocí získaného subkurzoru).

Tisk sestav

Tisk navržené sestavy

K vytištění sestavy lze použít *akci* **Vytisknout sestavu** (volanou odkudkoli) nebo *akci* **Vytisknout stejný záznam v sestavě** (volanou z vybraného záznamu ve formuláři). Funkce **Print_view** vytiskne všechny záznamy nebo jejich vybraný úsek, a to buď ze zdroje dat uvedeného v sestavě nebo z kurzoru předaného funkci jako parametr. Funkce **View_print** nabízí nejjednodušší tisk sestav.

Tisk obsahu formuláře do sestavy

Akce **Vytisknout aktivní formulář** a metoda **Print** tisknou obsah aktuální formuláře a přitom respektují provedený výběr a setřídění záznamů a případně také provedené úpravy vzhledu. Před tiskem vyvolají dialog pro volbu parametrů tisku.

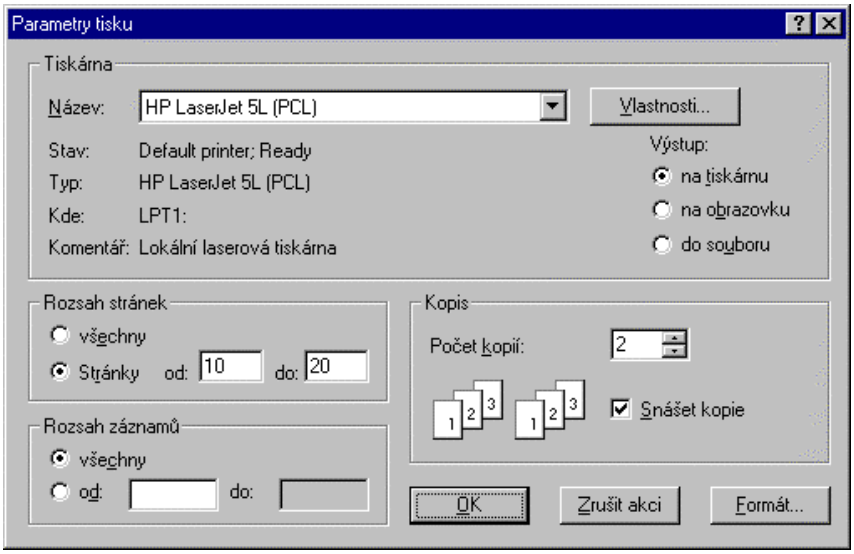
Tisk je ovlivněn řadou parametrů, které se dají nastavit dvěma cestami - buď interaktivně prostřednictvím dialogových oken nebo neinteraktivně předáním vhodných parametrů funkcím prezentační vrstvy.

Interaktivní volba tiskárny a nastavení parametrů

Základní dialogové okno pro volbu tiskárny a parametrů tisku se otevírá *akcí* **Nastavit parametry tisku** nebo funkcí **Print_opt**.

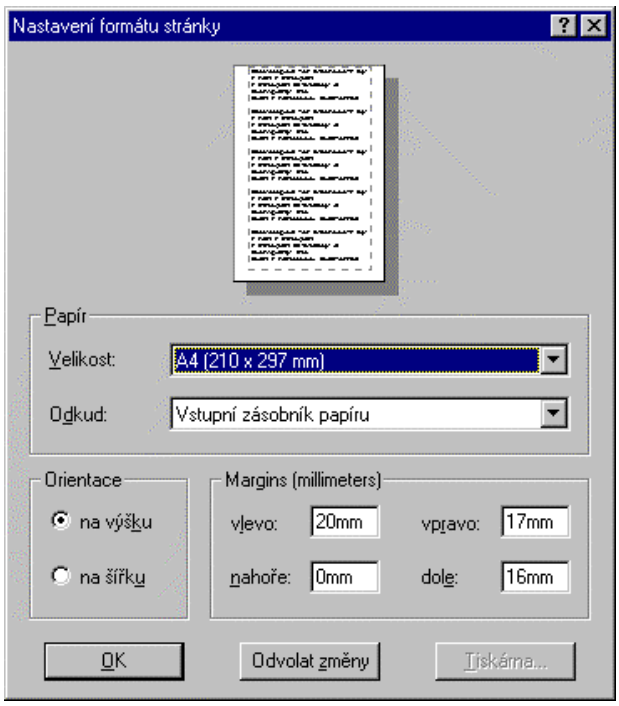
Stiskem tlačítka **Formát** lze z tohoto okna otevřít další dialogové okno umožňující nastavení formátu stránky.

