

Vysoké učení technické v Brně  
Fakulta elektrotechniky a informatiky  
Ústav informatiky a výpočetní techniky

## SQL Windows

Učební pomůcka pro laboratoře kursu

Databázové systémy  
*(transformováno ze souboru ve formátu T602 bez následného formátování)*

Doc.Ing.Jaroslav Zendulka,CSc. Brno 1994

## Obsah

1 Úvod.....	3
2 Definice uživatelského rozhraní.....	3
2.1 Nástroj Outliner.....	3
2.2 Grafické objekty.....	4
2.3 Pohledy na aplikaci.....	5
2.4 Menu okna Outline.....	6
2.5 Přidávání a editace grafických objektů rozhraní.....	8
2.6 Přízpůsobení (customizing) grafických položek.....	8
2.7 Kopírování.....	9
2.8 Jména objektů.....	9
2.9 Režimy činnosti nástroje Outliner.....	9
3 Jazyk SAL.....	10
3.1 Datové typy.....	10
3.2 Proměnné.....	11
3.3 Kvalifikované odkazy.....	12
3.4 Speciální klíčová slova.....	12
3.5 Získání identifikátoru instance okna.....	12
3.6 Pole.....	12
3.7 Konstanty.....	13
3.8 Operátory.....	13
3.9 Příkazy.....	13
3.10 Poznámky.....	14
3.11 Funkce.....	15
3.12 Prvky zprávy wParam a lParam.....	15
3.13 Použití Outline Options.....	15
3.14 Spuštění aplikace, překlad, ladění.....	15
4 Programování aplikace v jazyce SAL.....	15
4.1 Vytváření a rušení okna na nejvyšší úrovni.....	15
4.2 Definování funkcí.....	19
4.3 Okna MDI.....	20
4.4 Použití databáze.....	20
4.5 Přihlašovací blok.....	23
4.6 Okno seznamu.....	24
4.7 Kombo box.....	25
4.8 Radiové tlačítko.....	26
4.9 Řídicí okno.....	26
4.10 Posouvací sloupek (scroll bar).....	26
4.11 Maskování vstupu, formátování a validace polí.....	27
4.12 Fonty a barvy.....	29
4.13 Okno se zprávou pro uživatele.....	29
4.14 Programátorem definované zprávy.....	30
4.15 Obrázky, tlačítka, práce s objekty OLE.....	33
4.16 Okno tabulky - základy.....	33
5 Tiskové sestavy.....	40
5.1 Tiskové sestavy z databáze nebo souboru.....	41
5.2 Tiskové sestavy pro okno tabulky.....	43
5.3 Položky menu pro tisk v zákaznickém tiskovém okně.....	44
5.4 Obnova a fitrování dat sestavy.....	44
Literatura.....	44

# 1 Úvod

SQL Windows představuje produkt, který slouží pro vývoj databázových aplikací klient - server běžících pod Microsoft Windows. Tvorba aplikace probíhá ve dvou krocích:

**V prvním kroku** programátor definuje zpravidla interakčním způsobem grafický vzhled uživatelského rozhraní tím, že vytváří grafické objekty a rozmísťuje je na ploše obrazovky. Současně přizpůsobuje parametrizované vlastnosti objektů svým potřebám.

**Ve druhém kroku** provádí vlastní programování zadáváním příkazů **jazyka SAL** (SQL Application Language) popisujících chování vytvořených objektů. Programování je založeno na řízení zprávami. Systém SQL Windows zasílá zprávy objektům, jichž se týkají (např. stisk tlačítka myši na objektu typu tlačítko). Podstata programování v jazyce SAL potom spočívá především v příkazech popisujících reakce na přijaté zprávy. I programátor může definovat vlastní zprávy a zasílat je objektům.

Jazyk SAL obsahuje poměrně malý sortiment základních příkazů, používá však celou řadu systémových funkcí a je definována řada systémových zpráv.

I vlastní programování lze provádět do určité míry interaktivně, neboť lze využít kontextově závislé nabídky položek, které lze na dané místo popisu vložit.

## 2 Definice uživatelského rozhraní

### 2.1 Nástroj Outliner

SQL Windows se spustí z prostředí Windows dvojným stiskem tlačítka myši na odpovídající ikoně skupiny SQL Windows 4.1. Na obrazovce se objeví 3 okna:

a) **Okno nástroje zvaného Outliner** (Outline window), které obsahuje ikony různých pohledů na aplikaci. Pohledem se zde rozumí strukturovaný textový popis určitého objektu definující obecně jeho strukturu a chování. Takovým pohledem může být aplikace (Main), formulářové okno, interní funkce atd. Při vytváření nové aplikace jsou některé objekty a jim odpovídající pohledy vytvořeny automaticky.

Okno nástroje Outliner obsahuje titulek, řádek menu, řádek nástrojů, pracovní plochu a stavový řádek. Zobrazení řádku nástrojů a stavového řádku je nastavitelné.

Výběrem ikony (dvojným stiskem tlačítka) se otevře okno se strukturovaným popisem odpovídajícího objektu. Ten sestává z řady položek (sekcí). Každá položka je uvozena zleva ikonou ve tvaru plného nebo prázdného kosočtverce. Prázdný kosočtverec znamená, že položka již neobsahuje položky úrovně nižší, naopak dvojným stiskem na ikoně plného čtverce se položka rozvine a zobrazí se synovské položky. Opětovným dvojným stiskem tlačítka se položka opět sbalí.

Kromě **ikon** ve tvaru kosočtverce se u některých položek vyskytuje další ikona indikující, že položka má parametry, které lze nastavit dvojným stiskem tlačítka na této ikoně.

Sekce lze rozlišit na pevné, jejichž místo v pohledu je pevné, nelze je zrušit, ani přemístit (např. sekce Global Declarations, Application Description), naopak volitelné sekce lze vkládat, rušit, kopírovat apod. Volitelné sekce typicky odpovídají grafickým položkám a samy obsahují jiné sekce pevné, případně volitelné.

b) **Okno objektu frmMain**, což je formulářové okno vytvořené automaticky s předdefinovaným menu s položkami File a Edit.

c) **Okno palety nástrojů**. Slouží k výběru objektů, které se používají v prvním kroku tvorby aplikace, tj. při definici uživatelského rozhraní. Takovým objektem je např. formulářové okno, tlačítko apod. Paleta obsahuje tlačítka pro každý typ objektu, který lze do aplikace při definici rozhraní přidat. Kromě toho obsahuje tlačítka některých dalších nástrojů (pro výběr objektu, výběr skupiny objektů, apod.).

Poznámka: Většina činností, které lze vyvolat stiskem tlačítka myši, ukazuje-li kurzor na určitý objekt, je dostupná i z menu v horní části nástroje Outliner.

## 2.2 Grafické objekty

Grafické objekty jsou viditelné elementy, ze kterých se vytváří uživatelské rozhraní. Existují tři druhy grafických položek, které se liší možností parametrizace, zadávání kódu definujícího chování a schopností přijímat zprávy. Jde o tyto tři druhy:

druh parametry kód zprávy

objekty typu okno ano ano ano

položky pozadí ano ne ne

položky menu ne ano ne

### Objekty typu okno (windows objects)

Jsou to programovatelné objekty, které lze rozdělit do dvou skupin:

a) **okna nejvyšší úrovně**, která představují nejvyšší úroveň objektů v rámci dané aplikace.

Do této skupiny patří tři typy oken:

- **formulářové okno** - je nezávislé okno pro obecné operace. Může obsahovat řadu oken synovských. S formulářovým oknem může provádět koncový uživatel činnosti běžné v prostředí Windows (přemísťování, změna velikosti, atd.)

- **dialogové okno** - slouží uživateli k zadání nějaké informace (např. potvrzení) pro ukončení činnosti probíhající v jiném okně. Vyvolání dialogového okna může omezit chování uživatele, tj. může vyvolat speciální režim. Z tohoto pohledu rozlišujeme tři typy dialogových oken:

**režimové** (modal) - nastavuje speciální režim v rámci aplikace, tj. potlačuje činnosti mimo okno, ale pouze v rámci aplikace (např. nereaguje na stisk jiného tlačítka, než v dialogovém okně).

**systémově režimové** (system modal) - nastavuje speciální režim v celém systému, dokud není dialogové okno uzavřeno.

**nerežimové** - nevyvolává speciální režim, okno může zůstat otevřeno a činnost probíhat mimo ně.

- **okno tabulky** - okno zobrazující údaje ve tvaru tabulky. Jejím prostřednictvím lze data prohlížet, vkládat, rušit nebo měnit.

b) **synovská okna** (child windows) - objekty, které jsou součástí (tvoří obsah, a proto se jejich popis objevuje v sekci Contents) okna nejvyšší úrovně (rodičovského okna). Synovská okna nelze při běhu aplikace přemísťovat, měnit velikost a rušit. Patří sem tyto typy oken:

- **datové pole** (data field) - pole pro zobrazení nebo vstup hodnoty (text, číslo, datum, datum/čas). Může být editovatelné nebo pouze pro zobrazení.

- **víceřádkový text** (multiline text box) - několik řádků textu, které lze editovat nebo jen zobrazovat, text lze rolovat nebo zalamovat v okně.

- **seznam** (list box) - může obsahovat několik textových položek, které nelze editovat.

- **kombobox** (combo box) - je kombinací datového pole a seznamu. V datovém poli se zobrazuje položka vybraná ze seznamu. Seznam může být rozvinutý nebo ne.

- **synovská tabulka** (child table) - obdoba okna tabulky, ale jako synovské okno, tj. bez titulku a menu.

- **sloupec okna tabulky** (table window column) - sloupec tabulky okna tabulky nebo synovské tabulky. Může obsahovat editovatelná nebo pouze zobrazitelná data.

- **obrázek** (picture) - pole pro obrázek, ikonu a objekty OLE (Object Linking and Embedding). Může tvořit pouze pozadí nebo může pracovat jako tlačítko. Může být editovatelný.

- **rádiové tlačítko** (radio button) - prvek vyskytující se v sestavě jiných rádiových tlačítek, které tvoří booleovské spínače. Tlačítka ve skupině jsou vzájemně výlučná, tj. pouze jedno

může být aktivní. V textovém popisu chápe Outliner souvislý seznam tlačítek jako jednu skupinu.

- **kontrolní okno** (check box) - booleovský spínač, který však na rozdíl od rádiového tlačítka není součástí skupiny vzájemně vylučných tlačítek. Typický použitím je např. označení výběru z nabízených možností.

- **tlačítko volby** (option button) - čtvercové tlačítko, často s obrázkem, které lze použít jako rádiové nebo kontrolní nebo tlačítko v paletě nástrojů.

- **tlačítko** (push button) - obdélníkové tlačítko, které se používá typicky u formulářů a dialogových oken k výběru činnosti.

- **posouvací sloupec** (scroll bar) - pro znázornění pozice v rozsahu hodnot.

- **zákaznické řízení** (custom control) - zákaznické okno mající libovolnou z různých forem, např. přístrojový panel. Jednotlivé řídicí prvky jsou definované programátorem s využitím DLL (Dynamic Link Library).

#### **Položky pozadí** (background items)

Slouží především ke zlepšení vzhledu rozhraní. Patří sem tyto typy objektů:

- **text na pozadí** (background text) - používá se např. jako návěští datových polí na formulářích.

- **skupina** (group box) - slouží k vyznačení skupiny objektů, typicky rádiových tlačítek. Je tvořena rámečkem a titulkem.

- **oddělovač skupiny** (group separator) - objekt viditelný pouze v textovém popisu nástroje Outliner. Jde o řádek sloužící k oddělení dvou skupin rádiových tlačítek.

- **úsečka** (line) - grafický prvek pro vylepšení vzhledu.

- **rámeček** (frame) - grafický prvek pro vylepšení vzhledu.

#### **Položky menu** (menu elements)

- **roletové menu** (popup menu) - může být v řádku menu (menu bar) okna formuláře nebo tabulky, v přidaném řádku menu pod ním nebo uvnitř jiného roletového menu (v takovém případě je další úroveň menu indikována u položky menu).

- **pojmenované menu** (named menu) - vypadá a chová se stejně jako roletové menu. Rozdíl spočívá v tom, že může být přidáno do násobných oken a vytvořeno při běhu aplikace na specifikovaných souřadnicích. Při vytváření nové aplikace jsou předdefinována menu menuEdit, menuOLEEdit a menuMDIWindows. K vytvoření pojmenovaného menu slouží funkce jazyka SAL SalTrackPopupMenu().

- **menu oken** (windows menu) - speciální menu pro správu synovských MDI (Multiple Document Interface) oken. Může být přidáno pouze do oken této kategorie.

- **položka menu** (menu item) - položka roletového menu nebo řádku menu.

- **řádek menu** (menu row) - řádek s položkami menu pod řádkem titulku okna a standardního řádku menu (menu bar).

- **oddělovač menu** (menu separator) - horizontální čára mezi položkami roletového menu k vizuálnímu oddělení skupin položek.

- **sloupec menu** (menu column) - obsahuje položky, které vytvoří další sloupec položek, místo aby byly položky umístěny v jednom sloupci (např. pokud by byl sloupec příliš dlouhý).

## **2.3 Pohledy na aplikaci**

Při rozsáhlé aplikaci je užitečné mít možnost vidět části návrhu v samostatných oknech. To umožňuje položka menu **Window - New Outline View**. Volbou této položky se vytvoří ikona reprezentující vybranou sekci textového popisu.

## 2.4 Menu okna Outline

Menu okna Outline obsahuje položky **File**, **Edit**, **View**, **Outline**, **Arrange**, **Tools**, **Run**, **Windows** a **Help**. Každá z těchto položek (kromě **Help**) vyvolá roletové menu, které v některých případech (File, Edit, Windows) obsahuje položky běžné pro aplikace pod Windows. V této části jsou stručně popsána jednotlivá menu.

### Menu File

**New** - vytvoří novou nepojmenovanou aplikaci.

**Open** - na základě dialogu otevře vybranou aplikaci (standardně s příponou .app). Lze mít otevřeno pouze jednu aplikaci.

**Save** - uloží otevřenou aplikaci na disk. Nemá-li aplikace dosud jméno, je nejprve požadováno jméno.

**Save As** - uloží aplikaci pod zadaným jménem. Aplikace může být uložena ve vnitřním formátu SQL Windows (normal), ve zkompilem tvaru (compiled), v textovém tvaru (text) nebo textovém s odsazením sekcí jednotlivých úrovní (Indented Text).

**Libraries** - umožňuje práci s knihovnami objektů.

**Make Executable** - vytvoří spustitelnou verzi otevřené aplikace (.EXE). Při spuštění musí být v cestě SQLRUN40.EXE.

**Make Runtime** - vytvoří spustitelnou aplikaci (.RUN), která se dává jako parametr SQLRUN40.EXE.

**Print Outline** - tisk celého textového popisu aplikace nebo jeho části.

**Printer Setup** - nastavení tiskárny.

**Preferences** - zobrazí dialogové okno, kde lze nastavit jméno aplikace, ze které se vytváří nová aplikace (standardně NEWAPP.APP), implicitní příponu souboru aplikace (.APP) a některé další.

**Exit** - ukončuje činnost SQLWindows.

### Menu Edit

**Undo** - vrací stav před poslední editací.

**Cut** - uloží vybrané položky (textové nebo grafické) do schránky (clipboard).

**Copy** - zkopíruje vybrané položky do schránky.

**Paste** - zkopíruje obsah schránky na místo, kde je kurzor myši.

**Paste Link ...** - zkopíruje OLE objekt ze schránky a vytvoří vazbu na zdrojovou aplikaci, kde objekt vznikl.

**Paste Special ...** - umožňuje zvolit formát schránky, než je zkopírována obrázek.

**Paste From ...** - uložení obrázku ze souboru.

**Clear** - zruší vybrané položky (totéž klávesa Delete).

**Grid** - nastavení zaokrouhlování souřadnic do rastru mřížky.

**Show Grid** - zobrazení rastru mřížky pro zaokrouhlování souřadnic.

**Align to Grid** - umístí vybrané položky do nejbližších bodů rastru.

**Find** - vyhledání textového řetězce v pohledu na aplikaci.

**Repeat Last Find** - opakuje poslední hledání textového řetězce.

**Change** - vyhledání a záměna textového řetězce v pohledu na aplikaci.

**Links ...** - zobrazí dialogové okno s vazbami objektů OLE.

**Object** - spustí aplikaci, která vytvořila vybraný OLE objekt.

**Insert Object ...** - lze vložit OLE objekt zvolené aplikace vytvářející OLE objekty.

### Menu View

**Toolbar** - zobrazení řádku s ikonami nástrojů.

**Status Bar** - zobrazení stavového řádku.

**Outline Options** - zobrazí dialogové okno Outline Options s nabídkou položek, které lze vložit na danou pozici.

**Library Browser** - prohlížení knihoven objektů.

**Tool Palette** - zobrazení palety nástrojů.

**Show Window** - otevře vybrané okno nejvyšší úrovně.

**Hide Window** - uzavře vybrané okno nejvyšší úrovně.

**Temporary Positioning** - změny pozic oken budou dočasné (nepoznačí se do popisu).

### **Menu Outline**

**Insert Line** - vloží řádek pro novou položku popisu.

**Comment Items** - zapoznámkuje vybrané položky popisu (začínají vykřičníkem a mezerou).

**Uncomment Items** - odstraní zakomentování vybraných položek popisu.

**Expand One Level** - rozvine vybrané položky o jednu úroveň (totéž jako dvojitý stisk tlačítka myši na ikoně u položky).

