Programování ve vnějších jazycích

Tato kapitola popisuje, jak se dají databázové aplikace využívající WinBase602 **vytvářet ve vnějších programovacích jazycích C, C++, Pascal (Delphi) a Basic. Je doplňkem kapitol o programování komunikace s databází a uživatelského rozhraní** WinBase602**.**

Na programování v externích jazycích se vztahuje řada pravidel popsaných v kapitole o programování komunikace s databázovým serverem. V této kapitole budeme proto uvádět pouze odlišnosti.

Knihovny šířené s **WinBase602 SDK** jsou určeny pro vývojová prostředí ***Borland* C++**, ***Microsoft Visual C/C++*** (včetně MFC)**, *Borland Pascal*, *Delphi*** a ***Visual Basic***. Univerzální nástroje, jako ActiveX nebo ODBC drivery, jsou využitelné v nepřeberném množství dalších prostředí.

Přehled variant aplikací spolupracujících s WinBase602

Rozhraní mezi aplikací, napsanou ve vnějším jazyce, a **WinBase602,** lze vytvořit na různých úrovních. Komunikace aplikace s WinBase602 může probíhat:

1. prostřednictvím *specializovaných* API funkcí implementovaných ve **WinBase602** a volaných z aplikace;
2. prostřednictvím *specializovaných* tříd a komponent zapouzdřujících API **WinBase602**;
3. pomocí *univerzálních* rozhraní OLE, ODBC nebo DAO, případně pomocí *univerzální* komponenty ActiveX resp. OCX.

Pro jazyky C/C++ a Pascal **WinBase602** podporuje všechny úrovně, pro ***Visual Basic*** pouze první a třetí. Podívejme se detailně na vlastnosti jednotlivých rozhraní.

Komunikace prostřednictvím API funkcí a zpráv

WinBase602 nabízí řadu funkcí, jejichž voláním se aplikace může například přihlásit na server, položit SQL dotaz, číst data z databáze nebo je zapisovat, otevřít na obrazovce formulář nebo vytisknout sestavu.

Ke zpřístupnění těchto funkcí (a souvisejících definic typů a konstant) stačí do aplikace v jazyce C/C++ začlenit definiční soubory (*header files*) WBKERNEL.H a WBPREZEN.H, v jazyce Pascal použít unit WINBASE.PAS, v jazyce ***Visual Basic*** soubory WBKERNEL.TXT a WBPREZEN.TXT.

Při práci na této úrovni lze maximálně efektivně využívat schopností databázového serveru. Z **WinBase602** lze použít buď pouze služby SQL serveru anebo také objekty tvořící uživatelské rozhraní aplikace (formuláře, menu, sestavy). Cesta požadavků mezi aplikací a serverem je nejpřímější.

Komunikace prostřednictvím tříd a komponent zapouzdřujících API WinBase602

Nad vrstvou API funkcí byla vytvořena množina tříd (pro MFC) a komponent (pro ***Delphi***), jejichž hlavním účelem je *zjednodušit* vytváření aplikace komunikující s **WinBase602**. Tyto třídy a komponenty neobsahují nic, co by se nedalo vytvořit pomocí API funkcí, zapouzdřují však často používaný kód, snižují pracnost programátorské práce a omezují riziko vzniku chyb.

Tyto třídy a komponenty jsou dodávány ve zdrojovém tvaru. Kromě jejich metod a vlastností autor aplikace může libovolně využívat také API funkcí a zpravidla tak i činí.

Komponenty spadající do této kategorie jsou vyvíjeny více uživateli **WinBase602** a jsou k dispozici na Internetu.

Komunikace pomocí univerzálních rozhraní a komponent

Při využití univerzálních rozhraní aplikace nevolá API funkce **WinBase602**, nýbrž buď funkce těchto rozhraní (ODBC, DAO), nebo pomocí univerzálních nástrojů metody specifické pro **WinBase602** (ActiveX, OCX).

Uživatelské rozhraní se při využití ODBC a DAO vytvoří kompletně pomocí prostředků vizuálního programování nabízených vývojovým prostředím aplikace. Při použití ActiveX (OCX) se uplatní formuláře **WinBase602**, ale v zapouzdřené podobě.

Struktura dokumentace rozhraní

Tato kapitola se věnuje následujícím tématům:

- Specifika volání funkcí komunikujících s obsahem databáze z prostředí vnějších jazyků - informace potřebná pro programátory využívající rozhraní 1 a 2.
- Začátek a konec komunikace aplikace s databází - informace potřebná pro programátory využívající rozhraní 1, částečně 2.
- Specifika aplikací v C/C++ s rozhraním 1.
- Specifika aplikací v Pascalu s rozhraním 1.
- Komponenty pro Delphi (rozhraní 2).
- Třídy pro MFC (rozhraní 2).
- Interface pro Visual Basic (rozhraní 1).

Ve zvláštních kapitole je popsáno OCX (ActiveX) pro formulář **WinBase602** (rozhraní 3) využití rozhraní ODBC.

Knihovny WinBase602

Procedury a funkce API WinBase602**, které může volat aplikační program, jsou obsaženy v dynamicky linkovatelných knihovnách WBKERNEL.DLL, WBPREZEN.DLL a WBVIEWED.DLL.**

Tyto knihovny dále využívají knihovnu WBED.DLL.

Pokud by váš program potřeboval využívat služby instalační knihovny, nalezne je v souboru INSTSERV.DLL.

Všechny knihovní funkce vyžadují způsob předávání parametrů označovaný **`__stdcall`** v ***C/C++***, resp. **`stdcall`** v ***Pascalu***.

ODBC driver pro přístup *do* **WinBase602** používá knihovny WB6DRV.DLL a WB6STP.DLL. ODBC driver se instaluje v rámci **WinBase602** automaticky, lze jej však nainstalovat i samostatně volbou v instalačním programu.

Rozhraní OLE a implementace ActiveX (OCX) pracují s knihovou WBVIEW.OCX. OLE server, umožňující vložení formuláře nebo sestavy do cizího dokumentu, je implementován v souboru WBOLE602.EXE.

Knihovny byly vytvořeny pomocí překladače ***Microsoft Visual C/C++***. Nic nebrání jejich použití z jiných jazykových prostředí, např. ***Pascalu*** nebo ***C*** od firmy **Borland**.

Začátek a konec práce klienta v externím jazyce

Činnost aplikace, která chce být klientem **WinBase602**, se musí podřídit níže popsaným pravidlům. Pro ***MFC*** jsou k dispozici třídy a pro ***Delphi*** komponenty, odvádějící inicializační a deinicializační práci (nebo její část) požadovaným způsobem. Jich lze využít pro zjednodušení práce.

Klientská aplikace musí nejprve vytvořit své hlavní okno vhodného druhu (viz dále). Poté mohou následovat další inicializační kroky.

Kontextová proměnná klienta

Každý klient, který se chce připojit na databázový server **WinBase602,** musí mít speciální proměnnou pro průběžné uložení informací o kontextu komunikace. Tuto proměnnou označujeme jménem **`cd`** a je typu **`cd_t`**. Kontextová proměnná se předává jako parametr do řady podprogramů. V ***Pascalu*** se předává odkazem, v jazyce ***C/C++*** se předává ukazatel **`cdp`** typu **`cdp_t`**, který na ní ukazuje.

Každý klient musí ve svém programu:

1. Vytvořit proměnnou **cd** typu **cd_t** buď tak, že ji staticky deklaruje, nebo tak, že ji dynamicky alokuje, nejlépe voláním funkce **cdp_alloc**.
2. Inicializovat tuto proměnnou voláním funkce **cdp_init**. Tuto akci je nezbytně nutné provést před voláním ostatních funkcí využívajících kontextovou proměnnou.
3. Předávat tuto proměnnou funkci **interf_init** a případně i dalším funkcím (například funkcím serveru začínajícím prefixem **cd_**).
4. Byla-li proměnná **cd** alokována pomocí **cdp_alloc**, dealokovat tuto proměnnou po odpojení se od serveru voláním funkce **cdp_free**.

Alokujete-li proměnnou **cd** ve vlastní správě paměti, můžete zjistit její potřebnou velikost voláním funkce **cdp_size**.

Funkce s prefixem cd_

Klientský program má k dispozici dvě sady funkcí pro komunikaci se serverem. Funkce, začínající prefixem **cd_** mají oproti funkcím z druhé sady jeden parametr navíc - prvním parametrem je kontextová proměnná **cd** resp. ukazatel na ni.

Klient se může současně připojit na více serverů nebo si vytvořit vícenásobné spojení na stejný server. V takovém případě musí mít více kontextových proměnných a musí tuto proměnnou předávat jako první parametr do funkcí serveru, musí tedy používat funkce s prefixem **cd_**. Více paralelních spojení klienta na server však znemožňuje používat nástroje uživatelského rozhraní zabudované do **WinBase602**. Jde o náročnou techniku a doporučujeme se jí spíše vyhýbat.

Funkce s prefixem **cd_** je třeba používat také tehdy, pokud spojení na server vytvoří jedno vlákno a používá je jiné.

Za normálních provozních podmínek je použití obou sad funkcí ekvivalentní, funkce s prefixem **cd_** jsou nepatrně efektivnější.

Slupka klientské aplikace

Volba serveru nebo spuštění serveru

Každý klient musí zvolit server, s nímž bude komunikovat. Vybrat si lze pouze z takových serverů, které jsou zaregistrovány na počítači, na němž běží klient, nebo které běží ze stejném segmentu sítě (pro TCP/IP pouze je-li u klienta povolen broadcasting).

Volba serveru se provede zavoláním funkce **link_kernel** se jménem serveru předaným jako parametr. Tato funkce zjistí, zda požadovaný server běží lokálně nebo v síti. Pokud server neběží a je známa cesta k jeho databázovému souboru, pak je server lokálně spuštěn.

Pokud server běží lokálně, preferuje se přímé připojení klienta bez podpory sítě. Chcete-li se připojit po síti i na lokálně běžící server (např. kvůli testování síťového chování), musíte před jménem serveru v parametru funkce **link_kernel** uvést hvězdičku.

Zprávu o případné chybě při provádění funkcí **link_kernel** lze zobrazit v dialogovém okně pomocí procedury **Kernel_error_box**.

Funkce **unlink_kernel**, která se používala ve starších verzích **WinBase602** pro ukončení serveru, nyní nedělá nic a nemusí se volat.

Připojení na server a odpojení od serveru

Připojení

Připojení se klienta na server probíhá až po provedení procedury **cdp_init** a úspěšném provedení funkce **link_kernel**. Klient zavolá funkcí **interf_init** a tím vytvoří spojení.

Zprávu o případné chybě při provádění funkce **interf_init** lze zobrazit v dialogovém okně pomocí procedury **Kernel_error_box**.

Odpojení

Klient se odpojí od serveru zavoláním funkce **(cd_)interf_close**. Funkce **(cd_)interf_close** musí být zavolána před ukončením práce klienta a před případnou dealokací proměnné **cd**.