Okno otevřené funkcí Print_opt



Okno pro nastavení formátu stránky se dá otevřít také přímo, a to *akcí* **Nastavit formát stránky** nebo funkcí **`Page_setup`**.

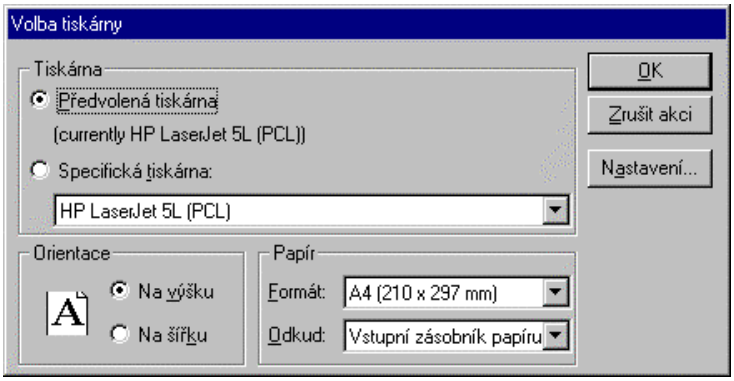
Okno otevřené funkcí Page_setup



Nákres obsahu stránky v tomto dialogu slouží při přípravě tisku štítků také k výběru pozice toho štítku na stránce, který má být vytištěn první. Aby rozvržení štítků v nákresu odpovídalo formátu štítků zvolenému v návrhu sestavy, je nutno před otevřením tohoto okna zavolat funkci **View_pattern** a předat ji číslo sestavy (číslo lze zjistit funkcí **Find_object**).

Zvolit tiskárnu a orientaci, zdroj a rozměry jejího papíru lze také ve specializovaném dialogovém okně, které se otevírá *akcí* **Zvolit tiskárnu** nebo funkcí **Printer_dialog**.

Okno pro volbu tiskárny



Příklad:

Složitějším případem je situace, kdy si program musí definici zdroje dat sám sestavit podle zadaných údajů. Sestava s názvem Pujceno je navržena do tabulky obsahující výpůjčky z knihovny. Uživatel za běhu programu vybere, že chce vypsat jen výpůjčky od ledna do března. Do proměnných programu (např. `vyp_od` a `vyp_do`) typu Date se zapíší hodnoty 1.1.1998 a 31.3.1998. Část programu pak může vypadat takto:

```
strpom := "SELECT * FROM Knihy WHERE
  dat_vyp>="+Date2str(vyp_od,1)+" AND
  dat_vyp<="+Date2str(vyp_do,1)+" ORDER BY dat_vyp" ;
if Open_sql_cursor(ctisk,strpom) then Signalize
else begin
  if Print_opt(0) then Print_view("*Pujceno",ctisk,-1,-1);
  Close_cursor(ctisk);
end;
```

Neinteraktivní volba tiskárny a nastavení parametrů

Základní parametry tisku - rozsahy stránek a záznamů, výstupní zařízení - lze zadat funkcí **Set_printer**. Počet kopií a snášení kopií se nastavuje funkcí **Print_copies**. Velikost okrajů definuje funkce **Print_margins**. Pořadové číslo prvního tištěného štítku na stránce se nastavuje funkcí **Set_first_label**.