**Collapse** - uschová synovské položky vybrané rodičovské položky v popisu (totéž jako dvojitý stisk tlačítka myši na ikoně u položky).

**Expand All Levels** - rozvine všechny úrovně vybrané položky popisu.

**Collapse Outline** - uschová položky nižší úrovně, takže zůstanou pouze položky nejvyšší úrovně.

**Move Left** - posune vybrané položky o jednu úroveň odsazení doleva.

**Move Right** - posune vybrané položky o jednu úroveň odsazení doprava.

**Move Up** - přesune vybrané položky nad předchozí položku v popisu.

**Move Down** - přesune vybrané položky pod předchozí položku v popisu.

### **Arrange Menu**

**Bring To Front** - přemístí vybrané objekty na nejvyšší vrstvu, tj. nad ostatní objekty.

**Send To Back** - umístí objekty na nejvyšší vrstvě za jiné objekty.

**Tab Order** - zobrazí dialogové okno, prostřednictvím něhož se specifikuje pořadí aktivity (focus) objektů při komunikaci s uživatelem (pohyb kurzoru po položkách formuláře).

Implicitně odpovídá tato posloupnost pořadí, v jakém jsou objekty uvedeny v popisu. Při nastavování se zobrazují u objektů pořadová čísla.

**Align** - zarovnání vybraných objektů.

**Even Spacing** - zajistí stejné mezery mezi vybranými objekty.

**Equal Sizing** - zajistí stejnou velikost vybraných objektů (jako prvního).

### **Menu Tools**

**Windows** - zobrazí menu, ze kterého lze vybrat okno nejvyšší úrovně nebo MDI okno a přidat ho k aplikaci.

**Draw** - zobrazí menu, ze kterého lze:

a) vybrat nástroj (pro výběr objektu (Window Grabber), skupiny objektů (Object selector), duplikaci vybraných objektů (Object Duplicator).

b) vybrat synovský objekt, který bude umístěn do aktivního okna nejvyšší úrovně (viz synovská okna v kap.2.2).

Položka Draw poskytuje stejné funkce jako tlačítka palety nástrojů (Tool Palette).

### **Menu Run**

**User Mode** - přeloží a spustí aplikaci. Aplikace se chová jako u koncového uživatele s tím rozdílem, že Outliner zůstává otevřen.

**Compile** - přeloží aplikaci.

**Break** - umožňuje nastavit bod zastavení na vybraný příkaz, zrušit vybraný bod zastavení, zrušit všechny body zastavení, nalézt další bod zastavení v aplikaci a nastavit zobrazování dialogového okna ladění při běhu aplikace.

**Animate** - zvýrazňuje každou položku popisu, jak je prováděna.

**Slow Animate** - jako Animate, ale s menší rychlostí, kterou lze nastavit výběrem **File - Preferences**.

**No Animate** - zastavuje režim Animate, resp. Slow Animate.

## **Menu Window**

**No Auto Tiling** - otevírané pohledy nejsou uspořádány.

**Auto Tile Horizontally** - otevírané pohledy jsou zobrazovány nad sebou.

**Auto Tile Vertically** - otevírané pohledy jsou zobrazovány vedle sebe.

**Tile Vertically** - zobrazí pohledy vedle sebe.

**Tile Horizontally** - zobrazí pohledy nad sebou.

**Cascade** - zobrazí pohledy do kaskády.

**Arrange Icons** - uspořádá ikony minimalizovaných pohledů.

**Close All** - uzavře všechny otevřené pohledy kromě okna Main.

## **Menu Help**

**Index** - zobrazí rejstřík.

**Functions** - nápověda k systémovým, interním a externím funkcím.

**Messages** - nápověda ke zprávám.

**About SQLWindows** - dialogové okno s informací o verzi SQLWindows.

## **2.5 Přidávání a editace grafických objektů rozhraní**

Existují tři způsoby přidávání grafických objektů při definování uživatelského rozhraní:

### **a) výběrem z menu Tools nebo palety nástrojů.**

Při tomto způsobu vkládání nových objektů lze použít tlačítka palety nástrojů nebo odpovídajících položek menu **Tools**. U takto umístěného objektu lze měnit myší velikost objektu a editovat titulky tlačítek. Pokud je přidávaným objektem okno tabulky nebo synovská tabulka, přidávají se sloupce automaticky po umístění kurzoru do levého horního rohu tabulky (do průsečíku záhlaví sloupců a řádek)-tvar kurzoru indikuje funkci přidávání sloupce. Umístění se provede stiskem tlačítka myši.

### **b) využitím dialogového okna Outline Options.**

Po volbě **View-Outline Options** se zobrazí dialogové okno, které pro zvolenou položku, resp.úroveň položek v textovém popisu nabízí položky, které mohou být vloženy do popisu na téže a nižší úrovni.

### **c) textovým zápisem.**

Novou položku lze vložit do textového popisu vložení nové řádky (**Outline-Insert Line**) a zápisem textu.

Poznámky: Způsob ad a) neumožňuje vkládat menu a jeho položky. K tomu je nutno použít některý ze zbývajících způsobů.

Pro definici pojmenovaných menu existuje v popisu samostatná sekce **Named Menu**, použití se uvádí v sekci **Menu**.

## **2.6 Přizpůsobení (customizing) grafických položek**

Tato činnost se provádí prostřednictvím nástroje označovaného jako **Customizer**. Lze ho vyvolat několika způsoby:

a) Je-li kurzor myši na grafickém objektu (má tvar čtveřice šipek -Window Grabber), lze Customizer spustit dvojným stisknutím tlačítka myši.

b) Pro sloupec tabulky se Customizer otevírá umístěním kurzoru na záhlaví sloupce a dvojným stiskem tlačítka.

c) Pro okno nejvyšší úrovně se otevře Customizer umístěním kurzoru ve tvaru čtveřice šipek na plochu okna, na které není žádný objekt, a dvojným stisknutím tlačítka myši.

d) Z textového popisu se spouští Customizer umístěním kurzoru na ikonu a dvojným stiskem levého tlačítka nebo jedním pravého.

Ve všech případech se objeví na obrazovce panel, který umožňuje nastavit hodnoty atributů (parametrů) vybraného objektu. Atributy závisí na druhu objektu a jsou členěny do několika částí.

Okna na nejvyšší úrovni mohou mít pozadí bílé (**Standard**) nebo se vzorkem (**Etched**). Typ lze nastavit v parametru **Display Style** pro okno nebo **Window Defaults** v globálních deklaracích (sekce **Global Declarations**).

## 2.7 Kopírování

Přidávat objekty kopírováním lze použitím **duplikátoru** z palety nástrojů nebo menu **Tools-Draw**. Jinou možností je použití editačních funkcí **Copy** a **Paste**. Kopírovat lze grafické objekty i textový popis. Při použití duplikátoru získávají nové objekty nová jednoznačná jména, zatímco použitím schránky se vytvoří objekt se stejným jménem a je nutné ho změnit.

## 2.8 Jména objektů

Objekty jsou v SQLWindows pojmenovány. Okna nejvyšší úrovně se také označují jako šablony (template), protože může existovat několik instancí vytvořených podle stejné "šablony". Jejich jména se potom označují jako jména šablony (template name). Pro jména objektů jsou zavedeny pro lepší orientaci konvence, které zavádějí prefixy pro jména podle typu objektu. Je vhodné tyto konvence dodržovat.

Objekt Prefix  
datové pole df  
víceřádkový text ml  
tlačítko pb  
rádiové tlačítko rb  
tlačítko volby ob  
kontrolní tlačítko cb  
seznam lb  
kombo cmb  
značky pro rolování sb  
sloupec tabulky col  
obrázek pic  
zákaznické řízení cc  
okno formuláře frm  
okno tabulky tbl  
dialogové okno dlg  
MDI okno mdi  
MDI synovské okno mdifrm  
MDI synovská tabulka mditbl

## 2.9 Režimy činnosti nástroje Outliner

Při práci s textovou reprezentací aplikace se může Outliner nacházet v jednom ze dvou režimů - **editace** nebo **návrh**. V prvním režimu je možné editovat části textu (např. jména), ve druhém režimu lze pracovat s celými položkami (kopírovat, přesouvat apod.). Některé příkazy z menu lze proto používat pouze v režimu návrhu, případně poněkud odlišně v závislosti na režimu. Režim je

indikován tvarem kurzoru, který je při editaci tvořen svislou čárkou, při návrhu šipkou.

Přechod z režimu editace do režimu návrhu se provádí klávesami Esc nebo Enter. Po stisku Tab v režimu návrhu Outliner hledá editovatelnou část řádku vybrané položky (nejvyšší úrovně vybrané položky) a najede-li ji, přepne se do režimu editace.

Význam některých kláves v režimu návrhu:

Home - vybere rodičovskou položku aktuální položky.

End - expanduje aktuální položku a vybere poslední její podsekcí (nejnižší úrovně).

Šipka nahoru, dolů - vybere předchozí, resp. následující položku téže úrovně.  
Šipka doleva, doprava - vybere předchozí položku, resp. následující položku (i jiné úrovně).  
Význam některých kláves v režimu editace:  
Šipky nahoru, dolů - mění možné hodnoty vybrané editovatelné položky.  
Enter, Esc - ukončuje režim editace. Sekce, která byla editována, je vybrána.  
Ctrl+Enter - nový řádek bez vytvoření nové sekce (normálně je sekce vždy na jednom řádku).

### 3 Jazyk SAL

Jazyk SAL slouží ke specifikaci chování objektů uživatelského rozhraní navrženého postupem popsáním v předchozí části. Jde o **objektově orientovaný jazyk**, který poskytuje prostředky pro definice tříd, podporuje dědičnost apod., ale lze ho použít i bez těchto prostředků.

Jazyk SAL je **řízen událostmi** - všechny operace v rámci aplikace jsou chápány jako události objektů. Objekty jsou především objekty uživatelského rozhraní. Každý objekt má své jméno, může mít strukturu tvořenou objekty nižší úrovně a lokálními proměnnými a dále obsahuje popis reakcí na přijaté **zprávy** (sekce Message Actions).

Zprávou se rozumí signál zasláný nějakému objektu, který může spustit v tomto objektu nějakou proceduru.

Poznámky: 1. Ne každý objekt rozhraní je objektem jazyka SAL v tomto smyslu (např. text na pozadí).

2. V systému je definováno asi 40 zpráv, programátor může definovat vlastní zprávy.

Příkazy jazyka SAL mohou být v těchto sekcích popisu aplikace:

- Message Actions.
- Actions (ve funkcích) - nereagují na zprávy, mechanismus spuštění je jiný.
- Menu Actions (u položek menu) - stejné jako Actions.
- Application Actions (v globálních deklaracích).

Př) Uvažujme dialogové okno s tlačítkem pbOK a předpokládejme, že chceme zablokovat stisk tohoto tlačítka, dokud není zadána hodnota v jistém datovém poli. Předpokládejme, že po zadání této hodnoty kód na jiném místě aplikace odblokuje tlačítko OK. Po stisku tohoto tlačítka se má dialog ukončit. V sekci Message Actions tlačítka pbOK by byly příkazy:

```
On SAM_Create  
Call SalDisableWindow(pbOK)  
On SAM_Click  
Set dfData=sString  
Call SalEndDialog(dlgDialog,0)
```

Zpráva SAM\_Create je zaslána, když je vytvořen grafický objekt, v našem případě tlačítko pbOK. Na tuto zprávu reaguje tlačítko tak, že se zablokuje (funkce SalDisableWindow).

Druhá zpráva SAM\_Click je zaslána tlačítku pbOK, je-li toto tlačítko stisknuto. Objekt na tuto zprávu reaguje tak, že nastaví hodnotu datového pole dfData a ukončí dialog.

**Jazyk SAL rozlišuje malá a velká písmena.**

#### 3.1 Datové typy

V jazyce SAL jsou definovány tyto datové typy:

**Boolean** - s konstantami TRUE (1) a FALSE (2).

**Date/Time** - lze použít libovolný běžný formát, implicitně YYYY-MM-DD-HH.MM.SS.MSMSMS.

**Number** - číselná data s přesností až 18 číslic, je-li použit pro typ DECIMAL serveru SQLBase může mít maximálně 15 cifer.

**String** - znakový řetězec délky až 254 B. Znakové literály se zapisují v apostrofech, slouží kvyznačení změny významu (escape char).

**Long String** - pro binární data nebo datový typ LONG VARCHAR serveru SQLBase. Další typy poskytují **identifikátory instancí oken**, souborů a spojení s databází. Takový identifikátor se nazývá identifikátor instance (handle). V jazyce SAL existují tři typy takových identifikátorů:

**Window Handle** - identifikace instance okna. Při běhu má každá instance okna jednoznačnou hodnotu tohoto identifikátoru.

**File Handle** - identifikace otevřeného souboru.

**SQL Handle** - identifikace spojení s databází. Každý přístup k databázi vyžaduje zadání hodnoty tohoto identifikátoru.

Další skupinu datových typů tvoří **ukazatelové varianty** uvedených typů (kromě Long string), které se používají k výstupu hodnot z funkce, tj. pro volání odkazem.

**Receive Boolean**

:  
:

**Receive SQL handle**

### 3.2 Proměnné

Proměnné mohou být v jazyce SAL:

a) **globální** - jsou globální v aplikaci, deklarované v sekci Global Declarations - Variables.

b) **lokální** - deklarované v okně nejvyšší úrovně, synovském okně tabulky (sekce Window Variables, Window Parameters) nebo funkci (sekce Local Variables).

c) **statické** - varianta lokálních proměnných u funkcí (sekce Static Variables). Statické proměnné si uchovávají hodnotu.

Jméno synovského okna označuje zároveň proměnnou nesoucí hodnotu (kromě seznamu a tabulky).

Př) Příkaz

Set cbCheckBox=TRUE

nastaví nastaví kontrolní okno.

Proměnné použité v příkazech SQL (tzv. vázané (bind) proměnné) jsou uvozeny dvojtečkou.

Př) Call SqlImmediate('select room from guest where name=:dfGuest into dfRoom)

Pro lepší orientaci v textovém popisu je vhodné dodržovat určité konvence v pojmenování proměnných. Podle těchto konvencí je začíná jméno několikapísmenným prefixem udávajícím typ:

b Boolean

s nebo str String

n Number

dt Data/Time

ls Long String

fh File Handle

hSql Sql Handle

hWnd Window Handle

V případě globální proměnné písmeno G indikuje tuto skutečnost. Je-li proměnná pole, uvádí se písmeno a.

Př) sGaStrStack je jméno globálního pole znakových řetězců.

### 3.3 Kvalifikované odkazy

Kvalifikované odkazy (externí odkazy) se používají pro odkazy mimo rozsah viditelnosti a v případě nejednoznačnosti. Kvalifikované jméno je tvořeno několika kvalifikátory oddělenými tečkou.

Kvalifikovaný odkaz může být tzv. **objektově kvalifikovaný**, kdy se pro kvalifikaci použije jméno objektu, např. frmMain.dfPocet.

Může však nastat situace, kdy existuje několik instancí daného objektu (např. oken frmMain). V takovém případě je nutné k jednoznačné identifikaci použít tzv. **plně kvalifikovaný odkaz**, který je tvořen navíc identifikátorem instance, např.

hWndHandle.frmMain.dfPocet

Poznámky: 1.) Rozsah viditelnosti symbolu je definován běžným způsobem, tj. zahrnuje objekt, v němž je symbol dekarován a objekty obsažené v definujícím, zanořené do ibovolné úrovně.

2.) Překladač je schopen v případě jednoznačnosti řešit i odkazy mimo rozsah viditelnosti, této možnosti je však vhodné se vyvarovat.

3.) Symboly synovského okna tabulky musí být z rodičovského okna referovány vždy kvalifikovaně, např. frmMain.tblChild.col1.

4.) V sekci aplikace Design-time Settings lze nastavit, že externí odkazy musí být vždy kvalifikovány a lze zakázat vícenásobné instance téhož okna na nejvyšší úrovni.