Přihlášení a odhlášení

Klient může se serverem komunikovat buď anonymně nebo pod určitým uživatelským jménem. V databázi, kterou používá více uživatelů jsou zpravidla anonymní komunikaci přidělena pouze minimální (nebo žádná) práva.

Přihlášení

Klient se *přihlašuje* na server pod určitým uživatelským jménem pomocí funkce **(cd_)Login**. Parametry této funkce jsou jméno uživatele a heslo. Server prověří, zda uživatele má v seznamu a zda souhlasí heslo. Pokud je vše v pořádku, je přihlášení se klienta na server provedeno.

(cd_)Login lze zavolat až po úspěšném provedení funkce **interf_init**. Interaktivní variantou funkce **(cd_)Login** je funkce prezentační vrstvy **(cd_)Alogin**.

Kdo jsem ?

Po dobu, po kterou je klient přihlášen na server, může své uživatelské jméno zpětně zjistit pomocí funkce **(cd_)Who_am_I**.

Odhlášení

Opakem přihlášení se na server je *odhlášení*, které provede funkce **(cd_)Logout**. Tím se odhlásí z komunikace se serverem pod dříve zadaným jménem a nadále může komunikovat pouze jako anonymní uživatel.

Aplikační program může uživateli umožnit zadat **(cd_)Logout**, pokud uživatel odchází od počítače a nechce, aby někdo zneužil jeho práv. Později se může opět přihlásit funkcí **(cd_)Login**.

Funkce **(cd_)Logout** musí být volána před funkcí **(cd_)interf_close**.

Volba aplikace

V programech vytvořených v externích jazycích je nutno otevřít aplikaci a tím specifikovat, se kterou aplikací se bude dále pracovat. V této aplikaci se pak budou hledat objekty zadané jako parametr funkci `(cd_)Find_object` **nebo ty, na něž odkazují příkazy SQL.**

Aplikaci otevřete po přihlášení uživatele voláním funkce **`(cd_)Set_application`**.

Z externího jazyka lze vytvořit novou aplikaci pomocí funkce **`(cd_)Insert_object`** s parametrem `CATEG_APPL`. Před vytvářením nových objektů v této aplikaci je nutno ji otevřít.

Komunikace s vnitřním programovacím prostředím

Aplikační program napsaný v externím jazyce může *nastavit jako projekt* program vytvořený ve vnitřním jazyce a umístěný v otevřené aplikaci. Tím získá možnost pracovat s proměnnými tohoto projektu, provádět příkazy v kontextu projektu, otevírat objekty (formuláře, menu, dotazy, sestavy) využívající deklarace z projektu a používat v SQL příkazech hostitelské proměnné.

Nastavení projektu se provede voláním funkce **`Open_project`**. Ke konkrétní proměnné z projektu lze přistoupit přes její adresu získanou od funkce **`Get_var_address`**. Provést zadanou posloupnost příkazů vnitřního programovacího jazyka v kontextu projektu umožňuje funkce **`Exec_statements`**.

Další informace o projektech naleznete v kapitole věnované vnitřnímu programovacímu prostředí.

Přehled pořadí akcí

Výše popsané akce tvoří slupku aplikace s touto strukturou:

1. alokace kontextové proměnné **`cd`**, není-li alokována staticky;
2. volání **`cdp_init`**;
3. vytvoření hlavního okna (nebylo-li otevřeno dříve) a inicializace okenního prostředí;
4. volání **`link_kernel`**;
5. volání **`interf_init`**;
6. volání **`(cd_)Login`**, pracuje-li se neanonymně;
7. volání **`(cd_)Set_application`**;
8. volání **`Open_project`**, je-li potřeba;

vlastní aplikace

n-3. **`(cd_)Logout;`**
n-2. **`(cd_)interf_close;`**

n-1. dealokace **cd**, není-li alokována staticky;
n. zavření hlavního okna aplikace.

Typy pro komunikaci s databázovým serverem

WinBase602 **definuje řadu typů a používá je zejména v popisech parametrů a výsledků API funkcí.**

Jsou definovány tyto celočíselné typy:

Jméno typu	rozsah od	do
uns8	0	255
uns16	0	65535
uns32	0	4294967295
sig8	-127	127
sig16	-32767	32767
sig32	-2147483639	2147483639

V jazyce Borland Pascal typ **uns32** má menší rozsah, na činnost **WinBase602** to však nemá vliv.

Dále pro konkrétní účely byly zavedeny tyto celočíselné typy:

Jméno typu	použití pro	totožný s
Ttablenum	číslo tabulky	sig16
Tobjnum	číslo objektu	sig16
Trecnum	číslo záznamu (řádku v tabulce)	uns32
Tcursnum	číslo otevřeného kurzoru	sig16
Tcurstab	číslo tabulky nebo otevřeného kurzoru	sig16
tattrib	číslo sloupce v tabulce nebo kurzoru	uns8
tcateg	číslo kategorie	uns8

V pravém sloupci je typ, který je totožný s typem v levém sloupci.

Typ **tobjname** je typem řetězce znaků obsahujícího jméno objektu. Má délku 32 bajtů pro 31 znaků jména a ukončující nulu. Maximální délka jména objektu je označena konstantou OBJ_NAME_LEN.

Typ **WBUUID** je typem univerzální identifikace objektu - uživatele, skupiny uživatelů, replikačního serveru nebo klíče. Má délku 12 bajtů.

Typ **cd_t** je typem *kontextové proměnné* aplikačního programu, typ **cdp_t** je typem *ukazatele* na kontextovou proměnnou.

Čtení a zápis dat

Pro čtení a zápis hodnot sloupců lze použít některé z těchto funkcí:
(cd_)Read a **(cd_)Write** - univerzální funkce pro práci s hodnotou sloupce,
(cd_)Read_ind, **(cd_)Write_ind** - práce se sloupcem pevné velikosti,
(cd_)Read_var, **(cd_)Write_var** - práce se sloupce proměnné velikosti,
(cd_)Read_len, **(cd_)Write_len** - práce s délkou hodnoty sloupce proměnné velikosti,
(cd_)Read_ind_cnt, **(cd_)Write_ind_cnt** - práce s počtem složek multiatributu.
(cd_)Read_record, **(cd_)Write_record** - práce s celým záznamem najednou.

Funkcím se předávají tyto parametry:

1. číslo tabulky nebo otevřeného kurzoru, s nimiž se pracuje;
2. číslo záznamu v tabulce nebo kurzoru;
3. číslo sloupce (kromě funkcí pracujících s celým záznamem);
4. pořadové číslo hodnoty v multiatributu; pokud funkce pracuje se sloupcem, který není multiatributem, pak tento parametr má hodnotu NOINDEX (pouze některé funkce).

Při zápisu hodnoty se dále udává její délka. Při práci s hodnotou proměnné velikosti se udává začátek a délka úseku, s nimž se pracuje.

Stav záznamu a hodnoty sloupce DELETED

Záznamy v každé tabulce mají sloupec DELETED s číslem 0 (DEL_ATR_NUM). Typ tohoto sloupce považujeme v externích jazycích za **uns8**. Nabývá tří hodnot:

- NOT_DELETED záznam existuje, není zrušen
- DELETED záznam je zrušený (lze jej obnovit)
- RECORD_EMPTY záznam je uvolněný (nelze jej obnovit)

Číslo tabulky a číslo kurzoru

Číslo tabulky lze získat pomocí funkce **(cd_)Find_object**, např.:

```
cd_Find_object(cdp, "MOJETABULKA", CATEG_TABLE, &tablenum);
```

Čísla *systémových* tabulek jsou v externích jazycích označena konstantami TAB_TABLENUM, OBJ_TABLENUM, USR_TABLENUM, SRV_TABLENUM, REPL_TABLENUM a KEY_TABLENUM.

Číslo kurzoru vznikne při otevření kurzoru. Pevný kurzor vytvořený podle dotazu v databázi se otevírá příkazem **`Open_cursor`**, proměnný kurzor z dynamicky sestrojeného dotazu se otevírá funkcí **`Open_cursor_direct`**. Otevřený kurzor je nutno uzavřít funkcí **`Close_cursor`**.

Čísla sloupců tabulek a kurzorů

***Jména* sloupců databázových tabulek a kurzorů se používají pouze v SQL a ve vnitřním programovacím jazyce. Ve funkcích pro externí jazyky se sloupce označují svými *pořadovými čísly*.**

Číslo sloupce se předává jako parametr například funkcím **`(cd_)Read`** a **`(cd_)Write`**, které čtou resp. zapisují hodnotu sloupce.

Sloupce jsou očíslovány souvisle od nuly. Číslo nula má služební sloupec, který se v tabulkách jmenuje DELETED a v kurzorech __DELETED a nese informaci o tom, zda záznam je zrušený. Za nim následují případné další služební sloupce týkající se replikací, koloběhu dokumentů a práv k jednotlivým záznamům (jejich jména začínají _W5_). Za nimi jsou normální sloupce v tom pořadí, v němž jsou uvedeny v definici tabulky nebo v klauzuli SELECT v definici kurzoru.

Definiční soubory s čísly sloupců tabulek a dotazů z aplikace

WinBase602 poskytuje speciální nástroj, jak kompilátorům jednoduše sdělit čísla všech sloupců ve všech tabulkách a dotazech v aplikaci. Tímto nástrojem je vytvoření DEFINIČNÍCH SOUBORŮ, které se poté využijí při překladu aplikačního programu.

V hlavním menu vývojového prostředí **WinBase602** je v menu ***Nástroje*** příkaz **Připravit include**, který dovoluje vytvořit definiční soubor pro jazyk ***C*** nebo ***Pascal***.

Pro jazyk ***C*** bude obsahem souboru řada direktiv **`#define`**, které definují identifikátor (makro) označující sloupec pomocí jeho čísla. Pro ***Pascal*** budou identifikátory definovány pomocí deklarace konstant.

Identifikátory pro sloupce tabulek

Identifikátory označující sloupce tabulek budou mít tvar:

```
A_tabulka_sloupec
```

kde *tabulka* je jméno tabulky a *sloupec* je jméno sloupce tabulky.

Identifikátory pro sloupce kurzorů

Identifikátory označující sloupce kurzorů budou jeden z tvarů:

```
B_kurzor_sloupec   nebo
B_kurzor_tabulka_sloupec nebo
B_kurzor_alias_sloupec
```

kde *kurzor* je jméno kurzoru, *sloupec* je jméno sloupce, *tabulka* je jméno tabulky, z níž sloupec pochází a *alias* je případné označení tabulky v rámci kurzoru (*tabulka* nebo *alias* se použije, pokud kurzor obsahuje více stejně pojmenovaných sloupců).

Pro jazyk ***C*** má definiční soubor standardně příponu SIC, pro ***Pascal*** příponu SIP. Do aplikačního programu jej pomocí vtahování souborů takto:

```
#include "appl.sic"   resp. {$I appl.sip}
```

Jména utvořená výše popsaným způsobem a definovaná v definičním souboru lze využít všude tam, kde je parametrem funkce číslo sloupce.