Zvolit tiskárnu, nastavit druh, zdroj a orientaci papíru a další parametry lze pomocí funkce **`Printer_select`**. U některých tiskáren však existují navíc specifické parametry, které lze nastavovat pouze interaktivně.

Velikost spodního okraje papíru

Zadání spodního okraje papíru je nutné, pokud ve vytištěné sestavě chybí spodní konec každé stránky. K tomuto efektu dochází u některých ovladačů tiskáren v důsledku špatné spolupráce mezi tiskárnou a ***Windows***. Pokud například pro laserovou tiskárnu nastavíte na ovládacím panelu ***Windows*** rozměr papíru A4, může se stát, že si ***Windows*** myslí, že mohou využít celý rozměr A4, tedy 210 krát 297 milimetrů. Ve skutečnosti však tiskárna neumí tisknout až k okrajům stránky. Zadáte-li ve **WinBase602** spodní okraj stránky, zajistíte tím, že se **WinBase602** nebude pokoušet využít plné rozměry stránky, které ji sdělil operační systém, a bude tisknout pouze tam, kam to připouští tiskárna.

Štítky

Zvláště při tisku štítků na nekonečný pás je nutné přesně nastavit okraje - většinou je nutné zadat dolní okraj roven nule.

U štítků věnujte zvýšenou pozornost také správné velikosti papíru. Chybná volba může způsobit například to, že na každé stránce bude scházet poslední řádka štítků. Pro tisk jmenovek. na nekonečný pás je rozumné používat formát papíru "Skládaný papír 8.5 x 12 palců".

Příklad:

V následujícím příkladu bude tisknuto na zadané tiskárně na formát A4 na výšku (zřejmě na standardní arch se štítky). Uživatel má možnost zvolit pouze počet kopií, směřování výstupu (preview nebo tiskárna) – to vše je zapsáno v pomocné tabulce `Parametry` a v dialogu Formát stránky pro štítky může zvolit pozici prvního tisknutého štítku na stránce:

```
if Get_fcursor(id_prehled,ctiskst,NIL) then begin
  Print_margins(0,0,0,0);         //u štítků vždy nulujte
  preview := Parametry[0].preview;
  Set_printer(0,99999,1,999,preview,'',0); //zvolit preview
  kolikrat := Parametry[0].kolikrat;
  Print_copies(kolikrat,true);             //zvolit počet kopií
  Printer_select('HP LaserJet IIP','',1,9,1,-4,1,2,1);
  if Find_object("Stitek3x7",CATEG_VIEW,objnum) then Signalize
  else begin
    View_pattern(objnum);
    if Page_setup(0) then       // nastavit pozici štítku
       Print_view("*Stitek3x7",ctisk,-1,-1);
  end;
end;
```

Výběr záznamů pro tisk

Obsahem sestavy je zpravidla buď jeden záznam nebo množina záznamů vybraná jako odpověď na dotaz. Pokud se výběr záznamů, které chcete vytisknout, nedá vyjádřit dotazem, můžete v programu postupovat takto:

1. Otevřete prázdný kurzor tj. kurzor neobsahující žádné záznamy. Použijte k tomu nesplnitelnou výběrovou podmínku, např.
2. `SELECT * FROM tab WHERE FALSE`
3. Pomocí funkce **`Add_record`** přidejte do tohoto kurzoru ty záznamy, které chcete vytisknout.
4. Přesměrujte tištěnou sestavu na tento kurzor.
5. Po vytištění sestavy kurzor uzavřete.

Ovládání menu

Menu je tradičním ovládacím prvkem aplikací pod ***Windows***. Menu navržená a uložená jako součást aplikace ve **WinBase602** lze z zobrazovat a přepínat z programu.

Programové ovládání se netýká obsahu, funkce, aktivity ani zatržení položek v menu, neboť tyto vlastnosti jsou pevně definovány v návrhu. Chcete-li tedy například změnit nezatrženou položku menu na zatrženou, pak nevoláte žádnou funkcí, ale postaráte se, aby se podmínka zatržení vyhodnotila jako splněná.