### 3.4 Speciální klíčová slova

V jazyce SAL jsou předdefinovány čtyři speciální identifikátory instancí okna:

**hWndMDI** - identifikuje aktuální MDI okno.

**hWndForm** - identifikuje aktuální okno nejvyšší úrovně.

**hWndItem** - identifikuje aktuální synovské okno, existuje-li.

**hWndNULL** - konstanta reprezentující nedefinovanou hodnotu identifikátoru instance okna. Používá se např. v případech, kdy je parametrem funkce identifikátor instance okna a v daném případě není jeho hodnota definována.

Aktuálním oknem zde rozumíme okno, které v daném okamžiku reaguje na nějakou zprávu.

### 3.5 Získání identifikátoru instance okna

Existují dvě možnosti, jak získat hodnotu identifikátoru:

a) při vytvoření okna vrací funkce hodnotu identifikátoru.

Set hWndFrm1 = Call SalCreateWindow(frmForm1,hWndNULL)

b) identifikátor aktivního okna je v předdefinované proměně hWndFrm, resp. hWndMDI, resp. hWndItem.

On SAM\_Create

Set hWndFrm1 = hWndForm

resp. pro synovské okno

On SAM\_Create

Set hWndDf1 = hWndItem

Pro získání identifikátoru instance rodičovského okna nějakého okna synovského lze použít funkci SalParentWindow:

hWndParent = Call SalParentWindow(hWndWindow)

### 3.6 Pole

V jazyce SAL lze používat i pole. Deklarace má tvar:

typ: jméno\_proměnné [počet\_prvků|od:do|\*]

Poznámky: 1.) Lze deklarovat pouze jednodimenzionální pole.  
2.) Symbol \* značí dynamický počet prvků pole.  
3.) Počet prvků polea meze lze zjistit funkcí SalArrayBounds, nový rozsah indexů nastavuje funkce SalSetArrayBounds.

### 3.7 Konstanty

Konstanty se deklarují v sekci Global Declarations-Constants v podsekci User. Zde se typicky deklarují např.konstanty pro programátorem definované zprávy.

### 3.8 Operátory

V jazyce SAL jsou definovány tyto operátory:

Číselné: +, -, \*, /

Unární: -

Relační: =, !=, >, <, >=, <=

Boleovské: AND, OR, NOT

Bitové: & - log.součin, | - log.součet

Řetězcové: || - zřetězení

Výrazem je syntaktická jednotka, která dává hodnotu.

### 3.9 Příkazy

Příkazy jazyka SAL se používají v těchto sekcích:

- Message Actions
- Function Actions
- Application Actions
- Menu Actions

V jazyce SAL existuje 10 příkazů:

#### Příkaz On

**On** zpráva  
příkazy

Příkaz zachytí danou zprávu a provede příkazy uvedené na úrovni nižší. Používá se v sekci Message Actions.

Př) On SAM\_Click

Call SalDestroyWindow(hWndForm)

#### Příkaz Set

**Set** proměnná = výraz

Provede dosazení hodnotu výrazu proměnné (musí být stejného typu).

Př) Set nIndex=0

#### Příkaz Call

**Call** jm\_funkce (parametry)

Spustí funkci bez uložení návratové hodnoty.

Př) Call SqlImmediate('rollback')

#### Příkazy If, Else

**If** výraz1

příkazy1

[**Else If** výraz2

příkazy2

:

:

]

[Else

příkazy3

### **Příkazy Loop a Break**

**Loop** [jméno smyčky]

:

**Break** [jméno smyčky]

Jméno smyčky je užitečné především v těch případech, kdy jsou příkazy Loop zanořeny a u příkazu Break je třeba specifikovat, který příkaz cyklu se má ukončit (implicitně na nejbližší vyšší úrovni).

### **Příkaz Select**

**Select** výraz

**Case** konstanta1

příkazy1

**Break**

:

:

[Default

příkazy2

]

Sémantika příkazu Select je jako v jazyce C, tj. pokud není uveden příkaz Break, provádí se další příkazy. Klauzule Default může být uvedena kdekoli na úrovni klauzulí Case.

### **Příkaz Return**

**Return** výraz

Ukončuje provádění posloupnosti příkazů (v rámci příkazu On nebo položky menu nebo funkce) a vrací hodnotu volající funkci nebo SQLWindows (v případě příkazu On a položky menu).

### **Příkaz While**

**While** výraz

příkaz

Jde o klasický příkaz While.

### **Příkaz WhenSqlError**

**WhenSqlError**

Určuje způsob zpracování chyb při databázových operacích. Implicitně se objevuje dialogové okno se standardním hlášením. Pokud potřebujeme jiné ošetření chyb, použijeme tento příkaz.

## **3.10 Poznámky**

Jako poznámky se chápou řádky textového popisu začínající vykřičníkem a mezerou (po ikoně sekce). Pro zakomentování lze použít položky menu **Outline-Comment Items**, resp. **Uncomment** pro odkomentování.

### 3.11 Funkce

V jazyce SAL existují 3 obecné kategorie funkcí:

- a) systémové funkce SQL - používají prefix Sal a Sql, jsou globálně dostupné.
- b) externí funkce definované mimo SQLWindows,
- c) uživatelské funkce definované programátorem v aplikaci.

Uživatelské funkce lze rozdělit do tří skupin:

- funkce okna - jsou deklarovány lokálně v okně (v sekci Functions) nejvyšší úroveň nebo synovského okna tabulky, tj. mohou být volány bez kvalifikace i ze všech synovských oken.
- interní funkce - definovány v sekci Global Declarations, jsou globálně dostupné,
- funkce třídy

### 3.12 Prvky zprávy wParam a lParam

Jde o předdefinované parametry zpráv, které využívají některé systémové funkce a mohou být využity i uživatelskými funkcemi.

**wParam** je slovo o délce 16 bitů, **lParam** je dlouhé slovo délky 32 bitů.

V systému SQLWindows jsou předdefinovány proměnné stejného jména, které uchovávají hodnoty odpovídajících parametrů právě zpracovávané zprávy. Tj. v příkaze On lze používat přímo tyto proměnné.

### 3.13 Použití Outline Options

Nastavením **View-Outline Options** se zobrazí dialogové okno, které v závislosti na právě aktuální sekci textového popisu aplikace nabízí příkazy jazyka SAL, konstanty zpráv, funkce a konstanty, které lze v daném místě použít.

Poznámky: 1.) V dialogovém okně lze začít psát jméno hledané položky do datového pole a systém automaticky vyhledá položku v seznamu.

2.) K většině položek existuje kontextová nápověda dostupná přes klávesu F1.

3.) V menu File-Preferences lze nastavit, zda se mají nabízet i globální prvky (List Globals).

### 3.14 Spuštění aplikace, překlad, ladění

K překladu, spuštění a ladění se používají položky menu **Run**.

Běh se zastaví před řádkou s bodem zastavení. V dialogovém okně, které se objeví po zastavení, jsou tlačítka

**Step** - provede jednu další proveditelnou položku textové reprezentace.

**Step Over** - jako Step, ale přeskočí zvýrazněnou položku.

**Continue** - pokračuje do dalšího bodu zastavení.

**Abort** - ukončí aplikaci.

**Close** - uzavře dialogové okno.

**Eval** - vyhodnotí výraz zapsaný v poli výrazu nebo provede příkaz Set. Hodnota výrazu se zobrazí v poli výsledku.

**Watch-Msgs** - ukáže seznam zaslaných zpráv (jaká, pro které okno, wParam, lParam).

**Vars** - seznam globálních proměnných s hodnotami, lze zadat seznam proměnných, které se mají monitorovat.

**Stack** - zobrazí se okno s informací o tom, které objekty byly zpracovány od posledního bodu zastavení.

## 4 Programování aplikace v jazyce SAL

### 4.1 Vytváření a rušení okna na nejvyšší úrovni

Okna MDI, formuláře a okna tabulky mohou být vytvořeny při běhu dvojitým způsobem:

- a) automaticky při startu,
- b) programově voláním funkce SalCreateWindow().

Dialogové okno lze vytvářet pouze programově voláním funkce SalCreatWindow pro nerezimové okno, resp.SalModalDialog() pro režimové.

### **Vytvoření okna formuláře**

Má-li být okno vytvořeno při spuštění, nastaví se použitím nástroje Customizer hodnota atributu **Automatically Create** na Yes.

Pro programové vytvoření okna je k dispozici funkce:

**hWndNew = SalCreateWindow(template, hWndOwner, Par1, Par2,...)**

template - jméno okna (šablony).

hWndOwner - identifikátor instance okna nebo jméno okna nejvyšší úrovně, které je vlastníkem vytvářeného okna. Nechceme-li specifikovat vlastníka, použijeme hWndNULL.

Par1, Par2, ... - předávané parametry deklarované v sekci Window Parameters, korespondence poziční. Mohou být i ukazatelového typu, pak ale nesmí být skutečným parametrem jméno datového pole.

hWndNew - identifikace instance vytvořeného okna.

### **Vytvoření okna formuláře se zablokováním tvorby násobných instancí**

V praxi zpravidla potřebujeme, aby se po vytvoření okna formuláře nedaly vytvářet další instance téhož okna, dokud není vytvořené okno uzavřeno. V této části je ukázáno, jak lze zajistit existenci nejvýše jedné instance daného okna při vytváření z menu a stiskem tlačítka.

### **Vytvoření z položky menu**

Okno se vytváří voláním funkce SalCreateWindow v sekci Actions odpovídající položky menu. Zabránění násobného výskytu lze v tomto případě dosáhnout zablokováním položky menu, pokud nějaká instance již existuje testem identifikátoru okna na hWndNULL.

Postup by mohl vypadat takto:

1. Nastav v attributech okna Automatically Create na No.

2. Deklaruj proměnnou v globálních deklaracích

Window Handle: hWndXXX

3. U dané položky menu v sekci Menu Actions vytvoř instanci okna

Set hWndXXX = SalCreateWindow(frmXXX, hWndNULL)

4. Zadej podmínku pro zablokování položky v sekci Enable when dané položky menu:

hWndXXX = hWndNULL

4. Po uzavření okna nastav v sekci Message Actions daného oknaformuláře hodnotu identifikátoru na hWndNULL:

On SAM\_Destroy

Set hWndXXX = hWndNULL

### **Vytvoření stiskem tlačítka**

Postup je obdobný jako u vytváření z menu, odlišný je způsob zablokování tlačítka:

1. Nastav v attributech okna Automatically Create na No.

2. Okno formuláře se vytváří při stisku tlačítka, proto bude v sekci Message Actions tlačítka příkaz na vytvoření okna. Pokud nepotřebujeme identifikátor instance okna uložit, stačí příkaz Set. Zablokování se provede voláním funkce SalDisableWindow():

On SAM\_Click

Call SalCreateWindow(frmXXX, hWndNULL)

Call SalDisableWindow(hWndItem)

3. Odblokuj tlačítko voláním funkce SalEnableWindow() v sekci Message Actions okna formuláře. V tomto případě je nutno použít externí odkaz (v následujícím příkladě předpokládáme jméno tlačítka pbTTT, které je v okně formuláře frmMain). Tvar příkazu by byl:

```
On SAM_Destroy  
Call SalEnableWindow(frmMain.pbTTT)
```

Pokud bychom se chtěli vyhnout externímu odkazu(při násobných instancích), mohli bychom využít možnosti předat parametry okna formuláře funkcí SalCreateWindow a možnosti zasílat uživatelské zprávy. Prvé možnosti využijeme k předání identifikace instance okna, která okno formuláře vytvořila, druhé potom k zaslání zprávy, prostřednictvím které tlačítko odblokujeme. Postup by byl následující:

1. V sekci globálních uživatelských konstant definuj konstantu pro uživatelskou zprávu:

```
Number: AM_Enable = SAM_User+1
```

SAM\_User udává nejvyšší konstantu použitou systémovými zprávami SQLWindows.

2. V okně, které se má vytvořit, deklaruuj parametr okna:

```
Window handle: hWndCreator
```

3. V sekci Message Actions tlačítka vytvoř okno formuláře a předej jako hodnotu parametru hWndCreator identifikaci tlačítka, které okno vytvořilo a zablokuj tlačítko:

```
On SAM_Click
```

```
Call SalCreateWindow(frmXXX, hWndNULL, hWndItem)
```

```
Call SalDisableWindow(hWndItem)
```

4. Při uzavření okna formuláře zašli uživatelskou zprávu AM\_Enable tlačítku (s identifikátorem hWndCreator). Způsob zasílání uživatelských zpráv bude vysvětlen později.

5. V sekci Message Actions tlačítka naprogramuj odblokování při zprávě AM\_Enable:

```
On EM_Enable
```

```
Call SalEnableWindow(hWndItem)
```

Uvedený postup je elegantnější také v tom, že jak vytvořené okno formuláře, tak tlačítko obsahují kód, který se týká jich samotných. Výjimkou je pouze vytvoření okna a zaslání zprávy.

### **Zrušení okna formuláře, tabulky a bezrežimového okna**

Okno lze uzavřít běžným způsobem dvojitým stiskem tlačítka myši na ikoně v levém horním rohu okna (příkaz Close) nebo programátorsky. Ve druhém případě se používá funkce:

#### **SalDestroyWindow(hWndForm)**

hWndForm - jméno nebo identifikátor výskytu okna.

V obou případech je při uzavření okna zaslána uzavíranému oknu a všem synovským oknům zpráva **SAM\_Destroy**. Při uzavření okna příkazem Close se posílá zpráva **SAM\_Close** ještě před uzavřením a čeká se na vrácenou booleovskou hodnotu. Toho lze využít např.k dotazu před vlastním zrušením. Např.před uzavřením hlavního okna by v jeho sekci Message Actions mohl být příkaz:

```
On SAM_Close
```

```
Set nResponse = SalMessageBox('Chcete ukončit aplikaci?', 'Aplikace',
```

```
MB_YesNo|MB_IconAsterik)
```

```
IfnResponse = IDNO
```

```
Return FALSE
```

```
On SAM_Destroy
```

```
Call SalQuit()
```

## Vytvoření a zrušení režimového dialogového okna

K vytvoření se používá funkce se syntaxí:

**nReturn = SalModalDialog(template, hWndOwner, Par1, Par2, ...)**

Význam parametrů je stejný jako u funkce SalCreateWindow.

nReturn - návratová hodnota, která se předává funkci při rušení dialogového okna.

K rušení dialogového okna se používá funkce:

**bOk = SalEndDialog(hWndModal, nReturn)**

hWndModal - jméno nebo identifikátor výskytu dialogového okna.

nReturn - hodnota, která chcete, aby byla návratovou hodnotou funkce SalModalDialog().

Vzhledem ke speciálnímu režimu dialogového okna jsou příkazy následující za voláním SalModalDialog() provedeny až po uzavření dialogového okna. Funkce SalModalDialog() potom vrací právě hodnotu, kterou dostává funkce SalEndDialog() jako parametr (nReturn). Taho lze využít např. k předání informace o tom, které tlačítko dialogového okna bylo stisknuto.

Př) Předpokládejme, že nějaké okno formuláře obsahuje příkaz:

```
If TRUE=SalModalDialog(dlgX,hWndForm)
```

```
Call ....
```

a že otevřené dialogové okno obsahuje tlačítka pbOk a pbCancel. V sekci Message Actions tlačítka pbOK bypotom mohl být příkaz:

```
On SAM_Click
```

```
Call SalEndDialog(hWndForm, TRUE)
```

a u tlačítka pbCancel příkaz:

```
On SAM_Click
```

```
Call SalEndDialog(hWndForm, FALSE)
```

Je-li vytvořeno režimové okno, lze události pro objekty mimo dialogové okno vytvářet pouze zasíláním uživatelských zpráv.

## Odkazy z dialogového okna na formulář

Režimové dialogové okno slouží typicky k nastavení nebo výběru nějakých hodnot, které musí být k dispozici, aby mohl objekt, který dialog vyvolal, pokračovat v činnosti. Takové hodnoty je potom třeba předat volajícímu objektu. Lze použít dva způsoby:

a) externí odkaz - použití příkazu Set s externím odkazem.

b) použití parametrů okna - v dialogovém okně se definují ukazatelové parametry, jejichž hodnoty se nastaví při volání funkce SalModalDialog() a prostřednictvím nich se předají požadované hodnoty (příkazem Set) před uzavřením dialogu.