Zjišťování čísel sloupců při běhu programu

Ke zjištění čísla sloupce lze také využít alternativní mechanismus, který nevyžaduje použití žádného definičního souboru.

Voláním funkce **`(cd_)Attribute_info`** lze při běhu programu převést řetězec znaků obsahující jméno sloupce na jeho číslo a zároveň se dozvědět další údaje o sloupci. Voláním funkce **`(cd_)Enum_attributes`** získáte přehled všech sloupců. Obě funkce se dají použít na tabulku nebo na otevřený kurzor.

Sloupce kurzorů vznikajících při běhu programu

Pro kurzory, jejichž dotaz není uložen v databázi, je třeba pro nalezení čísel sloupců použít pravidlo o číslování zleva doprava v pořadí, v němž jsou sloupce uvedeny v klauzuli SELECT. První sloupec uvedený za SELECT má číslo 1, další 2 atd.

Příklad:

Nechť tabulka T1 obsahuje sloupce A a B, tabulka T2 sloupce C a D. Otevřete kurzor CU v jazyce C voláním funkce:

```
Open_cursor_direct(
  "select A,D,B+C from T1,T2 where A<C order by D", &CU);
```

Pak přečíst hodnotu sloupce D ze záznamu REC v kurzoru CU lze funkcí:

```
Read(CU, REC, 2, NULL, &buffer);
```

zatímco hodnotu součtu B+C přečtete

```
Read(CU, REC, 3, NULL, &buffer);
```

Délka dat a datové buffery

Funkce pro zápis do databáze mají za poslední parametr údaj *datasize*. Hodnotou tohoto parametru je délka hodnoty zapisované do databáze v bajtech. Pokud píšete do sloupce pevné délky, pak tato hodnota musí být stejná jako velikost hodnoty tohoto sloupce, tedy např. pro typ **Char** je to 1, pro **Short** 2, pro **Date, Time, Timestamp,** ukazatele a **Integer** 4, pro **Money** 6, pro **Real** 8, pro typy **String, CSString** a **CSIString** je to délka řetězce uvedená v definici tabulky (nikoli tedy skutečná délka zapisovaného řetězce).

Při čtení hodnoty je funkci nutno předat adresu dostatečně velkého bufferu.

Délky hodnoty sloupce proměnné velikosti je 4-bajtový celočíselný údaj, při jejím zápisu má tedy parametr *datasize* hodnotu 4. Počet složek multiatributu je 2-bajtový celočíselný údaj, při jeho zápisu má tedy parametr *datasize* hodnotu 2.

Parametr Access v univerzálních funkcích Read a Write

Funkce **`(cd_)Read`** a **`(cd_)Write`** jsou schopny nahradit všechny ostatní funkce pro čtení a zápis dat, jejich volání je však poněkud složitější. Do rozhraní serveru jsou zahrnuty hlavně kvůli kompatibilitě se staršími verzemi. Vzhledem k tomu není nutno studovat zbytek této sekce.

Obě funkce mají parametr *ACCESS*, který popisuje cestu, jak se dostat k požadovaným datům v databázi.

Pokud pracujete se sloupcem, který není multiatributem ani nemá proměnnou délku, pak uvádějte hodnotu tohoto parametru rovnou NULL v jazyce **C**, NULL_ACCESS v **Pascalu**.

Práce s multiatributy, se sloupci proměnné délky a případně i s ukazateli vyžaduje zadání dalších parametrů. V případě multiatributů je to *index*, tedy pořadové číslo složky multiatributu, s níž chcete pracovat. V případě sloupců proměnné délky se zadává *interval*, tedy úsek z hodnoty.

Tyto údaje poskytuje parametr ***ACCESS*.** Mimo to umožňuje:

- přecházet od záznamu obsahujícího ukazatel na záznam, kam tento ukazatel ukazuje;
- zjistit počet všech hodnot multiatributu nebo délku hodnoty sloupce proměnné délky;
- zadat počet hodnot multiatributu nebo délku hodnoty sloupce proměnné délky.

Poslední operace má efekt pouze při zmenšení počtu složek multiatributu nebo délky. Zvětšení totiž nastává jedině automaticky při zápisu.

Při zadávání kteréhokoli z výše uvedených údajů parametr ***ACCESS*** je ukazatel na pole záznamů typu *modifrec*, které popisuje cestu vedoucí k hledaným hodnotám (v **Pascalu** je pole záznamů předáváno referencí, nikoli přes ukazatel). V obou případech hodnota

parametru *ACCESS* musí přísně korespondovat s typem a dalšími charakteristikami příslušného sloupce.

Hodnota NULL parametru *ACCESS* se použije právě tehdy, když příslušný sloupec je pevné délky a není to multiatribut. Jinak parametr *ACCESS* popisuje přístup k hodnotě a jednotlivé složky pole po řadě odpovídají jednotlivým krokům na cestě ke zvoleným datům. Cesta začíná u sloupce zadaného jako parametr funkce.

Lze volit tyto kroky:

1. Pokud cesta dospěla k multiatributu, pak lze uvést index složky, s níž se má pracovat. Použije se k tomu záznam se složkou *modtype* rovnou MODIND a s vhodnou hodnotou složky *index*.
2. Pokud cesta dospěla ke sloupci proměnné délky (ten může být i již vybranou složkou multiatributu), pak lze uvést interval hodnot, s nimiž se má pracovat. Použije se k tomu záznam se složkou *modtype* rovnou MODINT a s vhodnými hodnotami složek *start* a *size*, určujícími začátek a velikost úseku.
3. Pokud cesta dospěla k multiatributu nebo sloupci proměnné délky, pak lze uvést, že chcete pracovat nikoli s jejich obsahem, nýbrž s počtem složek multiatributu resp. délkou hodnoty sloupce . K tomu použijte záznam se složkou *modtype* rovnou MODLEN. Počet složek multiatributu je číslo typu **uns16**, délka hodnoty sloupce je číslo typu **uns32**.
4. Pokud cesta dospěla k ukazateli, který není multiatributem, pak můžete přejít k záznamu, na nějž ukazatel odkazuje. K tomu použijte záznam se složkou *modtype* rovnou MODPTR a s vhodnou hodnotou složky *attr*, jež určuje, ke kterému sloupci v cílovém záznamu přejdete. Pokud ukazatel nikam neukazuje, nebo je cílový záznam zrušený, dojde k chybě.
5. Pokud cesta dospěla k multiukazateli, můžete přejít na některý ze záznamů, na něž ukazuje. K tomu použijte záznam se složkou *modtype* rovnou MODINDPTR a s vhodnými hodnotami ve složkách *index* a *attr*. *Index* určuje, která složka multiukazatele se použije a *attr* na který sloupec cílového záznamu přejdete.
6. Posledním platným záznamem v poli musí být záznam se složkou *modtype* rovnou MODSTOP.

```
typedef struct
{ uns8 modtype;
    union umoddef
    { struct smodstop { }                              modstop;
      struct smodlen  { }                              modlen;
      struct smodind  { uns16 index; }                 modind;
      struct smodint  { uns32 start;  uns16 size; }    modint;
      struct smodp    { uns8  attr;  }                 modptr;
      struct smodindp { uns16 index;  uns8 attr; }     modindptr;
    } moddef;
} modifrec;
```

```
type modifrec = record
        modtype : uns8;
        case integer of
          MODSTOP   : ( );
          MODLEN    : ( );
          MODIND    : (index  : uns16);
          MODINT    : (start  : uns32;  size : uns16);
          MODPTR    : (attr   : uns8);
          MODINDPTR : (index2 : uns16;  attr2 : uns8)
        end;
```

Hodnotou složky *modtype* v každém záznamu musí být jedna z těchto konstant:

konstanta	Umožňuje
MODSTOP	tato hodnota je vždy v posledním záznamu v poli
MODLEN	Pracovat s velikostí multiatributu nebo záznamu proměnné délky
MODIND	Pracovat se složkou multiatributu zadanou indexem
MODINT	Pracovat s úsekem hodnoty sloupce proměnné délky zadaným začátkem a délkou
MODPTR	přejít do záznamu, na nějž odkazuje ukazatel, na zadaný sloupec
MODINDPTR	přejít do záznamu, na nějž odkazuje ukazatel se zadaným indexem na zadaný sloupec

Pozor:

Čtete-li funkcí **`(cd_)Read`** obsah sloupce proměnné velikosti, pak v poli, kam obsah směřuje, bude v prvních 2 bajtech délka přečteného úseku a za ní přečtená data.

Speciální způsoby komunikace se serverem

Souběžná práce klienta a serveru

Obvyklý model spolupráce klienta a serveru předpokládá, že klient vždy čeká, až server splní jeho požadavek.

Existují případy, kdy v zájmu efektivnějšího fungování klienta je žádoucí připustit jeho souběžnou práci se serverem. Jde o to, aby po odeslání požadavku klient mohl dále pracovat a nějakým způsobem zjistit, kdy dojde odpověď.

Příklad:

Příkladem může být program, který umožňuje editovat určitou posloupnost záznamů. V době, kdy uživatel edituje určitý záznam, může program zaslat serveru novou hodnotu

předchozího editovaného záznamu a přečíst hodnotu záznamu, o němž předpokládá, že ho uživatel bude editovat jako další.

Aktivace režimu souběžné práce

Souběžnou práci umožňuje funkce **(cd_)concurrent**. Zavoláte-li ji s parametrem TRUE, je zahájen režim souběžné práce. V tomto režimu zavolání funkce databázového jádra způsobí vyslání požadavku do serveru, ale nečeká se na výsledek. Po návratu z volání nemusí být tedy ve výstupních parametrech žádná smysluplná hodnota. Ta se do nich zapíše teprve poté, až přijde od serveru odpověď.

Pozor !

Uvědomte si, že důsledkem toho mohou být zcela asynchronní změny obsahu proměnných, na něž byla spuštěna operace čtení!

Odpověď na požadavek

Klient může zjistit, zda na jeho poslední požadavek již přišla odpověď, když zavolá funkci **(cd_)answered**. Její hodnota FALSE signalizuje, že odpověď dosud nedošla.

Pokud klient vyšle další požadavek před obdržením odpovědi na předchozí požadavek, je tento nový požadavek pozdržen do příchodu odpovědi. Bezprostředně poté je nový požadavek komunikační knihovnou odeslán. Pro klienta pak neexistuje způsob, jak zjistit úspěch či neúspěch provedení prvního z těchto požadavků. Proto takový postup nedoporučujeme.

Funkce **(cd_)Break**, která ruší právě prováděný požadavek, pracuje asynchronně bez ohledu na to, zda je povolena souběžná práce.

Balíky požadavků

Za normálních podmínek se po zavolání příslušné funkce požadavek klienta předává serveru. Požadavek musí pak urazit cestu od klienta k serveru a odpověď cestu nazpět.