Zobrazení menu

Je-li některé menu označeno jako startovní objekt aplikace, zobrazí se automaticky při spuštění aplikace. Jinak je aplikace při svém startu bez menu. K zobrazení horizontálního menu slouží *akce* **Otevřít menu** nebo *funkce* **`Main_menu`**.

Opětovným zavoláním této akce nebo funkce lze stávající menu nahradit novým.

Odstranění menu

Akce **Odstranit menu** nebo *funkce* **`Main_menu`** zavolaná s parametrem NIL odstraní horizontální menu hlavního okna **WinBase602**. Stane-li se tak v aplikaci, v níž je toto menu startovním objektem, aplikace skončí. Jinak aplikace pokračuje po provedení této akce v činnosti bez menu. Pokud menu bylo otevřeno přímo z vývojového prostředí **WinBase602**, pak se po jeho odstranění na obrazovku vrátí standardní vývojové menu.

Překreslení menu

Akce **Překreslit menu** nebo *funkce* **`Main_menu`** zavolaná s prázdným řetězcem jako parametrem způsobí pouze překreslení hlavního menu. Tuto akci je třeba zavolat tehdy, pokud se změnily podmínky aktivity položek umístěných na nejvyšší úrovní v menu - jinak se změna aktivity neprojeví na viditelné podobě menu.

Dynamické vytvoření návrhu menu

Kromě menu, jejichž návrh je uložen v databázi jako součást aplikace, lze otevírat i menu, jejichž návrh byl vytvořen dynamicky a v databázi není.

Text návrhu menu lze přímo předat jako parametr funkci **`Main_menu`**. Tato možnost se však příliš často nepoužívá, protože struktura zdrojového textu menu není dokumentována.

Přepínání horizontálních menu

Pokud se během práce s aplikací otevře nebo aktivuje vnitřní textový editor, je menu aplikace dočasně nahrazeno menu editoru. Jakmile se však stane aktivním některý formulář nebo jakmile je okno editoru zavřeno, hlavní okno získá zpět menu nastavené naposled volanou funkcí **`Main_menu`**.

Popup menu

K zobrazení popup menu slouží *akce* **Nabídnout popup menu**. Popup menu nelze volat přímo z programu pomocí funkce. Zpravidla se volá v reakci na stisk prvého tlačítka myši na složce formuláře nebo na pozadí formuláře a definuje se v návrhu formuláře ve vlastnosti **Akce na pravé tlačítko myši**.

Interaktivní návrh objektů za běhu aplikace

Objekty, tvořící návrh aplikace, nemusejí nutně všechny pocházet z pera autora aplikace. Některé databázové aplikace jsou do té míry pružné, že poskytují svým uživatelům možnost vytvářet nové objekty nebo upravovat stávající.

Vytvořit objekt lze pomocí funkce serveru **`Insert_object`**, přepsat objekt lze pomocí příkazů SQL nebo přepsáním definice objektu v systémové tabulce objektů. Mimo to existuje možnost využít k vytváření a modifikování objektů vizuální návrháře, které jsou součástí vývojového prostředí **WinBase602**.

Přehled funkcí

Vizuální návrháře se dají volat pomocí funkcí **`Edit_view`** (formuláře a sestavy), **`Edit_table`** a **`Edit_table_ex`** (tabulky), **`Edit_query`** a **`Edit_query_ex`** (dotazy) a **`Edit_impexp`** (přenosy dat). Tyto funkce otevřou příslušný vizuální návrhář a skončí poté, co uživatel ukončí návrh.