Př) Předpokládejme, že potřebujeme získat z dialogu řetězec znaků sXXX. V dialogovém okně definujeme parametr:

```
Window Parameters
```

```
Receive String: sXXXDialog
```

Předpokládejme, že dialogové okno obsahuje dvě tlačítka - pbOK a pbCancel. V sekcích Message Actions těchto tlačítek by byly příkazy:

```
On SAM_Click
```

```
Set sXXXDialog = sXXX
```

```
Call SalEndDialog(hWndItem, TRUE)
```

```
resp.
```

```
On SAM_Click
```

```
Call SalEndDialog(hWndItem, FALSE)
```

a vytvoření dialogového okna by bylo provedeno příkazem:

```
If TRUE=SalModalDialog(dlgDialog,hWndForm,sXXXForm)
```

```
Set dfXXX=sXXXForm
```

Proměnná sXXXForm je nutná, protože ukazatelovým parametrem nemůže být datové pole. Využití parametrů je programátorsky čistější. Zároveň je to způsob použitelný, je-li dialog volán z různých oken.

## 4.2 Definování funkcí

Funkce mohou být interní (globální v aplikaci), funkce okna (lokální v okně), funkce třídy a externí funkce (definované vně aplikace v DDL). Externími funkcemi se zde zabývat nebudeme.

### Interní funkce a funkce okna

Definují se v podsekcí Internal Functions globálních deklarácí, resp. v sekci Functions okna.

**Sekce Function** sestává z několika podsekcí:

**Description** - slovní popis.

**Returns** - specifikace typu návratové hodnoty (včetně dvojtečky za jménem typu).

Specifikace návratové hodnoty je nepovinná.

**Parameters** - deklarace parametrů, stejným způsobem jako poměnných.

**Static Variables** - statické proměnné inicializované v době překladu.

**Local Variables** - lokální proměnné, které nejsou statické.

**Actions** - kód funkce.

Př) Předpokládejme, že chceme vytvořit funkci, která vynuluje všechna datová pole formuláře. Aby funkce nebyla závislá na konkrétním formuláři, použijeme funkci

**SalGetFirstChild()**, která najde prvé synovské okno daného typu

pro daný formulář. Její syntax je:

```
hWndChild=SalGetFirstChild(hWnd,nTypeMask)
```

hWnd - identifikátor okna nejvyšší úrovně, jehož synovská okna hledáme.

nTypeMask - konstanta určující typ okna. Konstanty jsou předdefinovány. Např. konstanta pro datové pole je TYPE\_DataField. Výsledek může být získán i operací bitového součtu.

Další synovská okna budeme hledat použitím funkce **SalGetNextChild()** s podobnou syntaxí:

```
hWndChild=SalGetNextChild(hWndChild,nTypeMask)
```

Funkce by mohla mít tvar:

Function: ClearFields

Description: Nuluje všechna datová pole specifikovaného formuláře

Returns

Parameters

Window Handle: hWndCurrent

Static Variables

Local Variables

Window Handle: hWndChild

Actions

```
Set hWndChild=SalGetFirstChild(hWndCurrent,  
TYPE_DataField)
```

```
While NOT hWndChild=hWndNULL
```

```
Call SalClearField(hWndChild)
```

```
hWndChild=SalGetNextChild(hWndChild,TYPE_DataField)
```

Mezi další systémové funkce, které lze využít u uživatelských funkcí, které pracují se synovskými okny a neměly by být závislé na konkrétním okně nejvyšší úrovně, patří:

**SalParentWindow()** - vrací identifikátor rodičovského okna, resp. vlastníka.

**SalGetType()** - vrací hodnotu typu okna.

**SalGetItemName()** - ukládá do parametru jméno okna.

**SalPostMsg()**, **SalSendMsg()**, **SalSendMsgToChildren()** - zaslání uživatelské zprávy definované programátorem.

### 4.3 Okna MDI

MDI(Multiple Document Interface) okna jsou okna schopná minimalizace a maximalizace, která mohou obsahovat jako synovská okna formulářů a tabulek (tato okna nazýváme **synovská MDI okna**).

Celá aplikace může být tvořena jedním oknem MDI a synovskými okny MDI nebo může používat MDI okno jako doplněk nezávislých oken nejvyšší úrovně.

MDI okna také jinak pracují s menu. Menu aktivního synovského MDI okna je vždy umístěno v okně rodičovského MDI okna. Navíc existuje zvláštní menu Windows, které zahrnuje automaticky vytvořenou sekci se seznamem vytvořených synovských oken pro výběr aktivního okna. K tomuto účelu slouží předdefinované pojmenované menu **menuMDIWindows** s položkami známými z menu Windows aplikací běžících pod Windows.

Okno MDI lze definovat z palety nástrojů nebo pomocí Outline Options, podobně se definují i synovská okna.

MDI okno se vytváří automaticky po spuštění (je-li nastaveno Automatically Create na Yes) nebo programově. K programovému vytvoření se používá opět funkce **SalCreateWindow()**, u synovských oken se jako vlastník uvádí **hWndMDI** (aktivní okno MDI). Hodnoty **hWndMDI** a **hWndForm** jsou vždy různé.

Okno MDI tedy vytvoříme voláním:

```
Call SalCreateWindow(mdi1,hWndNULL)
```

a synovské okno:

```
Call SalCreateWindow(mdifrm1,hWndMDI)
```

Obecně mohou být násobné výskyty MDI okna i synovských oken. V takovém případě je nutné používat úplnou kvalifikaci jmen nebo lépevyvarovat se jí použitím funkcí okna MDI a synovského okna.

### 4.4 Použití databáze

SQLWindows používá pro styk s databází jazyk SQL. Příkazy jazyka SQL se zadávají jako znakové řetězce funkcím a jsou analyzovány až při běhu aplikace.

V této části budeme uvažovat pouze jednouzivatelskou aplikaci, nebudeme řešit problémy související se souběžným přístupem.

#### Přístup k databázi

Aplikace v SQLWindows může přistupovat k databázi dvojitým způsobem:

##### a) vícekrokový přístup

##### b) jednokrokový přístup

U vícekrokového přístupu je nutno provést tyto akce:

1. zajistit připojení databáze (Connect),
2. provádět databázové operace,
3. uzavřít spojení s databází.

Druhý krok sestává pro každý příkaz jazyka SQL z několika dílčích kroků:

- 2a. přeložení příkazu jazyka SQL,
- 2b. provedení příkazu jazyka SQL,
- 2c. výběr dat z databáze a přiřazení (pro příkaz SELECT)

U jednokrokového přístupu lze provést databázovou operaci voláním jediné funkce, která v sobě zahrnuje provedení všech kroků vícekrokového přístupu. Tento způsob je jednodušší, ale má některá omezení, především u příkazu SELECT lze vybrat a přiřadit pouze jeden řádek tabulky výsledku dotazu.

## Vícekrokový přístup

K **připojení databáze** slouží funkce

**bRes=SqlConnection(hSql)**

hSql - identifikátor spojení s databází, který funkce nastaví.

bRes - příznak úspěšnosti operace.

Před voláním funkce SqlConnection() je nutno nastavit hodnoty předdefinovaných proměnných, které se používají při kontrole oprávněnosti přístupu k databázi. Jde o tyto proměnné:

**SqlDatabase** - jméno databáze (implicitně DEMO),

**SqlUser** - jméno uživatele (implicitně SYSADM),

**SqlPassword** - heslo (implicitně SYSADM).

Lze vytvořit několik spojení s toutéž databází s různými hodnotami identifikátorů spojení.

K **uzavření spojení** slouží funkce:

**bRes=SqlDisconnect(hSql)**

Je třeba provést uzavření všech vytvořených spojení. Poslední volání funkce SqlDisconnect() pro danou databázi potvrdí změny provedené v databázi (implicitní COMMIT).

K **překlada příkazu SQL** slouží funkce:

**bOk=SqlPrepare(hSql,příkaz\_SQL)**

hSql - identifikátor spojení, které již bylo vytvořeno,

příkaz\_SQL - znakový řetězec příkazu jazyka SQL,

bOk - TRUE, je-li operace úspěšná.

K **provedení příkazu** (již přeloženého) slouží funkce:

**bOk=SqlExecute(hSql)**

**Přeložení a následné provedení příkazu** provádí funkce:

**bOk=SqlPrepareAndExecute(hSql,příkaz\_SQL)**

V příkazu SQL mohou být použity vázané proměnné (uvozeny dvojtečkou). Případné konverze podle typu proměnné jsou zajištěny automaticky.

## Dotazy (příkaz SELECT)

Provedením příkazu SELECT se na databázové serveru vytvoří tabulka výsledku, z níž však aplikace vybírá jeden řádek po druhém. Má k dispozici tyto funkce:

**SqlFetchNext(hSql,nFetch),**

**SqlFetchPrevious(hSql,nFetch),**

**SqlFetchRow(hSql,nRow)**

hSql - identifikátor spojení s databází,

nFetch - hodnota nastavovaná funkcí. Existují předdefinované konstanty: FETCH\_Ok - úspěšný výběr,

FETCH\_EOF - již nejsou řádky,

FETCH\_Delete - řádek byl zrušen od posledního výběru,

Fetch\_Update - řádek byl změněn od posledního výběru,

bOk - TRUE, jsou-li další řádky, jinak FALSE.

Př) Předpokládejme formulář s tlačítkem pbNext pro výběr dalšího řádku tabulky získané nějakým dotazem. V sekci Message Actions

tohoto tlačítka by potom mohl být příkaz:

On SAM\_Click

If NOT SqlFetchNext(hSql, nFetch)

Call SalDisableWindow(hWndItem)

## Jednokrokový přístup

K připojení databáze, překlada, provedení příkazu a uzavření spojení slouží funkce:

### **bOk=SqlImmediate(příkaz\_SQL)**

příkaz\_SQL - znakový řetězec příkazu jazyka SQL,

bOk - TRUE, je-li operace úspěšná.

Je-příkazem SQL příkaz SELECT, je výsledkem vždy pouze jeden řádek tabulky, který je dosazen do vázaných proměnných příkazu ve formuli INTO.

### **Potvrzení změn**

Pro potvrzení změn v databázi a jejich trvalé uložení slouží příkaz SQL COMMIT, pro návrat k předchozímu stavu příkaz ROLLBACK.

### **Zpracování chyb SQL**

Chyby SQL se hlásí vždy až při běhu, protože příkazy SQL jsou kompilovány až v době běhu aplikace.

Zpracování chyb může být **implicitní nebo naprogramované**. V prvním případě se zobrazí dialogové okno se standardním hlášením udávajícím číslo chyby, její význam, pozici, kde byla chyba zjištěna a dotazem, zda má být aplikace ukončena.

**Naprogramované zpracování** chyb může mít charakter **globálního** nebo **lokálního zpracování**. V prvním případě se využije skutečnosti, že při chybě SQL zasílá systém zprávu SAM\_SqlError sekci Application Actions.

Zpráva SAM\_SqlError nastavuje hodnoty wParam a lParam. wParam obsahuje hodnotu identifikátoru spojení s databází, u kterého došlo k chybě, a lParam obsahuje číslo chyby a pozici, kde došlo k chybě. SQLWindows poskytuje funkci SqlExtractArgs(), která umožňuje extrahovat potřebné informace. Tato funkce může být použita pouze jako reakce na zprávy SAM\_SqlError. Funkce má tvar:

#### **bOk= SqlExtractArgs(wParam, lParam, hSql, nError, nPos)**

wParam, lParam - parametry zprávy (lze využít odpovídajících proměnných).

hSql - vrácená hodnota identifikátoru spojení s chybou.

nError - vrácená hodnota s číslem chyby.

nPos - vrácená hodnota s pozicí na řádku příkazu.

bOk - příznak úspěšnosti operace.

SQLWindows dále poskytuje funkci pro získání textu chybového hlášení - SqlGetErrorText(). Globální zpracování chyb SQL by mohlo vypadat takto:

On SAM\_SqlError

```
Call SqlExtractArgs(wParam,lParam,hSqlGlobal,nErr,nPos)
```

```
Call SqlGetErrorText(nErr,sErrmsg)
```

```
Sall SalNumberToStr(nErr,0,sNum)
```

```
If nErr= UNKNOWN_PASSWORD Or nErr=UNKNOWN_USER
```

```
If IDOK=SalMessageBox(sNum||'-'||sErrmsg||'. Try again.',
```

```
'XXX', MB_RetryCancel|MB_IconStop)
```

```
Return FALSE
```

```
Else
```

```
Call SalMessageBox(sNum||'-'||sErrmsg||'.Choose OK to
```

```
quit.','XXX', MB_Ok|MB_IconStop)
```

```
Call SalQuit()
```

V uvedeném příkladu je použita návratová hodnota FALSE pro případ, kdy chceme zkusit přihlášení znovu. K tomu jsou dva důvody. Jednak implicitní zpracování chyb je spuštěno v případě, že zpráva SAM\_SqlError nevrací žádnou hodnotu, jednak je tato hodnota předána volané funkci s příkazem SQL a následně použita jako návratová hodnota (bOk).

K **lokálnímu zpracování chyb** SQL slouží příkaz **WhenSQLError**, který se chová podobně jako příkaz On, ale jeho smyslem je pouze lokálně odchytnout chyby SQL na dané úrovni zanoření příkazů.

Pokud se vyskytne chyba SQL, nejprve se hledá příkaz WhenSqlError pro danou úroveň a pokud se nenalezne, posílá se zpráva SAM\_SqlError.

Př) Pokud bychom chtěli lokálně zpracovat chybu přihlášení uživatele, mohli bychom za předpokladu existence tlačítka ukončujícího dialog použít příkaz WhenSqlError v sekci Message actions tohoto tlačítka:

```
On SAM_Click  
WhenSqlError  
:  
:
```

#### 4.5 Přihlašovací blok

Řada aplikací vyžaduje rozlišit různé uživatele, kteří mají různá práva přístupu k databázi. V této části je naznačen způsob naprogramování přihlašovacího bloku, který nastaví hodnoty proměnných SqlDatabase, SqlUser a SqlPassword a provede připojení požadované databáze. Současně je ukázáno lokální zpracování chyb příkazem WhenSqlError.

Otevření přihlašovacího okna lze provést automaticky při spuštění aplikace nebo z menu. V prvním případě lze využít zprávy SAM\_AppStartup, která je poslána sekci Application Actions před otevřením jakéhokoli okna. Příkaz by mohl mít tvar:

```
On SAM_AppStartup  
If TRUE=SalModalDialog(dlgLogin,hWndNULL)  
Call SalCreateWindow(frmMain,hWndNULL)  
Else  
Call SalQuit()
```

Pokud proběhne přihlášení v pořádku, vrací funkce SalModalDialog hodnotu TRUE a otevře se hlavní okno. V opačném případě je aplikace ukončena.

Pokud by bylo přihlášení iniciováno z menu, mohl by být u příslušné položky podobný podmíněný příkaz (If). Spuštění přihlašování z menu má oproti předchozímu případu tu výhodu, že lze provádět nové přihlášení bez ukončení aplikace.

Předpokládáme, že dialogové okno dlgLogin obsahuje datová pole dfUser pro zadání jména uživatele a dfPassword pro zadání hesla. Dále jsou v dialogovém okně tlačítka pbOk a pbCancel pro potvrzení zadaných údajů, resp. ukončení dialogu bez připojení databáze.

Většina kódu dialogového okna bude u tlačítka pbOk. Předpokládáme, že požadujeme, aby jak jméno uživatele, tak heslo byly zadané, a že připojíme databázi s názvem XXX (stejně titulky budeme používat u oken zpráv). Současně zajistíme lokální zpracování chyb SQL, v našem případě chybné přihlášení. U vedené činnosti by mohly být zapsány v jazyce SAL v sekci Message Actions tlačítka pbOk takto:

```
On SAM_Click  
WhenSqlError  
If IDRETRY=SalMessageBox('Chybné přihlášení! Zkuste prosím  
znovu.','XXX',MB_RetryCancel|MB_IconStop)  
Return FALSE  
Else  
Call SalQuit()  
If NOT SalIsNull(dfUser)  
SetSqlUser=dfUser  
Else  
Call SalMessageBox('Zadejte nejprve jméno uživatele.',
```

```
'XXX',MB_Ok|MB_IconStop)
Return TRUE
If NOT SalIsNull(dfPassword)
SetSqlPassword=dfPassword
Else
Call SalMessageBox('Zadejte nejprve heslo.',
'XXX',MB_Ok|MB_IconStop)
Return TRUE
Set SqlDatabase ='XXX'
Set bConnect=SqlConnect(hSql)
If bConnect
Call SalEndDialog(dlgLogin,TRUE)
Příkazy Return jsou použity při ošetření chyb SQL k nastavení návratové hodnoty (v našem
případě SqlConnect), v ostatních případech slouží příkaz Return pouze k ukončení
vyhodnocování a uvedená hodnota není podstatná.
Kód u tlačítka pbCancel by byl jednodušší a zahrnoval by pouze uzavření dialogového okna a
předání návratové hodnoty FALSE pro funkci SalModalDialog():
On SAM_Click
Call SalEndDialog(hWndForm, FALSE)
```

## 4.6 Okno seznamu

Okno seznamu je užitečné v případě, kdy chceme nabídnout uživateli výběr z určité množiny položek, které jsou známy v době návrhu nebo existují např. v databázi.