Při provozu v síti vyžaduje každý takový požadavek zaslání zprávy z jednoho počítače na druhý a pak zpátky. To trvá určitou dobu a v některých případech může být žádoucí tuto dobu zkracovat. Proto byla zavedena možnost sdružovat požadavky do balíků.

Pokud je sdružování aktivní, pak každý požadavek klienta se zaznamená, ale jeho odeslání se odloží. Nastřádané požadavky se odešlou najednou, až na explicitní žádost klienta.

Sdružování požadavků začíná zavoláním procedury **(cd_)start_package** a trvá až do té doby, než je zavolána procedura **(cd_)send_package**. Maximální počet požadavků v balíku je MAX_PACKAGED_REQS (viz definiční soubory).

Je-li funkce generující požadavek na jádro zavolána uvnitř balíku, pak po jejím provedení jsou hodnoty jejích výstupních parametrů nedefinovány. Parametry nabudou hodnot až po provedení procedury **(cd_)send_package**. Je-li navíc zapnuto souběžné zpracování, je třeba dokončení požadavku **(cd_)send_package** zjistit voláním funkce **(cd_)answered**.

Při použití balíků se komplikuje sledování chyb a varování. Platí, že zpracování balíku se ukončuje při výskytu první chyby. Číslo této chyby pak vrací funkce **`(cd_)Sz_error`**. Funkce **`(cd_)Sz_warning`** vrací sjednocení všech varování, která se při zpracování balíku vyskytla.

Uživatelské rozhraní aplikace nad WinBase602

Místem, kde se nejtěsněji setkávají aplikace a **WinBase602**, je správa hlavního okna aplikace a jeho suboken.

Klientská aplikace musí v každém případě vytvořit hlavní okno aplikace (nejčastěji okno typu MDI-frame). Vytvoření okna je někdy skryto do volání knihovních funkcí aplikačního prostředí a není pod plnou kontrolou aplikace. Aplikace zpracovává zprávy docházející hlavnímu oknu a reaguje na ně.

Předávání zpráv z hlavního okna

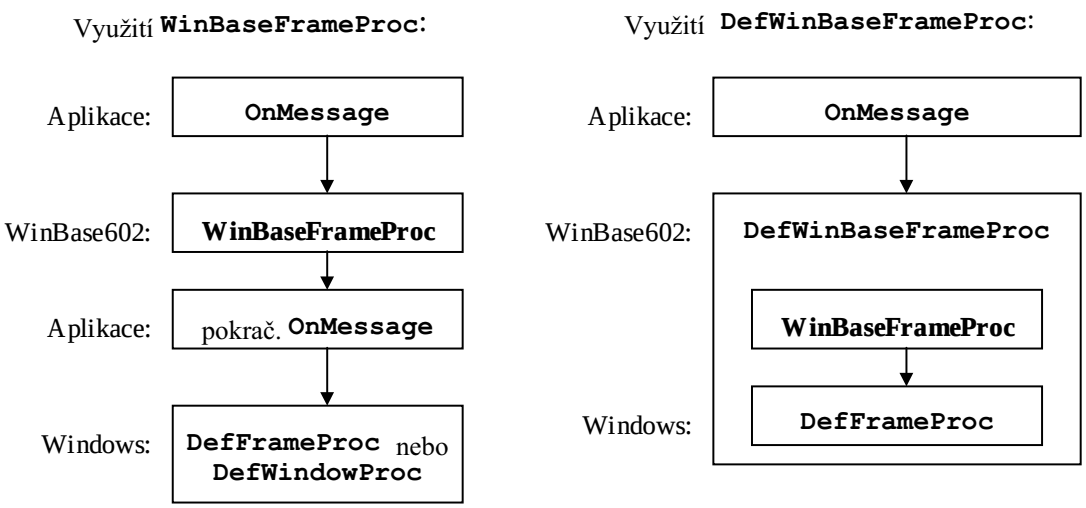
Nezpracované zprávy a zprávy, které nemají čistě soukromý charakter, musí okenní procedura hlavního okna aplikace předávat k dalšímu zpracování. ***Windows*** API nabízí pro tento účel funkce **`DefFrameProc`**.(pro hlavní okno MDI) a **`DefWindowProc`** (pro ostatní případy). Různá prostředí pro vývoj aplikací (MFC, Delphi) mají své vlastní funkce, které doplňují a nahrazují tuto univerzální funkci. Také **WinBase602** potřebuje sledovat zprávy docházející hlavnímu oknu a zpracovávat některé z nich.

Pro zpracování zpráv ve **WinBase602** je nutno volat jednu ze dvou funkcí: **`WinBaseFrameProc`** nebo **`DefWinBaseFrameProc`**.

Funkce **`WinBaseFrameProc`** provede interní zpracování předané zprávy a vrátí nulu, pokud zprávu kompletně zpracovala, případně nenulovou hodnotu, pokud zprávu je třeba předat dál k dalšímu zpracování.

Funkce **`DefWinBaseFrameProc`** nejprve zavolá funkci **`WinBaseFrameProc`**, a pokud ta vrátí 0, pak skončí a vrátí nulu také. Jinak zavolá funkci **`DefFrameProc`** a vrátí její výsledek. Tento postup se hodí pouze pro MDI.

V typické MDI aplikaci se z okenní procedury volá funkce **`DefWinBaseFrameProc`**. Funkci **`WinBaseFrameProc`** a následně **`DefFrameProc`** voláme místo ní tehdy, pokud aplikace není MDI nebo pokud pokud mezi volání těchto funkcí je třeba zařadit ještě nějaké další zpracování.



Zprávy zpracovávané ve `WinBaseFrameProc`

Funkce **`WinBaseFrameProc`** zpracovává tyto systémové zprávy: `WM_ACTIVATEAPP`, `WM_CLOSE`, `WM_DESTROY`, `WM_COMMAND`, `WM_CREATE`, `WM_INITMENU`, `WM_INITMENUPOPUP`, `WM_MENUSELECT`, `WM_PALETTECHANGED`, `WM_WININICHANGE`.

Dále zpracovává soukromé zprávy nutné pro implementaci importu a exportu dat v interním formátu.

Vytvoření oken MDI aplikace a inicializace uživatelského rozhraní

Uvnitř hlavního okna typu MDI-frame bude umístěn MDI-klient a případná ovládací lišta a stavový řádek (tool bar, status bar). Formuláře se pak typicky otevírají jako okna typu MDI-child, tedy potomci MDI-klienta.

MDI-klienta, ovládací lištu a stavový řádek může vytvořit *buď* **WinBase602**, *nebo* aplikace. Důležité je však dodržet toto pravidlo: **Všechna tato okna** (pokud existují) **musí být vytvořena stejným subjektem**. V opačném případě by se při změně velikosti okna MDI-frame někteří jeho potomci umístili přes sebe a v důsledku tohoto by například některé části později otevřených formulářů nemusely být viditelné.

MDI-klienta vytváří WinBase602

Varianta, v níž MDI-klienta, ovládací lištu a stavový řádek vytváří **WinBase602**, využijete zejména v těch aplikacích, které nepotřebují příliš velkou kontrolu nad těmito okny a v nichž převažuje práce s databází.

Ovládací lišta, vytvořená **WinBase602**, se mění v souladu s běžnými pravidly: při otevření formuláře se může objevit jeho soukromá lišta, také interní textový editor má vlastní lištu. Na stavový řádek může aplikace psát vlastní texty nebo čísla pomocí API funkcí.

Aplikace, která volí tuto variantu, postupuje takto:

1. Vytvoří hlavní okno aplikace jako MDI-frame. Jako poslední parametr funkce **`CreateWindow`** předá ukazatel na strukturu *WBFramePars* v níž zadáte tyto dodatečné parametry:

- Do složky *WinMenu* zapíšete pořadové číslo submenu ***Okna*** v hlavním menu, resp. hodnotu -1, pokud submenu ***Okna*** neexistuje.
- Pokud nechcete, aby okno mělo standardní ovládací lištu nebo stavový řádek, pak do složky *Flags* zapíšete hodnoty `WBF_NO_TOOLBAR` resp. `WBF_NO_STATUSBAR` resp. sjednocení obou. Jinak do *Flags* zapíšete nulu.

2. Zprávu `WM_CREATE`, kterou obdrží takto vytvářené okno, předá do funkce **`DefWinBaseFrameProc`** resp. **`WinBaseFrameProc`**.

Pro použití některých funkcí prezentační vrstvy musíte znát handle MDI-klienta. Tento handle vrátí (poté, co je MDI-klient vytvořen) funkce **`GetClient`**.

MDI-klienta vytváří aplikace

Druhou variantu, v níž MDI-klienta (ovládací lištu, stavový řádek) vytváří aplikace, je vhodné použít zejména tehdy, pokud vytvoření MDI-klienta je integrální součástí inicializace aplikace, jejíž kostru vytvořilo některé vývojové prostředí. Vyčleňovat vytváření MDI-klienta nebo předávat vlastní parametry vytvářenému oknu MDI-frame může pak být zcela nemožné.

Ovládací lišta a stavový řádek, vytvořené aplikací, jsou pod plnou kontrolou aplikace. **WinBase602** pomocí zpráv bude informovat aplikaci, že žádá o umístění textu nebo čísel na stavový řádek, případně o změnu ovládací lišty. Bude na aplikaci, aby zvážila, do jaké míry těmto požadavkům vyhoví, a případně je provedla.

Aplikace, která volí tuto variantu, postupuje takto:

1. Vytvoří MDI-frame a jeho subokna svým vlastním způsobem.
2. Zprávu WM_CREATE, kterou obdrží MDI-frame, **nesmí předat** do funkce **`DefWinBaseFrameProc`** ani **`WinBaseFrameProc`**.
3. Poté zavolá funkcí **`Init_window_env`** a tím informuje **WinBase602** o vytvořených oknech.

Inicializace uživatelského rozhraní bez MDI

Pokud hlavní okno aplikace není typu MDI, pak po jeho vytvoření je nutno provést inicializaci okenního prostředí **WinBase602** pomocí funkce **`Init_non_mdi_env`**.

Formuláře lze pak otevírat pouze jako modální nebo plovoucí okna.

Spolupráce WinBase602 s ovládací lištou aplikace

Aplikace může používat vlastní ovládací lištu, například objekt třídy **`CToolBar`** z ***MFC***. **WinBase602** pak do obsahu lišty přímo nezasahuje, ale v těch situacích, v nichž by měnila obsah své lišty, pošle hlavnímu oknu zprávu `SZM_SETTOOLBAR`. Typ lišty lze určit podle jejího parametru **wParam**, který může mít některou z následujících hodnot:

hodnota	Význam
WB_TB_UNKNOWN	neznámá lišta
WB_TB_DEFAULT	implicitní lišta
WB_TB_VIEW	lišta formuláře
WB_TB_VIEWNOMOVE	lišta formuláře, ve kterém nelze přecházet mezi záznamy
WB_TB_VIEWPROJ	lišta formuláře do proměnných projektu
WB_TB_VIEWQUERY	lišta dotazu
WB_TB_HIST	lišta pro okno historie
WB_TB_INDEX	lišta pro setřídění záznamů ve formuláři
WB_TB_EDITTEXT	lišta editačního okna

Aplikační program může na základě této zprávy přizpůsobit svoji lištu aktuálnímu stavu **WinBase602**.