Výchozí stav

Vytváření nového objektu může začínat buď od prázdného návrhu, anebo od výchozího návrhu připraveného aplikací. V druhém případě aplikace na uživatelovu žádost vytvoří výchozí definici objektu, uloží ji do databáze a poté na ní otevře příslušný návrhář. Tento přístup je vhodný zejména tehdy, když uživatel nemá kompletní znalost ovládání návrhá-

ře a jeho cílem je pouze mírně upravit stávající návrh, například umístění složek ve formuláři či v sestavě.

Vizuální návrháře tabulek a dotazů jsou schopny pracovat nejen nad návrhem uloženým v databázi, ale také nad návrhem v textovém bufferu.

Podpora pro odesílání zásilek elektronickou poštou

Podpora zabezpečuje interface na libovolného klienta elektronické pošty, který podporuje rozhraní ***MAPI*** nebo na klienta pošty ***Mail602***. Uživatelům **WinBase602** je umožněno z prostředí databázové aplikace vytvořit a odeslat zásilku (dopis) libovolnému uživateli. V této verzi není možné žádným způsobem přebírat jména a adresy ze seznamů poštovního klienta, tato jména a adresy musí být součástí databázové aplikace.

Před odesláním první zásilky, je třeba poštu inicializovat pomocí funkce **`InitWBMail`** nebo **`InitWBMail602`**, podle typu klienta, který je na počítači nainstalován. V okamžiku kdy je jisté, že aplikace nebude odesílat další zásilky, je možné prostředky potřebné pro zpracování pošty uvolnit pomocí funkce **`CloseWBMail`**.

Vlastní zásilka se vytvoří v několika krocích:

1. Funkce **`LetterCreate`** vytvoří zásilku, nastaví věc (subjekt) zásilky, text dopisu a parametry zásilky. Text dopisu může být předán jako obsah řetězcové proměnné nebo jako jméno souboru, který text dopisu obsahuje. Soubor musí obsahovat pouze čistý neformátovaný text. Funkce vrací v posledním (výstupním) parametru identifikátor zásilky, který se pak předává ostatním funkcím. Všechny texty musí být ve formátu Windows CP1250, má-li být zachována diakritika.
2. Seznam adresátů zásilky (včetně případných adresátů "na vědomí") se vytvoří opakovaným voláním funkce **`LetterAddAddr`**.
3. K zásilce lze (není nutno) připojit libovolný počet souborů - funkcí **`LetterAddFile`**.

Zásilka se předá poštovnímu klientu k odeslání pomocí funkce **`LetterSend`**. Pokud je na počítači nainstalován vzdálený klient pošty ***Mail602***, lze v parametrech zásilky specifikovat příznak, který zabezpečí to, že se WinBase602 pokusí ihned navázat telefonické spojení s mateřským poštovním úřadem a zásilku mu předat. Jestliže příznak není nastaven, zůstane zásilka v poště k odeslání a poštovnímu úřadu je možné ji předat později buď “ručně“ z prostředí klienta ***Mail602*** nebo programově pomocí funkce **`TakeMailToRemOffice`**.

Pokud během vytváření zásilky pominou důvody k jejímu odeslání, lze ji zrušit a uvolnit tak alokované zdroje pomocí funkce **`LetterCancel`**.

Veškeré poštovní funkce jsou přímo přístupné z vnitřního programovacího jazyka. Pro externí jazyky jsou k dispozici v knihovně WBKERNEL.DLL.

Pro případné volání funkcí z SQL použijte knihovnu WBMAIL.DLL.

```
DECLARE PROCEDURE Init(INOUT EmiPath CHAR(30),
          INOUT UserID CHAR(8), INOUT Passw CHAR(20));
EXTERNAL NAME "InitWBMail@WBMAIL.DLL";
```

Podrobnější příklad použití naleznete v elektronické nápovědě.

Přehled funkcí pro spolupráci s poštou

Většina funkcí vrací 0 v případě úspěchu a číslo chyby v případě neúspěchu. Tabulka čísel chyb je obsažena v elektronické nápovědě.