Mezi atributy, které lze pro seznam nastavit, patří:

**Multiple selection** - povolení/zákaz výběru více než jedné položky seznamu.

**Sorted** - požadavek na setřídění položek.

**Vertical Scroll** - požadavek na přidání sloupku pro vertikální posun (scroll bar), je-li v seznamu více položek, než se zobrazí.

**Edit List Initialization** - požadavek na zobrazení obsahu sekce List Initialization okna seznamu. Takto lze nabízet např. výběr z množiny možností daných již v době návrhu aplikace.

Seznamy by měly být používány pro ne příliš rozsáhlé seznamy, protože celý seznam je v tomto případě uchovávan v paměti (na rozdíl od tabulky - viz dále).

### Přidání položky do seznamu

Jsou k dispozici dvě funkce:

**nRet=SalListAdd(hWndList,sItem)**

Funkce přidá k seznamu jednu položku.

hWndList - jméno nebo identifikátor výskytu okna.

sItem - výraz typu řetězec znaků udávající text položky.

nRet - index přidané položky, LB\_Err nebo LB\_Errspace při chybě.

**bOk=SalListPopulate(hWndList,hSql,příkaz\_SELECT)**

Funkce přidá k seznamu výsledek dotazu. Má-li výsledná tabulka více sloupců, jsou hodnoty odděleny tabelátory. U netextových sloupců dochází ke konverzi.

hSql - identifikátor spojení s databází.

příkaz\_SELECT - příkaz SQL pro výběr z databáze, je-li prázdný ("), je použit dříve přeložený příkaz spojený s identifikátorem hSql.

bOk - příznak úspěšnosti provedení příkazu.

### Získání vybrané položky

K získání vybrané položky je třeba pro okno seznamu zachytit událost SAM\_Click a při jejím výskytu volat dvojici funkcí:

**nIndex=SalListQuerySelection(hWndList)**

**nLength = SalListQueryText(hWndList,nIndex,sString)**

Prvá funkce vrací index vybrané položky seznamu a druhá získá text položky s daným indexem.

hWndList - jméno nebo identifikátor výskytu okna.

nIndex - index vybrané položky seznamu.

sString - vrácený text položky.

Př) Předpokládejme, že chceme použít vybranou položku, jejíž hodnotu přiřadíme proměnné sRoom, k dotazu. Výsledek dotazu uložíme do proměnné mlDescr.

On SAM\_Click

Set nIndex=SalListQuerySelection(lbRooms)

Call SalListQueryText(lbRooms,nIndex,sRoom)

Call SqlImmediate('select description from room where name=:sRoom into :mlDescr')

### Dvojí stisknutí tlačítka myši na seznamu

Standardně představuje dvojí stisknutí tlačítka v okně seznamu výběr položky a uzavření celého dialogového okna, jehož součástí seznam je. V systému SQLWindows první stisk tlačítka posílá zprávu SAM\_Click, kterou lze využít ke zjištění vybrané položky.

Navíc se při dvojitým stisku vyšle zpráva SAM\_DoubleClick, kterou lze využít k akcím spojeným s ukončením dialogu a k ukončení dialogu.

### Další užitečné funkce

bOk=SalListClear(hWndList) - zruší všechny položky seznamu.

nRemaining=SalListDelete(hWndList,nIndex) - zruší položku.

bOk=SalListFiles(...) - vytvoří seznam jmen souborů a adresářů.

bOk=SalListGetMultiSelect(...) - vrací pole indexů vybraných položek.

nIndex=SalListInsert(hWndList,nIndex,sAdd) - vloží položku na danou pozici v seznamu.

bDirectory=SalListQueryFile(...) - vybere jméno souboru.

nSel=SalListQueryMultiCount(hWndList) - vrací počet vybraných položek.

bSel=SalListQueryState(hWndList,nIndex) - zjišťuje, zda je položka na zadané pozici v seznamu vybraná.

bOk=SalListRedraw(hWndList,bRedraw) - povolení/zákaz překreslování seznamu na obrazovce (např. při postupném zařazování do seznamu).

bOk=SalListSetSelect(hWndList,nIndex) - vybere, resp. zruší výběr zadané položky (vhodné např. pro implicitní výběr).

## 4.7 Kombo box

Jde o kombinaci seznamu a datového pole. Podle nastavené hodnoty atributu **Always Show List** bude seznam při otevření okna zobrazen nebo ne. Dále lze nastavit hodnotu atributu **String Type**, který může být String nebo Long String.

Pro práci s oknem typu kombo box se používají stejné funkce SQLWindows jako pro okno seznamu. Navíc jméno okna lze použít pro dosazení hodnoty, resp. čtení hodnoty datového pole combo boxu. Lze tedy psát např.

Set sJmeno = cbUcitele

nebo Set cbUcitele = sJmeno.

Při dosazení hodnoty do okna typu kombo box záleží na tom, zda je okno editovatelné nebo ne (dáno nastavením odpovídajícího atributu). V prvním případě lze dosadit jakoukoli hodnotu a ta je potom přidána k seznamu, pokud tam již neexistuje, ve druhém případě lze přiřadit pouze některou z hodnot v seznamu.

## 4.8 Radiové tlačítko

Jak již bylo uvedeno, radiová tlačítka jsou seskupována do skupin, v nichž pouze jedno tlačítko může být sepnuto. Tuto vlastnost zajišťuje systém automaticky. Při otevření okna se skupinou tlačítek je automaticky nastaveno jako sepnuté prvé tlačítko skupiny (při tomto se ale tlačítku neposílá žádná zvláštní zpráva).

V sekci **Message Actions** budou příkazy pro zprávu **SAM\_Click**, případně **SAM\_Create**. Hodnota objektu je booleovská, proto ji lze použít přímo v příkazu **If**, případně testovat funkcí **SalIsButtonChecked()**.

## 4.9 Řídicí okno

Řídicí okno je také booleovským spínačem a používá se zpravidla k řízení výběru voleb z nějaké nabídky. Na rozdíl od radiového tlačítka může být sepnutý libovolný počet těchto spínačů.

V sekci **Message Actions** zpravidla nebude žádný kód, případně se zpracovává událost **SAM\_Click**. K testování stavu řídicího okna lze použít opět přímo hodnotu okna nebo funkci **SalIsButtonChecked()**.

## 4.10 Posouvací sloupek (scroll bar)

Kromě nástroje pro posun, který je součástí okna seznamu, lze v **SQLWindows** využít tento nástroj i samostatně. Typickým použitím bývá:

- a) grafické vyjádření hodnoty (grafické měřítko),
- b) řízení prohlížení hodnot v určitém rozsahu.

### Použití jako grafické měřítko

K nastavení rozsahu se používá funkce:

**bOk=SalScrollSetRange(hWndsb, nMin, nMax, nLine, nPage)**

**hWndsb** - jméno nebo identifikátor výskytu

**nMin, nMax** - rozsah hodnot,

**nLine, nPage** - krok při stisku tlačítka na šipce posunu (**nLine**) a indikátoru pozice (**nPage**).

Implicitní hodnoty jsou: **nMin=0, nMax=100, nLine=1, nPage=10**.

K nastavení pozice indikátoru slouží funkce:

**bOk=SalScrollSetPosition(hWndsb, nPos)**

**nPos** - hodnota v rozsahu hodnot, kterou má indikátor ukázat.

### Použití k řízení prohlížení

Je třeba zpracovávat zprávu **SAM\_ScrollBar**, posílanou objektu sloupku posunu při změně pozice sloupku (tažením sloupku nebo stiskem tlačítka myši při kurzoru na šipce posunu nebo na sloupku).

Zpráva **SAM\_ScrollBar** využívá parametrů **lParam** a **wParam**. Prvý obsahuje pozici posouvacího sloupku a druhý blíže určuje událost. Jsou předdefinovány konstanty, které lze použít k testování **wParam**, např. **SB\_Bottom** značí posun na maximum rozsahu, **SB\_LineUp** posun o řádek směrem k dolní mezi, **SB\_ThumbPosition** posun sloupku na novou pozici a uvolnění tlačítka.

Uvažujme např., že použijeme posouvací sloupek k nastavení proměnné **nHodnota** na hodnotu danou sloupkem. Zpracování události by mohlo mít tvar:

```
On SAM_ScrollBar
```

```
Set nHodnota=lParam
```

## 4.11 Maskování vstupu, formátování a validace polí

**Vstupní maska** omezuje uživatelský vstup na určitých pozicích vstupního pole. Uplatňuje se bezprostředně při vstupu jednotlivých znaků.

**Formát** upravuje zobrazení dat a neprojevuje se při vstupu dat (s výjimkou formátů Uppercase a Lowercase).

**Validací** se rozumí proces kontroly vstupních dat před přijetím aplikací. SQLWindows automaticky provádí validaci kompatibility s typem datového pole, prostřednictvím kterého je hodnota zadávána. Lze však provádět další kontroly, např. množiny povolených hodnot.

### Vstupní masky

Vstupní masky lze použít u datových polí, sloupců okna tabulky a kombo boxů. Masky jsou definovány v Global Declaration-Formats-Input. Nejjednodušším způsobem zadání masky je zobrazením atributů daného objektu a definování přímo zde (systém automaticky přidá novou masku do globálních deklarácí).

V masce lze použít těchto speciálních symbolů:

X ... libovolný znak, jak je zadán,

! ... libovolný znak, velký,

a ... alfa znak, jak je zadán,

A ... alfa znak, velký,

9 ... číslice 0-9,

n ... alfanumerický znak, jak je zadán,

N ... alfanumerický znak, velký,

všechny další znaky jsou chápány jako konstanty.

Př) 999999/9999 by mohla být maska pro rodné číslo.

Při použití masky je nutné vždy zadat tolik znaků, kolik jich maska definuje. Konstanty ve tvaru nečíselných znaků mohou být použity i pro numerická pole (viz rodné číslo).

Vstupní masky lze nastavovat i programově, ale zpravidla se nastavují výše popsaným způsobem v době návrhu aplikace.

### Formáty

Formáty jsou určeny pouze pro datová pole a sloupce tabulek. Formát se uplatňuje až v okamžiku, kdy kurzor opouští datové pole (s výjimkou formátů Uppercase a Lowercase).

Formát se přiřazuje analogicky jako maska prostřednictvím atributu objektu. V systému existují formáty předdefinované, které jsou pojmenovány symbolicky (např. Uppercase, Currency apod.) a jsou nastavovány v dialogovém okně národního prostředí MS Windows. Druhou skupinu tvoří formáty, vytvořené automaticky systémem nebo definované programátorem ve tvaru obrazu formátu.

Definice formátů jsou uloženy opět v sekci Global Declarations.

Obrazy formátů pro typ date/time používají symboly:

M ... měsíc,

d ... den,

y ... rok,

h ... hodina,

m ... minuta,

s ... sekunda,

mmmmmm ...mikrosekundy,

AMPM ... řetězec AM nebo PM.

: / - , mezera ... oddělovače.

Dny a měsíce mohou být vyjádřeny číselně nebo textovou zkratkou.

Obrazy formátů pro čísla používají symboly:

0 ... znak pro vymezení maximálního počtu cifer s případným doplněním nulami,  
# ... znak pro vymezení maximálního počtu cifer, bez doplnění.  
, ... oddělovač tisíců,  
% ... vyjádření v procentech (x100 a znak %),  
\$ ... znak dolaru na dané pozici, resp. samostatně indikuje  
E+/e+;E-/e- ... vědecká notace,  
; ... oddělovač formátů pro kladné a záporné číslo,  
\_ ... vkládá mezeru odpovídající následujícímu znaku.

Jsou-li data formátována, může se stát, že datové pole neobsáhne všechny zadávané cifry(protože formát může dodávat další znaky). Šířka pole určuje maximální šířku neformátovaných dat. Přesáhne-li šířka po naformátování specifikovanou šířku pole, zobrazí se všechny znaky.

Formát dat je potlačen v okamžiku, kdy je kurzor na daném poli.

Formáty lze opět měnit i programově.

### Validace polí

Proces validace je zahájen v okamžiku, kdy se uživatel pokouší opustit dané pole. Pokud pole obsahuje neplatnou hodnotu z hlediska typu nebo formátu, objeví se chybové hlášení a uživatel nemůže opustit pole s neplatnou hodnotou. Výjimkou je výběr položky menu. V takovém případě nejsou neplatná data přijata a programátor musí situaci ošetřit, např tak, že zablokuje některé položky menu sám. K tomu lze využít funkci **SalValidateMsg()**, poslanou z menu. Tato funkce pošle zprávu SAM\_Validate aktuálnímu poli. Postup bude vysvětlen později.

Každé pole, jehož hodnotu uživatel změnil, dostává zprávu SAM\_Validate. Při této zprávě lze provádět potřebnou validaci hodnoty. Samotná zpráva může vracet systému SQLWindows hodnotu, která určuje další postup. Pro tyto účely jsou v SQLWindows definovány konstanty:

**VALIDATE\_Cancel** ... zastaví zpracování, nedovolí uživateli opustit pole,

**VALIDATE\_Ok**, případně jakákoliv jiná hodnota než VALIDATE\_Cancel povolí pokračovat dál,

**VALIDATE\_Ok** ... povolí pokračování a nuluje příznak, že pole bylo editováno. Hodnotu tohoto příznaku lze zjistit, resp. nastavit funkcemi SalQueryFieldEdit(), resp.SalSetFieldEdit().

Př) Zkontrolujeme, že hodnota v poli nVek je v intervalu 0 až 150.

```
On SAM_Validate
```

```
If (dVek<0) OR (dVek>150)
```

```
CallSalMessageBox('Věk musí být v intervalu 0 až 150.',
```

```
'XXX',MB_Ok|MB_IconStop)
```

```
Return VALIDATE_Cancel
```

```
Return VALIDATE_Ok
```

Jinou zprávou, kterou lze využít pro validaci je **SAM\_AnyEdit**. Tato zpráva je poslána při každé změně editovatelného pole, např.po vstupu každého znaku. Zpráva se neposílá, pokud je hodnota měněna programově.

Při validaci lze mohou být užitečné funkce:

SalIsNull() - test, je-li zadaná hodnota,

SalIsValidInteger() - test na platnost celého čísla,

SalIsDecimal() - test na platné desetinné číslo zadaného počtu cifer a přesnosti.

### Zablokování položky menu

V předchozí části bylo uvedeno, že ani automatická validace, ani zpracování zprávy SAM\_Validate nezabrání uživateli ve výběru z menu. Je to proto, že pokud uživatel chce

opustit pole s hodnotou, která ještě nebyla validována, výběrem položky menu, neposílá se zpráva SAM\_Validate poli automaticky a záleží na programátorovi, zda zajistí její zaslání. To zajišťuje volání funkce **SalSendValidateMsg()** v sekci **Menu Actions** položky menu, např.:

```
If VALIDATE_Ok=SalSendValidateMsg()
```

.....  
Při zaslání této zprávy se provádí i automatická validace, zajišťovaná systémem.

Jisté problémy mohou nastat, pokud **není hodnota v poli zadaná** a není naprogramována validace s ohledem na tuto skutečnost. Při automatické kontrole se totiž nezadaná hodnota chápe jako platná. Pokud chceme zabránit výběru položky menu, pokud není některá hodnota zadaná, můžeme volat funkci **SalIsNull()** v sekci Enabled when odpovídající položky menu, např.:

```
Enabled when: NOT SalIsNull(sJméno)
```

## 4.12 Fonty a barvy

SQLWindows poskytuje plnou kontrolu nad fonty a barvami v aplikaci. Přiřazení lze provést v době návrhu nebo lze naprogramovat výběr pro uživatele a době běhu aplikace.

V době návrhu se nastavují fonty a barvy prostřednictvím atributů grafických objektů.

Implicitní hodnoty jsou nastaveny v sekci **Global Declarations-Window Defaults**.

K programovému nastavení barvy slouží funkce **SalColorSet()**. Buď lze využít předdefinované palety 32 barev nebo vytvořit libovolnou barvu kombinací složek RGB. K dispozici jsou funkce **SalColorToRGB()** a **SalColorFromRGB()**. Dále lze použít otevření dialogového okna barev funkcí **SalDlgChooseColor()**.

K programovému nastavení fontů slouží funkce **SalDlgChooseFont()**, **SalFontSet()**, **SalFontGet()**, **SalFontGetNames()** a **SalFontGetSizes()**.

Bližší informace o nastavování barev a fontů lze nalézt v [1], případně manuálech SQLWindows.