V reakci na stisknutí tlačítka může uživatelská aplikace programově ovládat formuláře například pomocí níže popsaných zpráv.

Řízení formulářů a editoru

Aplikace v externím jazyce nemůže přímo volat metody a pracovat s vlastnostmi formulářů, s výjimkou případu, kdy je formulář začleněn do aplikace jako ActiveX objekt. Pokud aplikace chce zasahovat do formuláře, například z vlastního menu nebo ovládací lišty, má k dispozici dvě cesty.

První možností je zavolat proceduru vytvořenou ve vnitřním jazyce a umístěnou do projektu, která potřebnou manipulaci s formulářem provede. Takové procedury se dají volat po nastavení projektu pomocí funkce **`Exec_statements`**.

Druhou možností je zaslat oknu formuláře zprávu z níže uvedeného seznamu. Zprávy se zasílají aktivnímu oknu a nemají žádné parametry.

hodnota	význam
WM_CUT	Vystřihnout vybranou část položky
WM_COPY	Zkopírovat vybranou část položky do clipboardu
WM_PASTE	Vlepit do aktuální položky obsah clipboardu

Pohyb mezi záznamy

hodnota	význam
SZM_FIRSTREC	Na první záznam
SZM_PREVPAGE	Na předchozí stránku
SZM_PREVREC	Na předchozí záznam
SZM_NEXTREC	Na další záznam
SZM_NEXTPAGE	Na další stránku
SZM_LASTREC	Na poslední záznam

Pohyb mezi složkami

hodnota	význam
SZM_FIRSTITEM	Na první složku
SZM_LASTITEM	Na poslední složku
SZM_PREVTAB	Na předchozí složku
SZM_NEXTTAB	Na další složku
SZM_UPITEM	O složku výš
SZM_DOWNITEM	O složku níž

Dotaz QBE a uspořádání záznamů

hodnota	význam
SZM_QBE	Položení dotazu QBE
SZM_ACCEPT_Q	Akceptovat dotaz QBE
SZM_ORDER	Uspořádání záznamů ve formuláři
SZM_UNLIMIT	Zrušení omezení daného dotazem QBE nebo zrušení uspořádání

Vkládání a mazání záznamů

hodnota	význam
SZM_INSERT	Vložení nového záznamu
SZM_DELREC	Smazání záznamu
SZM_DELASK	Smazání všech záznamů
SZM_UNBINDDEL	Odvázání zrušených záznamů

Ostatní

hodnota	význam
SZM_PRINT	Tisk formuláře
SZM_INDEX	Zobrazení okna s hodnotami multiatributu
SZM_LOCKS	Zobrazení dialogboxu s nabídkou zamykání záznamů
SZM_HELP	Lokální nápověda

Editace

Editaci v textovém editačním okně lze ovládat pomocí zprávy `WM_COMMAND`, jejíž parametr **wParam**, bude mít některou z těchto hodnot:

hodnota	význam
MI_FSED_SAVE	Uložení editovaného textu
MI_FSED_FIND	Vyhledání textového řetězce
MI_FSED_REPLACE	Náhrada textového řetězce
MI_FSED_REFIND	Zopakování posledního hledání nebo náhrady řetězce
MI_FSED_FORMAT	Přeformátovat text

Stavový řádek

Obdobně jako v případě ovládací lišty, aplikační program může používat vlastní stavový řádek. WinBase602 **do něj pak přímo nezasahuje, ale v situacích, kdy chce na něj umístit text nebo čísla, zašle hlavnímu oknu aplikace zprávu.**

K tomuto účelu se používá zpráva `SZM_SETSTATUSTEXT`. Parametr **`wParam`** rozlišuje typ informace. Má hodnotu 0, pokud se vypisuje textové hlášení, nebo 1, pokud se vypisuje číslo aktuálního záznamu resp. průběžná čísla informující o postupu práce. Parametr **`lParam`** je ukazatel na vypisovaný text (čísla jsou předávaná v textovém podobě).

Menu a jeho přepínání

Aplikace vytvořená v externím jazyce může používat buď menu navržené ve **WinBase602** nebo vlastní menu.

Menu z **WinBase602**

Menu z **WinBase602** ovládáte z externího jazyka funkcí **`Main_menu`**. Aby menu používající objekty z projektu mohlo být otevřeno, je nutno projekt předem nastavit pomocí funkce **`Open_project`**.

Vlastní menu externí aplikace

Aplikace v externím jazyce může obsahovat vlastní menu. Toto menu může být zobrazeno různými způsoby, například zasláním zprávy `WM_MDISETMENU`.

K tomu, aby v aplikaci správně fungovalo přepínání menu při aktivaci oken, je nutno každé menu zobrazené jinak než funkcí **`Main_menu`** zaregistrovat voláním funkce **`Register_ext_menu`**.

Zprávy od menu editoru

Otevřením textového editoru bude uživatelské menu dočasně nahrazeno speciálním menu patřícím k editoru. Toto menu je na obrazovce právě tehdy, když je aktivním oknem editor. Zprávy pocházející od tohoto menu (například WM_COMMAND a WM_INITMENUPOPUP) je nutno předávat funkci **`DefWinBaseFrameProc`** resp. **`WinBaseFrameProc`**.

Příjem automaticky generovaných zpráv v externích jazycích

Automaticky generované zprávy jsou zasílány hlavnímu oknu aplikace jako zprávy `WM_COMMAND` s číslem 999 pomocí funkce **`SendMessage`**.

Parametry zprávy odpovídají konvencím ***Windows.*** Handle formuláře, který zprávu vyslal, je v parametru *lParam*, označení události je v horním slově parametru *wParam*, spodní slovo parametru *wParam* je rovno 999.

Ukončení činnosti nápovědného systému

Aplikace používající nápovědu je povinna před svým ukončením provést určité deinicializační akce. Tyto akce se provedou automaticky, pokud aplikace předá zprávu `WM_DESTROY` ke zpracování do funkce **`DefWinBaseFrameProc`**. Pokud tato zpráva není této funkci doručena, aplikace musí explicitně zavolat funkci **`Help_file`** s parametrem NULL.

Struktura aplikačního programu v jazyce C/C++

Aplikační program v jazyce C musí provést zahájení a ukončení komunikace s WinBase602 **podle níže uvedeného vzoru.**

Pro spolehlivou práci aplikací **WinBase602** je nutno zvětšit zásobník na cca 60 KB. Pro jazyk **C** zadejte tuto velikost v definičním souboru vaší aplikace (soubor s příponou DEF) příkazem:

```
STACKSIZE  60000
```

Kostra aplikace

Základní schéma aplikace v jazyce **C** naleznete v souboru APPL.C.

Volání funkce **`link_kernel`** je v tomto příkladu uvedeno tak, že aplikace bude hledat běžící databázový server a nabídne uživateli seznam jmen serverů. Pokud má aplikace komunikovat pouze s určeným serverem nebo pokud se má připojit přímo na lokální server, pak příslušně pozměníte parametry této funkce.

Volání funkce **`WinHelp`** v reakci na zprávy `WM_COMMAND` a `WM_DESTROY` zajišťuje aktivaci a deaktivaci nápovědy. Pokud nevytvoříte vlastní soubor nápověd, uveďte místo souboru MYHELP.HLP standardní soubor WB602.HLP.

Definiční soubory pro jazyk C

Podprogramy, typy, konstanty a proměnné podporující komunikaci s **WinBase602** jsou popsány ve dvou definičních souborech: WBKERNEL.H (komunikace s jádrem) a WBPREZEN.H (komunikace s prezentačním subsystémem). Tyto soubory dále používají definiční soubory GENERAL.H a CDP.H.

Pokud by váš program potřeboval využívat služby instalační knihovny, nalezne příslušné definice v INSTSERV.H .

Linkování aplikace

Při linkování je nezbytné rozlišovat mezi velkými a malými písmeny (tj. zapnout **`case-sensitive exports`** a **`case-sensitive link`**).

Do fáze linkování vašeho aplikačního programu přidejte v jazyce **C** importní knihovny WBKERNEL.LIB, WBPREZEN.LIB a případně WBVIEWED.LIB nebo INSTSERV.LIB . Použijete-li ***Borland C***, pak musíte vytvořit vlastní variantu těchto souborů pomocí utility IMPLIB, která je součástí ***Borland C***.

Aplikační programy využívající MFC

Součástí distribuční sady WinBase602 SDK **je i podpora pro aplikační programy psané v jazyce C++ s využitím knihovny tříd *MFC*.**

Podpora představuje implementaci tříd:

CWBApp	třída aplikačního programu WinBase602
CWBFrameWnd	třída hlavního okna aplikačního programu WinBase602
CWBViewWnd	třída formuláře
CWBModalViewWnd	třída modálního formuláře
CWBTable	třída tabulky
CWBCursor	třída kurzoru
CWBCutsTab	třída pro kurzor nebo tabulku

class CWBApp : public CWinApp

Základní třída, ze které lze derivovat uživatelské třídy aplikačních programů WinBase602**.**

Zabezpečuje inicializaci programu a zahajení komunikace s jádrem **WinBase602**. Aplikační program může obsahovat pouze jeden objekt vyderivovaný z třídy **`CWBApp`** a musí ho vytvořit jako globální objekt. Dále je třeba přepsat metodu **`InitInstance`**, která bude zpravidla vytvářet okno třídy **`CWBFrameWnd`** a volat funkci **`WBAppInit`**. Třída poskytuje tyto metody:

InitInstance Virtuální funkce, která má za úkol vytvořit hlavní okno aplikace a navázat komunikaci s databázovým serverem;

WBAppInit Naváže spojení se serverem a nastaví databázovou aplikaci;

OnInitErr Virtuální metoda, která se volá, jestliže se nepodařilo inicializovat komunikaci s databázovým serverem;

OnLoginErr Virtuální metoda, která se volá, pokud dojde k chybě při přihlašování uživatele;

OnSetAppErr Virtuální metoda, která se volá, pokud se nepodařilo nastavit databázovou aplikaci.

class CWBFrameWnd : public CMDIFrameWnd

Základní třída pro hlavní okno aplikačního programu WinBase602**.**

Zabezpečuje vytvoření MDI klienta, ovládací lišty, stavového řádku a vazbu na **`WinBaseFrameProc`**. Při uzavírání hlavního okna zajistí odhlášení uživatele a ukončení komunikace s jádrem. Uživatelský program zpravidla vyderivuje z **CWBFrameWnd** vlastní třídu, ve které doplní metody ošetřující jednotlivé zprávy. Hlavní okno aplikace se pak vytvoří ve dvou

krocích. Nejprve se zkonstruuje nový objekt a okno se otevře voláním metody **`Create`** nebo **`LoadFrame`**. Třída poskytuje tyto metody:

DefWinBaseFrameProc	Předání zpráv pro hlavní okno do **WinBase602;**
WinBaseFrameProc	Předání zpráv pro hlavní okno do **WinBase602;**
GetWindowMenuPopup	Virtuální metoda, která vrací handle submenu ***Okna;***
UpdateMenuProps	Informuje **WinBase602** o změně menu aplikace.

class CWBViewWnd : public CMDIChildWnd

Základní třída pro formuláře WinBase602**.**

Umožňuje ošetřovat zprávy pro formuláře prostřednictvím standardního mechanizmu knihovny MFC. Uživatelský program zpravidla vyderivuje z **`CWBViewWnd`** vlastní třídu, ve které doplní metody ošetřující jednotlivé zprávy. Formulář se pak otevírá ve dvou krocích. Nejprve se zkonstruuje nový objekt (zpravidla dynamický pomocí **new**) a vlastní formulář se otevře voláním metody **`Open`**, případně **`Select_records`** nebo **`Relate_record`**. Při zpracovávání zpráv je nutné všechny plně neošetřené zprávy předat funkci **`DefWindowProc`** nebo **`Default`**. Třída poskytuje následující metody:

Otevírání formuláře

Open	Otevře zadaný formulář;
Select_records	Otevře formulář, který umožní vybrat množinu záznamů;
Relate_record	Otevře formulář, který slouží k editaci relace 1:N nebo M:N mezi dvěma tabulkami.

Zavírání a rušení formuláře

Close_view	Zavře formulář;
PostNcDestroy	Virtuální metoda, která zruší objekt třídy **`CWBViewWnd`**.

Pohyb mezi záznamy a složkami

FirstRec	Na první záznam;
LastRec	Na poslední záznam;
NextRec	Na další záznam;
PrevRec	Na předchozí záznam;
NextPage	Na další stránku;
PrevPage	Na předchozí stránku;
FirstItem	Na první složku v záznamu;
LastItem	Na poslední složku v záznamu;
NextTab	Na další složku v záznamu;
PrevTab	Na předchozí složku v záznamu;
UpItem	O složku výš;

DownItem	O složku níž;
Current_item	Vrátí identifikátor aktuální složky;
Get_view_pos	Vrátí aktuální číslo záznamu;
Set_int_pos	Nastaví ve formuláři záznam s daným pořadovým číslem;
Set_ext_pos	Nastaví ve formuláři záznam s daným absolutním číslem.

Dotazy a uspořádání

OpenQBE	Přepne formulář do režimu zadávání QBE dotazu;
OpenSort	Přepne formulář do režimu zadávání uspořádání;
AcceptQuery	Akceptuje QBE dotaz nebo zadané uspořádání;
CancelQuery	Zruší QBE dotaz nebo uspořádání;
QBE_state	Vrátí indikaci režimu zadávání QBE nebo uspořádání.

Editace složky

Cut	Vystřihne vybranou část editované položky a uloží ji do schránky;
Copy	Zkopíruje vybranou část editované položky do schránky;
Paste	Vlepí do aktuální položky obsah schránky.

Editace v textovém okně

EditFind	Otevře dialog pro vyhledání řetězce v textu;
EditFormat	Přeformátuje text;
EditRefind	Najde další výskyt řetězce v textu;
EditReplace	Otevře dialog pro náhradu řetězce;
EditSave	Uloží editovaný text.

Vkládání a rušení záznamů

Insert	Vloží nový záznam;
DelAsk	Smaže všechny záznamy;
DelRec	Smaže aktuální záznam;
UnbindDel	Odváže zrušené záznamy.

Čtení a zápis hodnoty složky

Get_item_value	Vrátí hodnotu sloupce, do kterého vede zadaná složka formuláře;
Set_item_value	Zapíše zadanou hodnotu do složky ve formuláři i do příslušného sloupce.

Synchronizace změn ve formuláři a v databázi

Reset_view	Uvede do souladu obsah formuláře s obsahem databáze;
Commit_view	Zapíše změny provedené ve formuláři do databáze;
Roll_back_view	Zruší změny provedené ve formuláři.

Tisky

Print	Vytiskne sestavu na tiskárně;
Printer_dialog	Otevře dialog pro volbu tiskárny;
Print_opt	Otevře dialog pro nastavení parametrů generované sestavy.

Otevírání pomocných oken a dialogů

MultiAttrs	Otevře okno s hodnotami multiatributu;
Help	Otevře okno s lokální nápovědou;
Locks	Otevře dialog s nabídkou zamykání záznamů;
Select_file	Otevře dialog pro volbu souboru;
Select_directory	Otevře dialog pro volbu adresáře.

Ostatní

Active_view	Zjišťuje, který formulář je právě aktivní;
Get_fcursor	Vrátí číslo tabulky nebo kurzoru, do kterého formulář vede;
Set_fcursor	Přesměruje otevřený formulář na nový kurzor;
Pick_window	"Vytáhne" formulář nad všechna ostatní okna; Otevře dialog pro volbu tiskárny;
Register_rec_syn	Zajistí, že aktuální formulář bude se zadaným formulářem synchronizován na stejný záznam.

class CWBModalViewWnd : public CFrameWnd

Základní třída pro modální formuláře WinBase602**.**

Umožňuje otevírat a ovládat modální formuláře. Na třídě jsou definovány stejné metody jako na **`CWBViewWnd`**.

class CWBCursTab

Tato třída definuje objekt, který může za běhu programu reprezentovat kurzor nebo tabulku.

V uživatelských programech se zpravidla používají objekty tříd **`CWBTable`** nebo **`CWBCursor`**, které jsou potomkem této třídy. Vlastní objekt třídy **`CWBCursTab`** lze zpřístupnit voláním metody **`Get_fcursor`** nebo **`CWBViewWnd :: Get_fcursor`**. Třída poskytuje následující metody:

Vkládání a mazání záznamů

Append	Vloží nový záznam;
Insert	Vloží nový záznam;
Delete	Smaže zadaný záznam;
Delete_all_records	Smaže všechny záznamy.

Čtení z databáze

Read	Načte hodnotu zadaného sloupce;
Read_ind	Načte hodnotu zadaného sloupce pevné délky;
Read_ind_cnt	Načte počet hodnot multiatributu;
Read_len	Načte délku hodnoty sloupce proměnné délky;
Read_var	Načte hodnotu sloupce proměnné délky, nebo její část;
Read_record	Načte záznam (kromě hodnot sloupců proměnné délky a multiatributů).

Zápis do databáze

Write	Zapíše hodnotu zadaného sloupce;
Write_ind	Zapíše hodnotu zadaného sloupce pevné délky;
Write_ind_cnt	Nastaví počet hodnot multiatributu;
Write_var	Zapíše hodnotu sloupce proměnné délky nebo její část;
Write_len	Nastaví délku hodnoty sloupce proměnné délky;
Write_record	Zapíše záznam (kromě sloupců proměnné délky a multiatributů).

Zámky

Read_lock_record	Zamkne záznam pro čtení;
Read_lock_table	Zamkne tabulku pro čtení;
Read_unlock_record	Odemkne záznam pro čtení;
Read_unlock_table	Odemkne tabulku pro čtení;
Write_lock_record	Zamkne záznam pro přepis;
Write_lock_table	Zamkne tabulku pro přepis;
Write_unlock_record	Odemkne záznam pro přepis;
Write_unlock_table	Odemkne tabulku pro přepis.

Výpočty s hodnotou ve sloupci přes všechny záznamy

Rec_cnt	Vrátí počet záznamů v tabulce nebo kurzoru;
C_avg	Spočítá průměrnou hodnotu v zadaném sloupci;
C_count	Vrátí počet záznamů v tabulce nebo kurzoru, ve kterých má zadaný sloupec neprázdnou hodnotu;
C_max	Vrátí maximální hodnotu v zadaném sloupci;
C_min	Vrátí minimální hodnotu v zadaném sloupci;
C_sum	Vrátí součet hodnot v zadaném sloupci.

Zdroj dat pro formulář

Get_fcursor	Zpřístupní tabulku (resp. kurzor), která je zdrojem dat pro zadaný formulář;
Set_fcursor	Přesměruje zadaný formulář na nový zdroj dat,

Ostatní

Close	Uvolní zdroje potřebné pro přístup k tabulce nebo kurzoru;
IsTable	Vrací indikaci, zda objekt reprezentuje tabulku nebo kurzor;
Data_export	Exportuje data z databáze do souboru nebo do pošty;
Look_up	Najde záznam, ve kterém má zadaný sloupec zadanou hodnotu.

class CWBTable : public CWBCursTab

Třída objektů typu tabulka.

Objekt se vytváří ve dvou krocích. Nejprve se zkonstruuje nová instance a vlastní tabulka se zpřístupní voláním metody **`Open`** nebo **`Find_object`**.

Otevření a uvolnění

Open	Zpřístupní tabulku;
Find_object	Zpřístupní tabulku;
Close	Uvolní zdroje potřebné pro přístup k tabulce;
Uninst_table	Uvolní zdroje potřebné pro přístup k tabulce.

Informace o sloupcích

Attribute_info	Vrátí informace o zadaném sloupci;
Enum_attributes	Vrátí informace o všech sloupcích tabulky nebo kurzoru.

Práva

Get_data_rights	Vrátí práva uživatele k tabulce;
Set_data_rights	Nastaví práva uživatele k tabulce.

Smazané záznamy

Free_deleted	Uvolní smazané záznamy;
Undelete	Obnoví smazaný záznam.

Indexy

Enable_index	Aktivuje nebo deaktivuje zadaný index.

class CWBCursor : public CWBCursTab

Třída objektů typu kurzor.

Objekt se vytváří ve dvou krocích. Nejprve se zkonstruuje nová instance a vlastní kurzor se pak zpřístupní voláním některé z metod **`Open`**, **`Open_cursor`**, **`Open_cursor_direct`**, **`ODBC_open_cursor`** nebo **`Open_subcursor`**.

Otevírání a zavírání

Open	Otevře kurzor uložený v databázi nebo zadaný SQL příkazem;
Open_cursor	Otevře kurzor uložený v databázi;
Open_cursor_direct	Otevře kurzor zadaný SQL příkazem;
Odbc_open_cursor	Otevře kurzor na zadaném ODBC spojení;
Open_subcursor	Otevře subkurzor ze zadaného kurzoru;
Close	Zavře kurzor;
Close_cursor	Zavře kurzor.

Ostatní

Add_record	Přidá do kurzoru nový záznam;
Super_recnum	Převede číslo záznamu v kurzoru na číslo záznamu ve zdrojové tabulce nebo kurzoru;
Translate	Převede číslo záznamu v kurzoru na číslo záznamu ve zdrojové tabulce.