`InitWBMail` Inicializuje odesílání pošty z **WinBase602**. Zpřístupňuje libovolnou poštu, která používá rozhraní ***MAPI*** např. ***Microsoft Exchange***.

`InitWBMail602` Inicializuje odesílání pošty z **WinBase602**. Zpřístupňuje pouze poštu ***Mail602***.

`CloseWBMail` Ukončuje používání pošty a uvolňuje alokované zdroje.

`LetterCreate` Vytvoří novou zásilku.

`LetterAddAddr` Přidá adresáta do seznamu adresátů zásilky.

`LetterAddFile` Připojí k zásilce soubor.

`LetterSend` Odešle zásilku a uvolní prostředky alokované pro vytvoření zásilky. Po návratu je ID zásilky neplatné.

`TakeMailToRemOffice` Funkce spustí gateway, která naváže telefonické spojení s mateřským poštovním úřadem a předá mu zásilky k odeslání. Funkce má význam pouze v případě vzdáleného klienta pošty ***Mail602***. Kladný výsledek funkce informuje pouze o tom, že se podařilo spustit gateway a předat ji zásilky, nic však o výsledku doručení.

`LetterCancel` Funkce zruší doposud neodeslanou zásilku a uvolní prostředky, které zásilka alokovala. Po návratu z funkce je ID zásilky neplatné.

Další funkce pro vytváření aplikací

Reakce na stisk klávesy

Aplikace běžící pod ***Windows*** se zpravidla ovládají myší, pevně danými klávesami a jejich kombinacemi (mezerník, šípky) nebo tzv. horkými klávesami vyznačenými podtržením znaku v menu či v nadpisech složek, na něž klávesa působí. Mimo to je možno (ač se to všeobecně nedoporučuje) použít starší styl ovládání, při němž aplikace předepíše jednotlivým klávesám (zejména tzv. funkčním klávesám) určitý význam a funkci.

WinBase602 podporuje ve vnitřním jazyce i tento styl pomocí funkce **`Register_key`**. Tato funkce umožňuje programátorovi aplikace přiřadit klávese nebo kombinaci kláves zprávu. Zpráva pak bude zaslána programu v okamžiku stisku kláves. Program získá zprávu voláním funkce **`Get_ext_message`** a může na ní vhodně reagovat.

Časovač

Aplikace může pomocí funkce **`Set_timer`** nastavit časovač, který ve stanovené periodě posílá programu zvolenou zprávu. Program tuto zprávu přijme funkcí **Get_ext_message** a může na ní vhodně reagovat. Takto lze naprogramovat akce, které budou prováděny "na pozadí" běžící aplikace. Zprávy od časovače se nezpracovávají, když je otevřeno modální okno.

Práce se soubory

Pro interaktivní výběr souboru slouží funkce **`Select_file`**, pro výběr adresáře funkce **`Select_directory`**.

Pro manipulace se soubory z prostředí vnitřního programovacího jazyka jsou k dispozici funkce:

- **`Delete_file`** - zrušení souboru;
- **`Make_directory`** - vytvoření adresáře;
- **`Seek`** - nastavení pozice v souboru;
- **`Filelength`** - zjištění délky souboru;
- **`Reset`**, **`Rewrite`**, **`Close`**, **`Read`**, **`Write`**, **`Writeln`**, **`Eof`** - funkce pro práci s textovými soubory - význam jako v jazyce Pascal.

Pro práci s binárními soubory je nutno volat funkce patřící do ***Windows*** API.

Různé

`Alogin` - interaktivní přihlášení uživatele.

`Exec` - spuštění jiného programu v prostředí ***Windows***.

`Set_cursor` - nastavení tvaru kurzoru myši (šipka nebo hodiny).

`Edit_text` - otevření interního textového editoru na hodnotu sloupce typu **Text**.

Další funkce, týkající se například importů a exportů dat, práce s uživateli, replikací, koloběhu dokumentů a pod. jsou popsány v samostatných kapitolách.