## 4.13 Okno se zprávou pro uživatele

Již několikrát byla v příkladech použita funkce **SalMessageBox()**, která otevře okno se zprávou pro uživatele a jedním až třemi tlačítky. Volání funkce má tvar:

```
nResp = SalMessageBox(sText, sTitle, nFlags), kde
```

sText je text zprávy,

sTitle je text titulku okna zprávy,

nFlags jsou příznaky, které určují tvar ikony nalevo od textu zprávy, počet a jména tlačítek, předvýběr tlačítka a režim okna. Příznaky jsou zadávány použitím definovaných konstant spojenými operátorem logického součtu (|).

nResp je návratová hodnota daná stiskem tlačítka dialogového okna.

Funkce bývá použita v příkazu **If**, pokud se testuje návratová hodnota, resp. **Call**, pokud na návratové hodnotě nezáleží.

Pro **ikonu nalevo** od textu jsou definovány tyto konstanty:

MB\_IconExclamation - ikona s vykřičníkem, používá se k varování,

MB\_IconQuestion - ikona s otazníkem, používá se k varování,

MB\_IconAsterisk - ikona se znakem i, používá se pro informace,

MB\_IconStop - ikona se značkou Stop, používá se pro akce.

**Tlačítka okna** a návratové hodnoty jsou určeny těmito konstantami:

MB\_OK - (implicitní), tlačítko OK, vrací hodnotu IDOK,

MB\_OKCancel - tlačítka OK a Cancel, vrací hodnotu IDOK, resp. IDCANCEL,

MB\_RetryCancel - tlačítka Retry a Cancel, vrací hodnotu IDRETRY, resp. IDCANCEL,

MB\_YesNo - tlačítka Yes a No, vrací hodnotu IDYES, resp. IDNO,

MB\_YesNoCancel - tlačítka Yes, No a Cancel, vrací hodnotu IDYES, resp.IDNO, resp.IDCANCEL,

MB\_AbortRetryCancel - tlačítka Abort, Retry a Ignore, vrací hodnotu IDABORT, resp.IDRETRY, resp.IDCANCEL.

Implicitně je přednastaven **výběr** prvního (nejlevějšího) tlačítka. Toto přednastavení lze změnit konstantami:

MB\_DefButton1 - implicitní přednastavení - první (nejlevější),

MB\_DefButton2 - druhé tlačítko,

MB\_DefButton3 - třetí tlačítko,

MB\_NoFocus - žádné tlačítko není předvybráno.

Př) If IDNO = SalMessageBox('Opravdu chcete zrušit záznam?', 'XXX',

MB\_YesNo|MBIconExclamation|MB\_DefButton2)

Funkce SalMessageBox vždy otevírá režimové dialogové okno.

**Režim** lze specifikovat konstantami:

MB\_ApplModal - implicitní - režim aplikace,

MB\_SystemModal - systémový režim okno (pro celá MS Windows).

#### **4.14 Programátorem definované zprávy**

Systém SQLWindows používá zpráv k provedení příkazů, uvedených u konkrétních objektů. Kromě systémových zpráv může programátor definovat vlastní zprávy a ty posílat objektům. Tyto zprávy umožňují vyvarovat se použití externích odkazů na objekty, mohou ušetřit kód při práci s příkazy SQL (dynamické sestavování příkazů) apod.

K zasílání zpráv slouží tři systémové funkce, které se vzájemně liší počtem adresátů, okamžikem zpracování a tím, zda vrací hodnotu od příjemce zprávy takto:

Funkce Počet adresátů Zpracována ihned? Vrací hodnotu?

SalSendMsg() 1 ano ano

SalPostMsg() 1 ne ne

SalSendMsgToChildren() více ano ne

#### **Definování konstanty zprávy**

Podobně, jako jsou definovány konstanty pro systémové zprávy (např.SAM\_Click), je třeba definovat konstanty pro zprávy definované programátorem. Každá taková konstanta musí být větší, než systémová konstanta **SAM\_User**. Tyto konstanty se definují v sekci **Global**

**Declarations-Constants-User**, např.:

Number: AM\_Evaluate = SAM\_User+1

#### **Funkce SalSendMsg()**

Posílá zprávu jedinému objektu typu okno. Zpráva je zpracována okamžitě, tj.příkazy reagující na zprávu jsou provedeny dříve, než příkazy následující za voláním funkce. Funkce SalSendMsg() návratovou hodnotou funkce je hodnota, kterou vrací příjemce. Tvar funkce je:

**nMsgReturn = SalSendMsg(hWnd, nMsg,nwParam,nlParam)**

hWnd - jméno okna nebo identifikátor výskytu,

nMsg - konstanta zprávy,

nwParam - hodnota parametru wParam,

nlParam - hodnota parametru lParam.

Parametrů wParam a lParam lze využívat k předávání příjemci. Ten může běžným způsobem pracovat s odpovídajícími speciálními globálními proměnnými wParam a lParam.

Př) Předpokládejme, že chceme uložit do databáze hodnoty z formuláře, tvořeného několika datovými poli. Před provedením příkazu INSERT chceme provést kontrolu, že hodnoty byly

zadány. Buď můžeme testovat nenulovost konkrétních datových polí přímo funkcí SalIsNull() nebo pošleme všem polím zprávu a při jejím přijetí objekt sám provede kontrolu. V tomto případě by kód u objektu (např. tlačítka), potvrzujícího požadavek na uložení, mohl vypadat takto:

```
Set hWndChild = SalGetFirsChild(hWndForm, TYPE_DataField)
While NOT hWndChild = hWndNULL
If NOT SalSendMsg(hWndChild, AM_Evaluate, 0, 0)
Return FALSE
Set hWndChild = SalGetNextChild(hWndChild,TYPE_DataField)
Call SqlImmediate('insert .....')
Call SqlImmediate('commit')
```

Pokud příjemce zprávy vrací hodnotu TRUE, posílá se zpráva dalšímu datovému poli, v opačném případě je posloupnost příkazů ukončena příkazem Return (vracená hodnota není podstatná). Protože jsou zprávy zpracovávány ihned, provádí se příkazy SQL až po kontrole.

U každého datového pole mohla být událost AM\_Evaluate zpracována takto:

```
On AM_Evaluate
If SalIsNull(hWndItem)
Call SalMessageBox(...)
Call SalSetFocuse(hWndItem)
Return FALSE
Return TRUE
```

Funkce SalSetFocus() nastaví jako aktivní datové pole, kde byla zjištěna hodnota NULL.

### **Funkce SalPostMsg()**

Adresuje opět jeden objekt, ale zpracování není okamžité, nýbrž až po provedení všech ostatních příkazů odesílatele. Navíc funkce nepředává hodnotu od příjemce, nýbrž pouze o úspěšnosti operace. Volání má tvar:

**bOk = SalPostMsg(hWnd, nMsg, nwParam, nlParam)**

Význam parametrů je stejný jako u funkce SalSendMsg().

V případech, kdy nezáleží na tom, kdy je zpráva zpracována a zda se vrací hodnota, lze použít SalSendMsg() nebo SalPostMsg(). Někdy je však nutné zpracování odložit a potom musíme použít funkci SalPostMsg().

Př) Předpokádejme, že chceme otevřít okno s nápovědou pro nějaké formulářové okno. Je žádoucí, aby se dialog s nápovědou otevřel až po vytvoření formulářového okna. Pokud bychom otvírali okno dialogu nápovědy při události SAM\_Create pro formulářové okno, nebyl by tento požadavek splněn, protože zpráva SAM\_Create se zpracovává ještě před vytvořením okna a otevření dialogového okna s nápovědou by neumožnilo dokončit otevření okna formuláře, pokud je dialogové okno režimové. Situaci můžeme vyřešit tak, že otevření dialogového okna zpozdíme posláním zprávy funkcí SalPostMsg(). Tedy formulářové okno by při události SAM\_Create neotvíralo přímo dialogové okno s nápovědou, nýbrž by samo sobě poslalo funkcí SalPostMsg() nějakou zprávu a teprve při jejím přijetí by se otevřelo okno s nápovědou. Např. takto:

```
On SAM_Create
Call SalPostMsg(hWndForm,AM_GetHelp), 0, 0)
On AM_GetHelp
Call SalModalDialog(.....)
```

### **Funkce SalSendMessageToChildren()**

Je zpracována okamžitě jako SalSendMessage(), avšak je poslána všem synovským oknům zadaného okna nejvyšší úrovně. Podobně jako SalPostMessage() vrací pouze informaci o úspěšnosti. Tvar volání je:

**bOk = SalSendMessageToChildren(hWnd, nMsg, wParam, lParam)**

Význam parametrů je stejný jako u předchozích funkcí.

Př) Použití funkce budeme ilustrovat na příkladě interní funkce, která vynuluje všechna datová pole zadaného formuláře.

Function: ClearFields

Description: Posílá zprávu AM\_ClearField všem synovským oknům zadaného formuláře.

Returns:

Parameters:

Window Handle: hWnd

Local Variables:

Actions:

Call SalSendMessageToChildren(hWnd, AM\_ClearField, 0, 0)

U každého pole, které se má po přijetí zprávy vynulovat, bude příkaz:

On AM\_ClearField

Call SalClearField(hWndItem)

### **Posílání systémových zpráv**

Funkce SalSendMessage(), SalPostMessage() a SalSendMessageToChildren() lze použít i k zaslání systémových zpráv, kterými můžeme simulovat nějakou situaci, např. stisk tlačítka, má-li být při nějaké jiné události provedena stejná akce. Tím lze ušetrit kód.

Př) Předpokládejme, že máme dialogové okno se seznamem pro výběr hodnoty, např. seznamem přípustných hodnot pro nějaké datové pole formuláře. Požadujeme, aby se dvojitý stisk tlačítka myši na některé položce seznamu choval stejně jako stisk tlačítka ukončujícího dialog. V takovém případě můžeme využít zprávy SAM\_Click, kterou pošle objekt typu seznam při události SAM\_DoubleClick tlačítka ukončujícímu dialog. Veškeré akce související s ukončením dialogu jsou potom programovány pouze u tohoto tlačítka pro událost SAM\_Click.

### **Použití zpráv k odstranění externích odkazů**

Uvažujme případ, kdy chceme stiskem tlačítka pbT v okně frmF1 otevřít formulářové okno frmF2. Po otevření okna tlačítko zablokujeme, aby nebylo možné otevírat další okna frmF2. Při uzavření okna frmF2 naopak tlačítko pbT odblokujeme. Můžeme k tomu použít funkci SalEnableWindow(frmF1.pbT) nebo využijeme možnosti zaslat zprávu tlačítku pbT. Lze to provést takto:

1. Definujeme parametr okna frmF2 (např. hWndCreator):

Window Parameters

Window Handle: hWndCreator

2. Při vytvoření okna frmF2 předá tlačítko pbT jako parametr svůj identifikátor:

On SAM\_Click

Call SalCreateWindow(frmF2, hWndNULL, hWndItem)

Call SalDisableWindow(hWndItem)

3. Při zrušení okna frmF2 posílá okno zprávu objektu hWndCreator:

On SAM\_Destroy

Call SalSendMessage(hWndCreator, AM\_Enable, 0, 0)

4. Tlačítko pbT se při přijetí zprávy AM\_Enable odblokuje:

On SAM\_Enable

Call SalEnableWindow(hWndItem)

## 4.15 Obrázky, tlačítka, práce s objekty OLE

Obrázky se podobají položkám na pozadí, mají však sekci Message Actions, tj. lze popsat chování pro různé zprávy. Obrázek lze zobrazit na tlačítku.

OLE objekty jsou objekty vytvořené v prostředí jiné aplikace, např. Paintbrush, které jsou vloženy do vytvářené aplikace. Takové objekty lze potom editovat využitím původního aplikačního prostředí. Bližší informace lze nalézt v [1] nebo manuálech systému SQLWindows.

## 4.16 Okno tabulky - základy

V okně tabulky lze zobrazovat data z různých zdrojů v tabelární formě. Řádky tabulky lze prohlížet v obou směrech, řádky lze modifikovat, rušit a vkládat nové.

Při práci s tabulkou je potřeba mít na paměti případný souběžný přístup k datům ve víceuživatelském prostředí. Využívá se zde speciálního sloupce každé tabulky v databázi s hodnotou ROWID řádku, což je jednoznačná identifikace řádku v rámci databáze. Zde se touto problematikou zabývat nebudeme. Informace lze nalézt v [1] nebo manuálech SQLWindows.

Tabulka je tvořena **sloupci** a **řádky**, průsečík řádku a sloupce se nazývá **buňka** (cell). **Aktuální řádek** při interakci s uživatelem je označen rámečkem (focus frame). Právě aktivní řádek tabulky se nazývá **kontextový řádek**. Zpravidla kontextový a aktuální řádek budou totožné, neboť při interakci se nově nastavený řádek stává kontextovým. Při neinteraktivní práci s řádky to však nemusí být pravda. Pokud použijeme v příkazech jméno sloupce, má hodnotu z kontextového řádku.

Na levé straně tabulky je speciální prázdný sloupec, zvaný **záhlaví řádku**, který má různé využití.

Synovské tabulky pracují s daty stejně jako tabulky nejvyšší úrovně. Objekty synovské tabulky však musí být z rodičovského okna odkazovány kvalifikovaně (např. tblZakaznik.colJmeno).

Při práci s oknem tabulky (nejvyšší úrovně nebo synovské) je třeba mít na paměti, že globální proměnná hWndForm vždy ukazuje na okno tabulky a hWndItem na sloupec.

Data v tabulce mohou pocházet z různých zdrojů (databáze, pole, soubory), podobně jako v případě seznamů. Na rozdíl od nich však nemusí obecně být všechna data tabulky uchovávána v paměti. Systém SQLWindows používá hierarchické uspořádání paměti: zobrazená část tabulky - **vyrovnávací paměť** (cache) - zdroj dat (databáze). V atributech tabulky lze nastavit maximální počet řádků (**Max Rows in Memory**), které se mají uchovávat ve vyrovnávací paměti. Implicitní hodnota je 100. Pokud se mají do zobrazovaného pohledu na tabulku dostat nové řádky, hledá je systém nejprve ve vyrovnávací paměti. Teprve pokud zde nejsou, vybírá je ze zdroje dat. Při výběru je nejprve uloží do vyrovnávací paměti a teprve potom je zobrazí. Kolik řádků je vybráno, závisí na parametrech funkce naplnění tabulky (viz dále). Je-li potřeba k uložení nově vybraných řádků ve vyrovnávací paměti uvolnit prostor, odstraňuje SQLWindows některé řádky z vyrovnávací paměti. Chceme-li tomu zabránit, můžeme využít atributu **Dischardable**. Je-li zrušení zakázáno a dojde k zaplnění vyrovnávací paměti, posílá systém tabulce zprávu **SAM\_CacheFull**.

### Naplnění tabulky

Budeme se zde zabývat pouze plněním tabulky z databáze. K tomu účelu slouží funkce:

**bOk = SalTblPopulate(hWndTbl, hSql, sQuery, nPopulateMethod)**

hWndTbl - jméno nebo identifikátor výskytu tabulky,

hSql - identifikátor spojení s databází, jehož výsledná tabulka má naplnit okno tabulky.

sQuery - řetězec dotazu nebo prázdný řetězec ("). Ve druhém případě se předpokládá, že dotaz již je přeložen.

nPopulateMethod - řídí plnění tabulky. Zadává se pomocí konstant:

TBL\_FillNormal - vždy vybírá pouze tolik nových řádků, kolik je třeba zobrazit.

TBL\_FillAll - vybírá ze zdroje dat všechny řádky, včetně těch, které nejsou zrovna zobrazovány.

TBL\_FillAllBackground - vybírá data k naplnění všech viditelných řádků a pak další s rychlostí 4řádky/s. Tento způsob se zpravidla používá v kombinaci s tlačítkem, které umožňuje uživateli kdykoliv výběr ukončit.

Př) Předpokládejme, že chceme naplnit okno tabulky tblZakaznik se sloupci colJmeno, colRodCislo, colUlice, colMesto a colRowID obsahem tabulky Zakaznik z databáze, včetně hodnoty ROWID. Předpokládejme, že připojení databáze již bylo provedeno a identifikátor je hSql. Naplnění by mohlo vypadat takto:k

```
On SAM_Create
```

```
Set sDotaz='SELECT jmeno,r_cislo,ulice,mesto,ROWID
```

```
INTO:colJmeno,:colRodCislo,:colUlice,:colMesto,:colRowID
```

```
FROM zakaznik ORDER BY jmeno'
```

```
Call SalTblPopulate(hWndForm, hSqlDotaz,sDotaz,
```

```
TBL_FillNormal)
```

### Přidávání a vkládání řádků

Vložení řádku do databáze prostřednictvím tabulky je operace probíhající ve dvou fázích. V první se vloží do zobrazované tabulky prázdný řádek, který uživatel potom naplní tak, aby se ve druhé fázi mohl vložit do databáze. Vhodným řídicím prvkem pro vkládání nových řádků je tlačítko.