Implementace podpory MFC

Třídy popsané v předchozím textu jsou deklarovány v souboru WBMFC.H a implementovány ve WBMFC.CPP. Soubor WBMFC.H v sobě zahrnuje i definiční soubory WBKERNEL.H, WBPREZEN.H a CDP.H, tudíž je není nutné uvádět ve zdrojových textech aplikace. Stačí:

```
#include "wbmfc.h"
```

Soubor WBMFC.CPP je třeba přidat do projektu aplikačního programu nebo do seznamu zdrojových souborů v příkazové řádce překladače.

Ošetření automaticky generovaných zpráv pod MFC

Automaticky generované notifikační zprávy **WinBase602** (t.j. `WM_COMMAND`, číslo zprávy = 999) nelze pod MFC zachytit v mapě zpráv pomocí makra `ON_COMMAND`, protože toto makro je určeno pro ošetření zpráv z menu nebo od akcelerátorů (handle složky == 0). V případě automaticky generovaných zpráv je nutné použít makro `ON_CONTROL(wNotif,id,mFce)`, kde:

wNotif - označení události (1 = formulář otevřen, 2 = formulář uzavřen, ...)
id - 999
mFce - obslužná funkce

Příklad:

```
BEGIN_MESSAGE_MAP(CMainWnd, CWBFrameWnd)
    ...
    ON_CONTROL(3, 999, OnNewRec)
END_MESSAGE_MAP()

void CMainWnd :: OnNewRec()
{
    // Ošetření přechodu na nový záznam
    ...
}
```

Druhou cestou jak ošetřit automaticky generované zprávy, je vytvoření vlastní instance virtuální metody `OnCommand`.

Příklad:

```
BOOL CMainWnd :: OnCommand(WPARAM wParam, LPARAM lParam)
{
    if (wParam == 999)
    {
        switch (HIWORD(lPARAM))
        {
        case 1:     // Ošetření otevření formuláře
            ...
             break;
        case 2:     // Ošetření uzavření formuláře
            ...
            break;
        case 3:
            ...
         }
        return TRUE;
    }
    return CWBFrameWnd :: OnCommand(wParam, lParam);
}
```

Díky tomu, že standardní mechanismus ošetření zpráv v MFC předává zprávy `WM_COMMAND` aktivnímu MDI oknu, lze obě tyto varianty použít jak na úrovni hlavního

okna aplikace, tak na úrovni okna formuláře. Ovšem při zpracovávání automaticky generovaných zpráv na úrovni formuláře je třeba počítat s tím, že v době, kdy **WinBase602** posílá zprávu " Formulář byl otevřen" a první zprávu "Uživatel přešel na nový záznam" není okno formuláře ještě aktivní.

WinBase602 AppWizard

Nejjednodušším způsobem jak vytvořit kostru aplikace WinBase602 **v prostředí** *Microsoft Visual C++* **je použití návrháře** WinBase602 AppWizard**.**

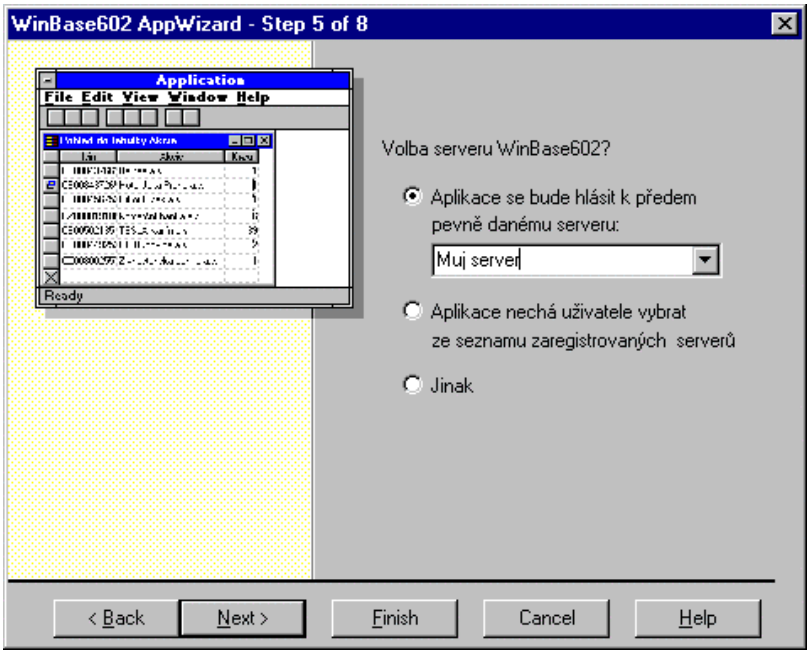
WinBase602 AppWizard je implementován v modulu WBWIZARD.AWX, který je součástí distribuční sady **WinBase602 SDK**. Modul je třeba zkopírovat do podadresáře ***TEMPLATE*** *v* adresáři vývojového prostředí **Microsoft Developer Studio** (implicitně C:\MSDEV\TEMPLATE), tím se stane **Aplikace WinBase602** součástí typů aplikací nabízených **AppWizardem**. Když vývojář při vytváření nového projektu zvolí typ **Aplikace WinBase602**, objeví se dialog podobný dialogu pro návrh standardní ***MFC*** aplikace doplněný o záložky, které popisují detaily týkající se spolupráce programu s **WinBase602**.

WinBase602 AppWizard umožňuje ovlivnit následující detaily:

Volba serveru WinBase602

- Server, ke kterému se aplikace bude hlásit, bude pevně daný, vývojář vybere jeho jméno ze seznamu zaregistrovaných serverů, nebo ručně uvede jeho jméno příslušném editovatelném comboboxu;
- Aplikace zobrazí seznam zaregistrovaných serverů podobný jako je na řídicím panelu vývojového prostředí a nechá uživatele vybrat;
- Volba serveru bude ponechána na vývojáři aplikace.

WinBase602 AppWizard



Volba jména uživatele

- Aplikace se bude hlásit k serveru jako anonymní uživatel;
- Aplikace se bude hlásit k serveru jako pevně daný uživatel bez hesla, vývojář uvede jeho jméno v příslušném editboxu;
- Aplikace zobrazí přihlašovací dialog;
- Volba jména uživatele bude ponechána na vývojáři aplikace.

Volba aplikace WinBase602

- Vyvíjená aplikace bude spolupracovat s pevně danou databázovou aplikací, vývojář vybere její jméno ze seznamu aplikací na zvoleném serveru, nebo ručně uvede její jméno příslušném editovatelném komboboxu;
- Vyvíjená aplikace zobrazí seznam aplikací na zvoleném serveru podobný jako je na řídicím panelu vývojového prostředí a nechá uživatele vybrat;
- Volba jména databázové aplikace bude ponechána na vývojáři.

Lišta s nástroji a stavový řádek

- Lištu s nástroji a stavový řádek bude kompletně obsluhovat **MFC;**
- Lištu s nástroji a stavový řádek bude kompletně obsluhovat **WinBase602.**

Inicializace komunikace s WinBase602

- Inicializace komunikace s **WinBase602** proběhne během inicializace aplikace;
- Inicializace komunikace s **WinBase602** proběhne až na základě nějaké události.

Vytvoření aplikace pomocí standardního MFC AppWizardu

Pokud vývojář z nějakého důvodu nemůže použít **WinBase602 AppWizard** a použije standardní **MFC AppWizard**, je třeba ve vygenerovaném projektu provést ručně některé úpravy:

- Do projektu přidat knihovny WBKERNEL.LIB a WBPREZEN.LIB a zdrojový soubor WBMFC.CPP.
- Na začátek hlavičkového souboru s deklarací třídy aplikace přidat příkaz :

  ```
  #include "wbmfc.h"
  ```

  a všechny odkazy na třídu **CWinApp** změnit na **CWBApp.**
- Obdobně začátek hlavičkového souboru s deklarací třídy hlavního okna přidat příkaz:

  ```
  #include "wbmfc.h"
  ```

  a všechny odkazy na třídu **CMDIFrameWnd** změnit na **CWBFrameWnd.**
- Ve zdrojovém souboru objektu aplikace všechny odkazy na třídu **CWinApp** změnit na **CWBApp** a na vhodné místo zařadit volání metody **`WBAppInit.`**
- Ve zdrojovém souboru objektu hlavního okna všechny odkazy na třídu **CMDIFrameWnd** změnit na **CWBFrameWnd** a případně upravit konstruktor hlavního okna.

Aplikace v Borland Pascalu a v Delphi

Základní definice pro aplikace v ***Borland Pascalu*** nebo ***Delphi*** komunikující s **WinBase602** jsou obsaženy v souboru WINBASE.PAS, který obsahuje potřebné typy, konstanty a funkce z DLL knihoven **WinBase602.** Tento soubor je třeba použít v aplikačním programu v klauzuli:

```
uses WinBase;
```

Zásobník

Pro spolehlivou práci aplikací **WinBase602** je nutno zvětšit zásobník na alespoň 40 KB. V **Pascalu** použijte direktivu **`$MINSTACKSIZE`** např. takto:

```
{$MINSTACKSIZE 40000 }
```

Přepínače

Při překladu programu musí být zapnut přepínač $X (extended syntax) a vypnut přepínač $A (word aling).

Využití rozhraní úrovně 1

Aplikace využívající tuto základní úroveň napojení na **WinBase602** musí mít strukturu odpovídající příkladům v souborech APPL1.PAS nebo APP2.PAS. Tyto příklady se liší tím, zda okno MDI-klient je vytvořeno aplikací nebo **WinBase602**.

Komponenty pro napojení na WinBase602

Cílem podpory založené na komponentách je zapouzdřit API tak, aby získalo charakter typický pro ***Delphi***. Zapouzdřené API je pak reprezentováno sadou komponent tvořících hierarchii. Tímto způsobem je možné zbavit programátora nutnosti pracovat s *handle* formuláře, aniž by byl zbaven možnosti nadále je používat, nadefinovat pro jednotlivé zprávy **WinBase602** *event handlery* a tím zpřehlednit jejich obsluhu, zbavit ho nutnosti pamatovat si parametry API funkcí, neboť je lze částečně přesunout do inspektoru objektů, využívat výhod IDE ***Delphi*** atd.

Podpora založená na komponentách předpokládá používání formulářů vytvořených ve **WinBase602**, zatímco lišty a menu jsou vytvořeny v ***Delphi***.

V době psaní tohoto manuálu byly k dispozici čtyři základní komponenty, které sice již poskytují základní prostředky pro vývoj aplikace, avšak programátor jistě pocítí potřebu tuto sadu rozšířit nebo zdokonalit. Zdrojový kód je dodáván s důkladnými komentáři spolu s podrobným popisem v elektronické nápovědě, takže je poskytnuta možnost soustavu komponent dále rozšiřovat a upravovat. Sada komponent bude ze strany **Software602** dále rozšiřována a s příslušnými nápovědami budou tato rozšíření přístupná na WWW serveru. Samozřejmě je možné souběžně s komponentami používat i API **WinBase602**.