Prázdný řádek lze přidat na libovolnou pozici v tabulce voláním funkce:

```
nNewRow = SalTblInsertRow(hWndTbl, nRow)
```

hWndTbl - jméno nebo identifikátor instance tabulky,

nRow - pozice, na kterou chceme řádek vložit,

nNewRow - číslo řádku nově přidaného řádku.

Další užitečné funkce při vkládání nového řádku jsou funkce SalTblQueryFocus(), která zjišťuje pozici aktuálního řádku a SalTblSetFocusCell(), která nastavuje aktuální buňku (řádek a sloupec). Jejich tvar je:

```
bOk = SalTblQueryFocus(hWndTbl, nRow, hWndCol)
```

nRow - vrácená hodnota čísla aktuálního řádku,

hWndCol - vrácená hodnota identifikátoru výskytu aktuálního sloupce.

```
bOk=SalTblStFocusCell(hWndTbl,nRow,hWndCol,nEditMin,nEditMax)
```

nEditMin - pozice nejlevějšího znaku v buňce pro editování, pro celou buňku má hodnotu 0 a nEditMax -1,

nEditMax - pozice nejpravějšího znaku v buňce pro editování, pro celou buňku má hodnotu -1 a nEditMin 0.

Př) Předpokládejme, že máme formulář s tabulkou tblZakaznik a tlačítkem pbNovy, které používáme ke vložení nového řádku. Při stisku tohoto tlačítka pošleme zprávu AM\_AddRow tabulce, která zajistí vložení prázdného řádku před aktuální řádek. Po vložení řádku nastavíme jako aktuální buňku sloupec colJmeno nového řádku.

Kód u tlačítka pbNovy potom bude:

```
On SAM_Click
```

```
Call SalSendMsg(tblZakaznik,AM_AddRow,0,0)
```

a reakce tabulky na zprávu AM\_AddRow:

```
On AM_AddRow
```

```
Call SalTblQueryFocus(hWndForm,nRow,hWndCol)
Set nRow=SalTblInsertRow(hWndForm,nRow)
Call SalTblSetFocusCell(hWndForm,nRow,colJmeno,0,-1)
Return TRUE
```

Není-li aktuální žádný řádek, je hodnota nRow=0(první řádek tabulky). Hodnota TRUE je vrácena funkci SalSendMsg().

Po vložení nového řádku může uživatel řádek editovat, vkládat řádky další, případně editovat jiné řádky. Pokyn pro vložení nových řádků, resp.změněných do databáze dává uživatel stiskem tlačítka nebo výběrem položky menu. Opět je vhodné poslat poslat zprávu oknu tabulky, které příslušné akce zajistí. V podstatě jde o volání funkce **SalTblDoInserts()** pro vložení nových řádků, resp.**SalTblDoUpdate()** pro aktualizaci změněných řádků. Před voláním těchto funkcí musí být zajištěno spojení s databází a musí být přeložen odpovídající příkaz INSERT, resp.UPDATE. Volání funkcí má tvar:

```
bOk = SalTblDoInserts(hWndTbl,hSql,bClearFlags)
```

```
bOk = SalTblDoUpdates(hWndTbl,hSql,bClearFlags)
```

hWndTbl - jméno nebo identifikátor výskytu tabulky,

hSql - identifikátor spojení s databází s přeloženým příkazem INSERT, resp.UPDATE,

bClearFlags - příznak nulování příznaku ROW\_New, resp. ROW\_Edited. Je-li TRUE, příznaky se nulují, v opačném případě ne. Hodnota FALSE je užitečná při neúspěšné operaci, kdy lze provést ROLLBACK a provést znovu.

Každý řádek obsahuje příznaky, které jsou potom využívány funkcemi SalTblDoInsert() a SalTblDoUpdate() ke zjištění, které řádky byly změněny,vloženy, zrušeny apod. Informace se zobrazuje ve sloupci se záhlavím řádku. Jsou definovány tyto příznaky:

**ROW\_New** - přidáný řádek, šipka v záhlaví řádku,

**ROW\_Edited** - editovaný řádek, zatržení v záhlaví řádku,

**ROW\_MarkDeleted** - nastavuje se programově, X v záhlaví řádku,

**ROW\_Selected** - vybraný řádek, zvýrazněn,

**ROW\_HideMarks** - nastavuje se programově, skrývá informace v záhlaví řádku,

**ROW\_Hidden** - nastavuje se programově, schová řádek,

Řádky s příznaky ROW\_New a ROW\_Edited nemohou být zrušeny z vyrovnávací paměti.

Hodnota atributu Max Rows in Memory udává maximum takto neoznačených řádků ve vyrovnávací paměti.

Pro práci s příznaky jsou k dispozici funkce:

**SalTblQueryRowFlags()** - dotaz, zda obsahuje řádek dané příznaky,

**SalTblSetRowFlags()** - nastavení příznaků,

**SalTblFindNextRow()** - nalezne další řádek se stejnými příznaky,

**SalTblFindPrevRow()** - nalezne předchozí řádek se stejnými příznaky.

Příznaky ROW\_MarkDeleted a ROW\_Selected se běžně používají k označení řádků, které mají být zrušeny.

Př) Předpokládejme, že použijeme položku menu Ulož změny pro uložení nových a změněných řádků. Opět použijeme zprávu zaslanou oknu tabulky (tblZakaznik) pro uložení řádků tabulky do databáze. U položky menu by mohl být příkaz:

```
Call SalSendMsg(tblZakaznik,AM_WriteDB,0,0)
```

a u tabulky tblZakaznik kód:

```
On AM_WriteDB
```

```
! vložení
```

```
Call SqlPrepare(hSqlDoTbl,'INSERT INTO zakaznik VALUES (:colJmeno, :colRodCislo, :colUlice, :colMesto)')
```

```
Call SalTblDoInserts(tblZakaznik,hSqlDoTbl,TRUE)
```

! aktualizace

```
Call SqlPrepare(hSqlDoTbl,'UPDATE zakaznik SET jmeno=:colJmeno,  
rod_cislo=:colRodCislo,  
ulice= :colUlice,mesto=:colMesto  
WHERE ROWID=:colRowID')
```

```
Set bOk=SalTblDoUpdates(tblZakaznik,hSqlDoTbl,TRUE)
```

Po uložení nových a změněných řádků je nutné provést příkaz SQL COMMIT, který potvrdí změny v databázi a případně provést nové načtení obsahu tabulky, pokud s ní uživatel bude dále pracovat. Mezi důvody, proč tabulku znovu načíst patří:

- nové a aktualizované řádky získávají novou hodnotu ROWID,
- nově vložené řádky mohou po uložení do databáze a smazání příznaku ROW\_New zmizet z vyrovnávací paměti (příznak ROW\_New tomu předtím bránil).

Př) Náš příklad by proto mohl pokračovat příkazy:

```
If bOk
```

```
Call SqlImmediate('Commit')
```

```
Call SalTblPopulate(hWndForm, hSql,
```

```
'SELECT jmeno, r_cislo, ulice mesto, ROWID INTO:colJmeno, :colRodCislo  
:colUlice,:colMesto,:colRowID
```

```
FROM zakaznik ORDER BY jmeno', TBL_FillNormal)
```

```
! zde by mohla být případně informace pro uživatele
```

```
! o úspěšném uložení
```

```
Return TRUE
```

Pokud tabulka obsahuje velký počet řádků, může být efektivnější načíst pouze nové a editované řádky. V takovém případě bychom bychom u funkcí SalTblDoInserts() a SalTblDoUpdates() nenulovali příznaky, tj. použili hodnotu třetího parametru FALSE a načtení by realizovaly příkazy:

```
! nastavení na začátek
```

```
Set nRow=TBL_MinRow
```

```
While SalTblFindNextRow(tblZakaznik,nRow,ROW_New|  
ROW_Edited,0)
```

```
Call SqlFetch(hSqlDotaz,nRow,nFetch)
```

```
! vynulování příznaků
```

```
Call SalTblSetRowFlags(tblZakaznik,nRow,ROW_New|  
ROW_Edited,FALSE)
```

### **Rušení řádků tabulky**

Podobně jako vkládání řádků i rušení může být chápáno jako dvoufázový proces. V první fázi se poznačí řádky, které mají být zrušeny a ve druhé se provede vlastní rušení.

Pro vlastní rušení jsou k dispozici funkce SalTblDeleteSelected() a SalTblDoDeletes(). Prvá ruší řádky s příznakem ROW\_Selected, zatímco u druhé lze specifikovat, zda se mají zrušit řádky s příznakem ROW\_Selected nebo ROW\_Selected. Funkce SalTblDoDeletes je obecnější. Volání má tvar:

```
bOk = SalTblDoDeletes(hWndTbl,hSql,nFlagsOn)
```

hWndTbl - jméno nebo identifikátor výskytu tabulky,

hSql - identifikátor spojení s databází s přeloženým příkazem DELETE,

nFlagsOn - buď ROW\_MarkDeleted nebo ROW\_Selected.

Příznak ROW\_Selected je automaticky nastaven, když uživatel vybere řádek tabulky.

Výhodnější je však používat k označení řádků, které mají být zrušeny, příznaku

ROW\_MarkDeleted (indikovaný X v záhlaví řádku). Vhodným způsobem pro nastavení je stisk tlačítka myši na záhlaví řádku, dvojí stisk příznak ruší.

SQLWindows posílá oknu tabulky zprávu SAM\_RowHeaderClick, resp. SAM\_RowHeaderDoubleClick při stisku tlačítka myši na záhlaví řádku. Sekce Message Actions tabulky proto může vypadat takto:

```
On SAM_RowHeaderClick  
Call SalTblSetRowFlags(hWndForm,IParam,ROW_MarkDeleted,TRUE)  
On SAM_RowHeaderDoubleClick Call  
SalTblSetRowFlags(hWndForm,IParam,ROW_MarkDeleted,FALSE)
```

V parametru IParam zprávy je číslo řádku, na kterém uživatel stiskl tlačítko. Tato hodnota je proto použita ve volání funkce nastavující, resp. nulující příznak.

Zrušení poznačených řádků lze provést analogicky vkládání řádků. Je-li použito stejné tlačítko nebo položka menu, lze pouze v sekci Message Actions tabulky přidat k příkazu On AM\_Write příkazy pro rušení:

```
! proved' rušení  
Call SqlPrepare(hSqlDoTbl,'DELETE FROM zakaznik  
WHERE ROWID=:colRowID')  
Set bOk =SalTblDoDeletes(tblZakaznik,hSqlDoTbl,  
ROW_MarkDeleted)
```

Následující úsek kódu představuje doplnění o test, zda jsou vůbec nějaké řádky vybrány, potvrzení požadavku na rušení uživatelem a teprve následné rušení:

```
! zpráva od ovládacího prvku  
On AM_DeleteRows  
! zjištění, zda je nějaký řádek poznačen  
If NOT SalTblAnyRows(tblZakaznik,ROW_MarkDeleted,0)  
Call SalMessageBox('Nejsou vybrány žádné řádky.',  
'Spořitelna',MB_IconAsterisk)  
Return FALSE  
! potvrzení zrušení  
If IDNO=SalMessageBox('Opravdu chcete zrušit označené řádky?',  
'Spořitelna',MB_YesNo|MB_IconAsterisk)  
Return FALSE  
! najdi a zruš řádky  
Call SqlPrepare(hSqlDoTbl,'DELETE FROM zakaznik  
WHERE ROWID=:colRowID')  
Set bOk =SalTblDoDeletes(tblZakaznik,hSqlDoTbl,  
ROW_MarkDeleted)  
If bOk  
Call SqlImmediate ('COMMIT')  
Return TRUE
```

Před příkazem Return TRUE by ještě mohlo proběhnout nové naplnění tabulky, případně vyprázdnění funkcí **SalTblReset**(tblZakaznik) podle další činnosti aplikace.

Je třeba si uvědomit, že při práci ve víceuživatelském prostředí je nutné respektovat možnost souběžného přístupu. My jsme předpokládali jednouživatelské prostředí, a proto jsme se těmito problémy nezabývali. Problémy jsou popsány a ukázáno jejich řešení v [1] a manuálech SQLWindows.

Okno tabulky poskytuje celou řadu dalších možností, kterých lze při tvorbě aplikace využít. V této části budou tyto možnosti pouze vyjmenovány, zájemce o podrobnější informace odkazujeme na [1], případně manuály SQLWindows.

## Možnosti menu Edit

Koncový uživatel může využít standardního menu Edit pro práci s tabulkou. Chování položek Paste a Clear závisí na tom, zda je vybrán pro práci celý řádek nebo pouze buňka. Systém nastavuje odpovídajícím způsobem příznaky řádků.

## Příznaky tabulky (table flags)

Implicitně lze měnit velikost sloupců tabulky, přesouvat je, zobrazovat sloupky posuvu, posouvat horizontálně o celé sloupce apod. Typo činnosti jsou řízeny příznaky tabulky, jejichž hodnoty lze nastavovat programově funkcí **SalTblSetTableFlags()**.

## Záhlaví řádku

Zatím jsme si ukázali použití záhlaví řádku pro zobrazení ikon příznaků řádků. Existuje však další důležité použití. Lze ho nastavit tak, aby zobrazoval stejnou hodnotu jako některý jiný sloupec okna tabulky. Sloupec záhlaví je needitovatelný a nelze ho přesouvat - lze vždy první zleva. K definování záhlaví řádku slouží funkce **SalTblDefineRowHeader()**.

Praktické použití spočívá v zobrazení primárního klíče nebo čísla řádku tabulky. Hodnoty sloupce s číslem řádku tabulky lze definovat zpracováním zprávy **SAM\_FetchRowDone**, kterou posílá funkce **SalTblPopulate()** vždy po zařazení řádku do tabulky, takto:

```
On SAM_FetchRowDone
```

```
! předpokládáme existenci sloupce colCisloRadku, který není
```

```
! vidět (nastaven atribut Visible) a je zobrazován v záhlaví řádků
```

```
Set colCisloRadku=IParam
```

Parametry definice záhlaví řádků lze zjistit funkcí **SalTblQueryRowHeader()**.

## Rozštěpení tabulky

Okno tabulky lze horizontálně rozdělit na dvě sekce. V horní se zobrazují řádky tabulky a v dolní lze zobrazit sumární informace získané funkcemi **SalTblColumnSum()** a **SalTblColumnAverage()** nebo přidávat nové řádky.

Tento způsob přidávání řádků do tabulky je zpravidla výhodnější, než způsob přímého vkládání do nerozdělené tabulky.

Funkce, která rozdělí tabulku, má tvar:

```
bOk = SalTblDefine SplitWindow(hWndTbl,nRowsLowerHalf,nFlags)
```

**hWndTbl** - jméno nebo identifikátor výskytu tabulky,

**nRowsLowerHalf** - počet řádků dolní části tabulky,

**nFlags** - konstanta **TBL\_Split\_Adjustable**, má-li mít koncový uživatel možnost posouvat hranici, nebo 0 v opačném případě.

Počet řádků dolní části zjistí funkcí **SalTblQuerySplitWindow()**.

Pro vložení řádku do dolní části tabulky se používá stejná funkce jako pro nerozdělenou tabulku s tím rozdílem, že čísla řádků jsou záporná (první řádek má nejmenší hodnotu).

Existují konstanty **TBL\_MinSplitRow**, udávající první řádek v dolní části tabulky, a

**TBL\_MinSlitRow**, udávající první nenaplněný řádek v dolní části tabulky. Tyto konstanty lze ji použít se specifikací řádku pro vložení u funkce **SalTblInsertRow()**.

Vložení řádku do dolní části tabulky a nastavení jako aktuálního provede posloupnost příkazů:

```
Set nRow = SalTblInsertRow(hWndForm,TBL_MinRow)
```

```
Call SalTblSetFocusRow(hWndForm,nRow)
```

## Atributy sloupců

Při návrhu lze nastavit atributy sloupců jako je možnost editace, zarovnání apod. Tyto atributy lze nastavovat i programově použitím funkce **SalTblSetColumnFlags()**.

## Přesouvání sloupců

Implicitně lze sloupce v tabulce přesouvat buď interaktivně nebo programově funkcí `SalTblSetColumnPos()`. Přesuny lze zakázat funkcí `SalTblSetLockedColumns()`.

## Další užitečné funkce

`SalTblQueryColumnPos()` - zjistí, kde v tabulce je daný sloupec.

`SalTblSortRows()` - provede setřídění tabulky podle určitého sloupce.

## Automatické a dynamické sloupce

SQLWindows umožňuje definovat sloupce v okně tabulky automaticky až v době běhu. Toho lze využít v případě, kdy umožníme koncovému uživateli formulovat vlastní dotaz, např. v SQL. V takovém případě se definuje pouze okno tabulky bez sloupců. Definování sloupců zajistí funkce `SalTblPopulate()`. Takto definované sloupce se označují jako **automatické**.

**Dynamickým sloupcem** rozumíme sloupec vytvořený za běhu programově. K tomuto účelu slouží funkce `SalTblCreateColumn()`.

### 4.17 Další možnosti při použití SQL

V této části jsou naznačeny některé další možnosti práce s jazykem SQL v prostředí SQLWindows.

## Použití vícenásobných identifikátorů spojení s databází

Při vkládání, aktualizaci a rušení řádků tabulek v databázi prostřednictvím okna tabulky, stejně jako pro naplnění tabulky v okně z databáze obecně používáme různé identifikátory spojení s databází, které identifikují přeložený odpovídající příkaz SQL.

Pokud bychom použili stejný identifikátor, došlo by při kompilaci nového příkazu ke ztrátě informace a případného výsledku příkazu předchozího. To by se projevilo tak, že by např. nebylo možné načíst další řádek tabulky okna.

Navíc se případně používá další identifikátor pro příkazy, které je třeba provést pouze za určitých okolností, např. při vzniku chyby.

Měla by platit zásada: pokud očekáváme opakované použití provádění téhož příkazu SQL, použijeme různé identifikátory pro takové příkazy a to i v případě, kdy se mění hodnoty vázaných proměnných v příkazu SQL (nemají na kompilaci příkazu vliv).

## Dotazy typu Master/Detail

Dotaz typu Master/Detail představuje ve skutečnosti dotaz na vztah s kardinalitou 1:M. Opět pracujeme se dvěma identifikátory příkazu SQL - jeden pro získání hodnoty atributu entity "Master" (tj. příslušného řádku tabulky v databázi) a druhý k nalezení odpovídajících entit (řádků tabulky) "Detail".

Př) Předpokládejme, že v aplikaci pro databázi spořitelny použijeme formulářové okno se synovským oknem typu seznam nebo tabulka k výběru zákazníka a dalším synovským oknem tabulka pro zobrazení účtů daného zákazníka. Pro naplnění seznamu nebo tabulky zákazníků bychom použili jeden příkaz SQL s jedním identifikátorem a k plnění tabulky s účty bychom použili další příkaz SQL s jiným identifikátorem. V tomto příkazu by se pracovalo s hodnotou rodného čísla zákazníka, vybraného v tabulce nebo seznamu. Příkaz by byl zkompileovaný a funkci `SalTblPopulate()` pro naplnění tabulky by se předával pouze identifikátor. Při plnění (tj. provedení zkompileovaného příkazu) se uplatní aktuální hodnota proměnné s rodným číslem.

## Uchování kontextu

Databázový server SQLBase, podobně jako jiné SRBD, implicitně ruší při provádění příkazu COMMIT všechny přeložené příkazy SQL a případné výsledky dotazu, uchovávané na serveru (jde ve skutečnosti o obdobu kurzoru). V některých případech, např. při dotazech tupu Master/Detail a práci s takto vybranými řádky tabulek, by bylo nutné po provedení příkazu COMMIT příkaz znovu přeložit a znovu naplnit tabulky v oknech. SQL Base, opět podobně jako jiné SRBD, umožňuje potlačit implicitní rušení. V SQLWindows k tomu slouží funkce **SqlSetParameter()**, která slouží k nastavování různých parametrů. V našem případě bychom ji použili např. po připojení databáze pro daný identifikátor spojení (hSql) takto:

```
Call SqlConnect(hSql)
```

```
Call SqlSetParameter(hSql,DBP_PRESERVE,TRUE,")
```

Funkce **SqlSetParameter()** zajistí uchování kontextu pro identifikátor spojení hSql při provedení příkazu COMMIT, avšak ne při provedení příkazu ROLLBACK.

### Dynamický SQL

Funkce **SqlPrepare()**, **SqlExecute()** a **SqlImmediate()** umožňují snadno kompilovat a provádět příkazy SQL, vytvořené až za běhu aplikace. Toho lze využít např. k dotazování příkladem, kdy se dotaz sestaví na základě zadaných hodnot v některých datových polích, zkompiluje a provede. Příklad použití lze nalézt v [1].

## 5 Tiskové sestavy

Chceme-li použít v aplikaci tiskovou sestavu, musíme nejprve vytvořit **šablonu** (template) takové sestavy. Jde o speciální soubor, ve kterém je definován tvar tiskové sestavy, vstupní proměnné, prostřednictvím kterých se dodávají do sestavy hodnoty, a některé další údaje.

Existují dva způsoby tvorby šablony tiskové sestavy: pomocí **generátoru sestav Report Windows** nebo lze šablonu vytvořit přímo v **prostředí SQLWindows**, případně i za běhu aplikace. Tento druhý způsob je vhodný v případech, kdy nepotřebujeme zvláštní formátování tiskové sestavy, nevytváříme na sestavě skupiny, pro které bychom potřebovali počítat souhrnné údaje apod. V této kapitole bude popsán pouze způsob vytvoření šablony v SQLWindows. Popis generátoru sestav Report Windows lze nalézt v příslušném manuálu.

Vytvoření tiskové sestavy pro prohlížení nebo tisk na tiskárně při běhu aplikace potom spočívá v použití **tiskového okna**, ve kterém se na základě informací z šablony tiskové sestavy zobrazují data (při přímém tisku na tiskárnu je toto okno skryto). V tiskovém okně se automaticky zobrazuje řádek ikon pro listování v sestavě, tisk apod. a posouvací sloupky pro vertikální a horizontální posun.

Tiskové okno může být vytvořeno systémem SQLWindows (tzv. standardní okno) nebo to může být jakékoliv okno, ve kterém se mohou zobrazovat data. Ve standardním okně se kromě již zmíněného řádku ikon a sloupků posunu automaticky vytváří menu s položkami File, View a Print. Podobné menu lze snadno vytvořit i v zákaznickém okně, neboť existují funkce pro všechny funkce položek menu standardního okna. V této kapitole se zaměříme především na použití standardního tiskového okna.

Data do tiskové sestavy mohou být vybírána z databáze, souboru nebo okna tabulky. Pro data z databáze a souboru poskytuje systém jednu sadu funkcí, pro data z okna tabulky druhou. V obou případech jsou k dispozici funkce pro vytvoření tiskové šablony, prohlížení, tisk a uzavření tiskové sestavy. Kromě jiných systémových funkcí spočívá základní rozdíl při práci s daty z databáze a tabulky v tom, že v prvním případě okno tiskové sestavy zasílá některému jinému oknu řadu zpráv, při jejichž zpracování se plní programové proměnné údaji z databáze, které se potom v sestavě zobrazují. Pokud jsou v sestavě data z okna tabulky, zobrazí se automaticky v sestavě celý obsah této tabulky.

## 5.1 Tiskové sestavy z databáze nebo souboru

### Vytvoření šablony tiskové sestavy

K vytvoření šablony (tzv. Quick report) slouží funkce:

**bOk = SalReportCreate(sReportTemplate,sBindVars,sInputs,  
bQuick,nError)**

sReportTemplate - jméno souboru, kam se uloží vytvořená šablona, s příponou .qrp.

sBindVars - znakový řetězec se seznamem programových proměnných, které budou dostávat data řádek po řádku. Oddělovačem je čárka.

sInputs - znakový řetězec se seznamem vstupních položek sestavy, který koresponduje se seznamem programových proměnných sBindVars. Jména vstupních položek jsou uvedena v záhlaví sloupců sestavy.

bQuick - nastavujte na TRUE, jinak se generuje prázdná šablona.

nError - vrácená hodnota chyby. Existují předdefinované konstanty RPT\_Err...

bOk - TRUE, pokud byla tisková šablona vytvořena.

Př) Předpokládejme, že chceme vytvořit tiskovou sestavu pro tisk zákazníků spořitelny.

Vytvoření šablony lze provést následující posloupností příkazů:

```
Menu Actions
```

```
Set sReport='zakaznik.qrp'
```

```
Set sBindVars='sJmeno,sR_cislo,sUlice,sMesto'
```

```
Set sInputs='Jméno,Rodné_číslo,Ulice,Město'
```

```
Set nError=1
```

```
Call SalWaitCursor(TRUE)
```

```
Set bTempl=FALSE
```

```
Loop
```

```
Set hWndReport=SalReportView(hWndForm,hWndNULL,  
sReport,sBindVars,sInputs,nError)
```

```
If bTempl
```

```
Break
```

```
If nError=RPT_ErrFileOpen
```

```
If IDYES=SalMessageBox('Soubor "zakaznik.qrp"  
s definicí tiskové sestavy nenalezen. Chcete vytvořit  
novou definici?','Spořitelna',MB_YesNo)
```

```
Set nError=1
```

```
Call SalWaitCursor(TRUE)
```

```
If NOT SalReportCreate(sReport,sBindVars,sInputs,  
TRUE,nError)
```

```
Break
```

```
Set bTempl=TRUE
```

```
Else
```

```
Break
```

```
If NOT bTempl
```

```
Break
```

```
Call SalWaitCursor(FALSE)
```

```
! překreslení menu kvůli zablokování položky Tisk
```

```
Call SalDrawMenuBar(hWndForm)
```

Funkce **SalWaitCursor()** slouží ke zobrazení ikony přesýpacích hodin, která indikuje, že operace neproběhne okamžitě.

### Vytvoření sestavy pro prohlížení sestavy

K vytvoření sestavy v tiskovém okně slouží funkce:

### **hWndReport = SalReportView(hWndServer, hWndReport, sReportTemplate, sBindVars, sInputs, nError)**

hWndServer - jméno nebo identifikátor instance okna, kterému bude tiskové okno posílat zprávy. Toto okno budeme v dalším nazývat tiskový server. Ten musí při zpracování zpráv zajistit plnění příslušných proměnných.

hWndReport - jméno nebo identifikátor instance tiskového okna. Má-li být vytvořeno standardní okno, zadává se hWndNULL. Jde-li o zákaznické okno, musí být před voláním funkce otevřeno.

Ostání parametry mají stejný význam jako u funkce SalReportCreate(). Vracená hodnota je identifikátor instance tiskového okna.

Funkce SalReportView() posílá tiskovému serveru zprávu **SAM\_ReportStart**, která zahajuje komunikaci tiskového okna s tiskovým serverem.

Posloupnost příkazů pro vytvoření standardního tiskového okna by se velice podobala posloupnosti pro generování tiskové šablony.

Př) Následující posloupnost příkazů otevírá standardní tiskové okno pro tisk zákazníků spořitelny. Pokud šablona neexistuje (nError=RPT\_ErrFileOpen), je vytvořen.

### **Uzavření tiskové sestavy**

Ve standardním tiskovém okně je položka menu **File-Close**, která okno uzavře. K uzavření tiskové sestavy u zákaznického okna slouží funkce:

#### **bOk = SalReportClose(hWndReport)**

Funkce pouze uzavře tiskovou sestavu, zákaznické okno neuzavírá.

### **Tisk sestavy**

Menu standardního tiskového okna obsahuje položku Print pro tisk na tiskárnu. Pro tisk ze zákaznického okna lze volat funkci **SalReportCmd()**, která se chová stejně.

Pokud chceme tisknout sestavu bez předchozího zobrazení v tiskovém okně, použijeme funkci:

```
hWndReport = SalReportPrint(hWndServer, sReportTemplate,  
sBindVars, sInputs, nCopies, nOptions, nFirstPage,  
nLastPage, nError)
```

Většina parametrů má stejný význam jako u předchozích funkcí, případně je zřejmý z názvu. nOptions specifikuje pomocí předdefinovaných konstant některé volby (tisk celé sestavy, draft mode).

Před zahájením tisku se zobrazuje standardní dialogové okno, jehož některé údaje (titulek, 2 řádky textu) lze modifikovat funkcí **SalReportDlgOptions()**.

Přímý tisk na tiskárnu je řízen stejně jako při prohlížení, tj. tiskový server dostává na začátku zprávu **SAM\_ReportStart**.

### **Zpracování tiskových zpráv**

Již bylo uvedeno, že při vytváření sestavy pro účely prohlížení nebo tisku posílá tiskové okno několik zpráv tiskovému serveru, který zprávy zpracovává a je zodpovědný za naplnění programových proměnných, jejichž hodnoty se v sestavě zobrazují.

Jednotlivé zprávy zpracovává tiskový server takto:

### **SAM\_ReportStar**

Je poslána jednou na začátku vytváření sestavy. Lze provést připojení k databázi, zkompileování dotazu apod.

### **SAM\_ReportFetchInit**

Posílá se automaticky po dokončení zpracování zprávy SAM\_ReportStart. Typicky se zde provádí příkaz SQL (SqlExecute()). Pokud zpracování zprávy vrátí hodnotu FALSE, ukončí zpracování tiskových zpráv.

### **SAM\_ReportFetchNext**

Posílá se, nevrátí-li zpracování zprávy SAM\_ReportFetchInit hodnotu FALSE. Je třeba provést funkci SqlFetch() a vrátit FALSE, pokud se již výběr ukončil. Při hodnotě TRUE se zpráva posílá znovu.

### **SAM\_ReportFinish**

Zprávu posílá funkce SalReportClose() nebo při uzavření standardního tiskového okna. V rámci zpracování lze zrušit spojení s databází, vynulovat identifikátor instance tiskového okna, pokud byl použit k zablokování vytváření násobných instancí.

Př) Následující posloupnost příkazů ukazuje zpracování událostí při tisku sestavy zákazníků:

```
Message Actions
On SAM_ReportStart
Set SqlDatabase='sporit'
Call SqlConnect(hSql)
Return SqlPrepare(hSql,'select jmeno,r_cislo,ulice,mesto
into :sJmeno,:sR_cislo,:sUlice,:sMesto from zakaznik')
On SAM_ReportFetchInit
Return SqlExecute(hSql)
On SAM_ReportFetchNext
Return SqlFetchNext(hSql,nFetch)
On SAM_ReportFinish
Call SqlDisconnect(hSql)
! odblokuji položku Tisk v menu
Set hWndReport=hWndNULL
Call SalDrawMenuBar(hWndForm)
```

## **5.2 Tiskové sestavy pro okno tabulky**

Již bylo uvedeno, že u sestav pro okno tabulky se používá jiná sada funkcí a že se nezpracovávají žádné tiskové události.

### **Vytvoření šablony tiskové sestavy**

**bOk = SalReportTableCreate(sReportTemplate, hWndTable, nError)**

hWndTable je jméno nebo identifikátor instance okna tabulky, jejíž hodnoty se mají v sestavě zobrazit.

nError - vrácené číslo chyby. Má-li se v případě chyby zobrazit standardní dialogové okno s chybovým hlášením, nastavuje se 1.

Př) Předpokládejme okno tabulky tblZakaznik. Příkazy pro vytvoření tiskové šablony by mohly mít tvar:

```
Set sReport='zakaznici.qrp'
Set nError=1
Call SalWaitCursor(TRUE)
Cal SalReportTableCreate(sReport,tblZakaznik,nError)
Call SalWaitCursor(FALSE)
```

### **Vytvoření sestavy pro prohlížení**

K dispozici je funkce:

**bOk = SalReportTableView(hWndTable, hWndReport, sReportTemplate, nError)**

Význam parametrů je stejný jako u předchozích funkcí.

Podobně jako u sestavy z databáze i tady lze vytvořit tiskovou šablonu v závislosti na úspěšnosti funkce SalReportTableView().

### **Uzavření sestavy**

Je stejné jako v případě sestavy pro data z databáze.

### **Tisk sestavy**

Podobně jako u dat z databáz lze použít položku z menu **Print** standardního tiskového okna nebo použít funkci **SalReportTablePrint()**. Funkce má podobné parametry jako SalReportPrint(). Jejich popis je uveden v [1] nebo manuálu.

### **5.3 Položky menu pro tisk v zákaznickém tiskovém okně**

**Funkce SalReportCmd()** s parametrem specifikujícím požadovanou činnost umožňuje vykonat všechny činnosti, které lze spouštět z menu standardního tiskového okna.

### **5.4 Obnova a fitrování dat sestavy**

Pokud potřebujeme data sestavy znovu zobrazit (např. byly provedeny změny v databázi nebo tabulce), případně chceme zmodifikovat dotaz pro výběr z databáze (filtrovat data použitím klauzule WHERE), lze využít funkce:

**bOk = SalReportReset(hWndReport)**

kteřá v případě data z databáze zahajuje nový cyklus tiskových zpráv posláním zprávy SAM\_ReportInit, u dat z tabulky automaticky provede nové načtení.

## **Literatura**

[1] Gietz W.: SQLWindows Programming. Gupta Corporation. 1993.