Vytvořeny jsou tyto čtyři komponenty:

- **`TWBaseServer602 = class(TForm)`** nevisuální komponenta, kterou budete používat jako předka vašeho rámcového okna. Třída především:

1. zapouzdřuje funkčnost pro napojení k a odpojení od databázového serveru **WinBase602**,
2. zajišťuje předávání zpráv mezi databázovým serverem **WinBase602** a ***Delphi***.

- **`TMsgFrame = class(TComponent)`** nevisuální komponenta pro obsluhu zpráv z **WinBase602** přijímaných rámcovým oknem. Implementuje sadu událostí odpovídajících těmto zprávám.

Tyto dvě komponenty jsou definovány v jednotce MSGFRAME.PAS (bitmapa je v MSGFRAME.DCR).

- **`TWBNavig = class(TCustomPanel)`** analog ke standardní komponentě z ***Delphi*** `TDBNavigator`. Slouží k ovládání formulářů. Poskytuje navíc sadu událostí typu `OnBeforeDelete`.

Komponenta je definována v jednotce WBNAVIG.PAS (bitmapa je v WBNAVIG.DCR).

- **`TView602 = class(TComponent)`** nevisuální komponenta zapouzdřující formulář. Vedle základních metod `Open`, `Close` a `Toggle` implementuje několik událostí.

Komponenta je definována v jednotce VIEW602.PAS (bitmapa je v VIEW602.DCR).

Viz téma v elektronické nápovědě 'Podpora pro ***Delphi*** aplikace založená na zapouzdření API do komponent ***Delphi***'.

Na distribuční disketě SDK je umístěn vzorový projekt (s komentáři) WB_D_CMP.DPR + jednotky, který ukazuje aplikaci, jež provádí ruční (tj. z menu) napojení se na databázový server, spuštění aplikace **PER_AGENDA** vytvořené ve **WinBase602** a následné odpojení se od serveru. Z příkladu je patrné použití a účel výše zmíněných čtyř komponent, jakož i struktura aplikace založené na těchto komponentách. V jednotce MAIN.PAS příkladu nahraďte název serveru WB5 názvem serveru, na kterém máte uloženy vzorové aplikace dodávané s **WinBase602**. Jednu z těchto aplikací (PER_AGENDA) příklad používá. Pokud jde o automatické přihlášení se k serveru **WinBase602** (tj. hned po startu aplikace ***Delphi***), naleznete podrobnosti v elektronické nápovědě pro `TWBaseServer602.WBinit` .

Komponenty jsou po instalaci implicitně umístěny na stránku palety WB602.

Při vytváření aplikace v ***Delphi*** nezapomeňte umístit komunikační jednotku WINBASE.PAS do sekce **`uses`** v těch souborech, v nichž používáte API **WinBase602**. Dále umístěte WINBASE.PAS též do adresáře, ze kterého budete instalovat výše zmíněné komponenty.

Třída pro podporu aplikací založených na API

Pokud chcete dát přednost vytváření ***Delphi*** programu téměř výlučně na úrovni API, pak můžete využít podpory založené na třídě **`TWBMainForm = class(TForm),`** kterou budete používat jako předka vašeho rámcového okna. Třída pouze zapouzdřuje funkčnost pro napojení k a odpojení od databázového serveru **WinBase602** a zajišťuje předávání zpráv mezi databázovým serverem **WinBase602** a ***Delphi***. Všechny ostatní úlohy jako např. otevírání formulářů budete řešit přímým voláním funkcí API **WinBase602.** Viz téma v elektronické nápovědě 'Podpora pro ***Delphi*** aplikace založená na funkcích a zprávách z **WinBase602** API'. Třída je uložena v jednotce, WBPARENT.PAS, kterou musíte zařadit mezi jednotky vašeho projektu, založeného na třídě **`TWBMainForm.`**

Na distribuční disketě SDK je umístěn vzorový projekt (s komentáři) WB_D_API.DPR + jednotky, který ukazuje napojení se na databázový server, spuštění aplikací vytvořených ve **WinBase602** a následné odpojení se od serveru. Z příkladu je patrné jak použití a účel třídy **`TWBMainForm`**, tak i struktura aplikace založené na této třídě. V jednotce MAIN.PAS příkladu nahraďte název serveru WB5 názvem serveru, na kterém máte uloženy vzorové aplikace dodávané s **WinBase602**. Tři z těchto aplikací (PER_AGENDA, PŘÍKLADY, SERVIS) příklad používá.

Vzorové aplikace (PER_AGENDA, PŘÍKLADY, SERVIS) používané v tomto příkladu musíte doplnit tak, aby položka menu **WinBase602** ukončující práci s **WinBase602** vyslala při své aktivaci zprávu 1001. Touto zprávou je ohlášeno do ***Delphi*** (viz `procedure TWBMainForm.WMCOMMAND(var Msg: TMessage);`), že aplikace **WinBase602** (např. PER_AGENDA) chce skončit.

Nezapomeňte v jednotce hlavního okna aplikace (ve vzorovém projektu MAIN.PAS), přidat do deklarace **uses** jednotky WBPARENT.PAS a WINBASE.PAS**:**

```
uses Windows, SysUtils, Classes, Graphics, Forms, Controls,
      Menus,StdCtrls, Dialogs, Buttons, Messages, ExtCtrls,
      ComCtrls, WBParent, WinBase;
```

Aplikace ***Delphi*** sloužící k volání formulářů **WinBase602** prostřednictvím jejího API (ať zapouzdřeného do komponent či nezapouzdřeného) musí být MDI aplikací. Formuláře **WinBase602** jsou pak okny typu MDI child.

Aplikační programy ve Visual Basicu

Obdobně jako programy v jazyce **C** nebo **Pascal** může program ve **Visual Basicu** přistupovat k objektům **WinBase602** prostřednictvím funkcí z DLL knihoven WBKERNEL.DLL, WBPREZEN.DLL a WBVIEWED.DLL. Deklarace těchto funkcí a deklarace příslušných konstant a struktur jsou uvedeny v souborech WBKERNEL.TXT a WBPREZEN.TXT. Tyto soubory není nutné začleňovat do zdrojového textu celé, stačí z nich vybrat pouze potřebné deklarace.

Navázání spolupráce s **WinBase602** zabezpečují funkce **`cdp_alloc`**, **`cdp_init`**, **`link_kernel`**, **`interf_init`**, **`Login`** resp. **`Alogin`** a **`Set_application`**. V úvodní části aplikace, tzn. v proceduře **`Main`** nebo v proceduře **`Load`** hlavního okna aplikace, je třeba zavolat funkce **`cdp_alloc`** a **`cdp_init`**, které inicializují vnitřní datové struktury, potřebné pro provoz **WinBase602**. Funkce **`link_kernel`** a **`interf_init`** inicializují spojení s databázovým serverem. Funkci **`interf_init`** se jako první parametr předává hodnota, kterou vrátila funkce **`cdp_alloc`**. Přihlášení uživatele zabezpečí funkce **`Login`** nebo **`Alogin`** a funkce **`Set_application`** nastaví požadovanou databázovou aplikaci. Pokud aplikace používá formuláře **WinBase602**, je nutné, aby její hlavní okno bylo vygenerováno z objektu `MDIForm`. Při otevírání tohoto okna, nejlépe v proceduře **`Load`**, je třeba zavolat funkci **`ConWinBaseFrameProc`**. Tato funkce zprostředkovává spolupráci formulářů s hlavním oknem aplikace, aktualizaci menu, zobrazení lišty s nástroji apod. Má dva parametry, prvním je *handle* hlavního okna a druhý udává pořadové číslo submenu ***Okna*** (***Window***) v hlavním menu (pokud aplikace submenu ***Okna*** nemá, předává se -1). Zahájení spolupráce s **WinBase602** může vypadat například takto:

```
Sub MDIForm_Load ()
    ConWinBaseFrameProc hWnd, 3  ' Napojení na hlavní okno
    cdp = cdp_alloc()       ' Alokace datových struktur
    cdp_init(cdp)           ' Inicializace datových struktur
    sts = link_kernel("", SW_MINIMIZE)' Spuštění serveru
    If (sts <> KSE_OK) Then
        MsgBox "Nejde spustit server", MB_OK
        End
    End If

    sts = interf_init(cdp, 0)     ' Inicializace spojení
    If (sts <> KSE_OK) Then
        MsgBox "Nejde inicializovat spojení se serverem", MB_OK
        End
    End If

    sts = Login("", "")           ' Přihlášení uživatele
    If (sts = 0) Then
        interf_close
        End
    End If

    sts = Set_application("Faktury")' Nastavení aplikace
    If (sts <> 0) Then
        MsgBox "Zadaná aplikace v databázi není", MB_OK,
"Fakturace"
        sts = Logout
        interf_close
        End
    End If
End Sub
```

Ukončení práce s **WinBase602** zajišťují funkce **`Logout`** a **`interf_close`**. Je vhodné je umístit do procedury **`Unload`** objektu `Form` resp. `MDIForm`. např.:

```
Sub Form_Unload (Cancel As Integer)
     s = Logout
     interf_close
End Sub
```

Funkce **`Delete`**, **`Undelete`**, **`Read`**, **`Write`**, **`Append`**, **`Insert`**, **`Day`**, **`Month`**, **`Year`**, **`Now`** a **`Like`** jsou kvůli kolizi s vyhrazenými jmény ***Visual Basicu*** přejmenovány na **`WBDelete`**, **`WBUndelete`**, **`WBRead`**, **`WBWrite`**, **`WBAppend`**, **`WBInsert`**, **`WBDay`**, **`WBMonth`**, **`WBYear`**, **`WBNow`** a **`WBLike`**.

Struktura `modifrec` je pro potřeby **Visual Basicu** deklarována jako:

```
Type modifrec
```

```
    modtype As String * 1
    word1   As Integer
    word2   As Integer
    word3   As Integer
End Type
```

Složka `modtype` má rozměr 1 byte a proto je třeba ji inicializovat prostřednictvím funkce **`Chr$`** např.:

```
mf.modtype = Chr$(MODINT)
```

Zbývající složky **word1**, **word2** a **word3** mají v jednotlivých případech následující význam:

Složka	**Význam**
MODIND	Word1 = index multiatributu
MODINT	Word1 = nižší bity ofsetu Word2 = vyšší bity ofsetu Word3 = velikost bloku
MODPTR	Word1 = číslo sloupce
MODINDPTR	Word1 = intex hodnoty Word2 = číslo sloupce

Pro snadnější přístup ke sloupcům proměnné délky je navíc definována struktura:

```
Type modint
      modtype As String * 1       ' = CHR$(MODINT)
      start As Long         ' Offset od začátku hodnoty
      size  As Integer            ' Velikost bloku
End Type
```