Více procesové & Víceuživatelské programování

Obsah

1. Úvod 1
   - 1.1. Vítejte 1
   - 1.2. Jak budete v kursu pracovat 1
   - 1.3. Scénář 1
   - 1.4. Konvence 4
   - 1.5. Copyright 5
   - 1.6. Příkazy 5
   - 1.7. Konvence názvů pro proměnné 7
   - 1.8. Aktivní objekty 8
   - 1.9. Konvence názvů pro metody a příkazy a funkce 8

Q. QuickCode Pro™ 10
   - Q.1. Import maker do QCP 10

2. Prostředí více procesů 4D 11
   - 2.1. Procesy 11
   - 2.2. Proměnné a jejich rozměr rozsah 12
     - 2.2.1. Lokální (místní) proměnné 13
     - 2.2.2. Proměnné procesu 13
     - 2.2.3 Meziprocesní proměnné 13
   - 2.3. Úlohy pro sledování s použitím meziprocesních proměnných 14

3. Začínáme s procesy 15
   - 3.1. Okno Procesu návrháře 15
   - 3.2. Číslo ID procesu 15
   - 3.3. Název procesu 15
   - 3.4. Stav procesu 16
   - 3.5. Čas procesu 16
   - 3.6. Spuštění procesu 16
   - 3.7. Není dost místa v paměti ke spuštění procesu 18
   - 3.8. Spuštění procesu při spuštění aplikace 18
   - 3.9. Použití přikazu BEEP k upozornění uživatele 18
   - 3.10. Funkce New Process 19
     - 3.10.1. Změna okna Denní proeje, tak že se otevře po spuštění 19

4. Události formuláře 22
   - 4.1. Události a metody 23
   - 4.2. Události, objekty a vlastnosti 25
     - 4.2.1. Klepnutelné objekty 25
     - 4.2.2. Objekty dostupné z klávesnice 26
     - 4.2.3. Změnitelné objekty 27
     - 4.2.4. Objekty dostupné tabelátorem 27
   - 4.3. Kategorie událostí 28

5. Vytvoření Ohlašovacího procesu 29
   - 5.1. Typy procesů 29
   - 5.2. DELAY PROCESS(Proces ID; Trvání) 29
   - 5.3. Current process -> Process ID 30
   - 5.4. CALL PROCESS (Process ID) 30
   - 5.5. Process state (Process ID) -> Integer 31

5.5.1. Vytvoření Ohlašovacího procesu k obnově dialogového okna DenníProdeje 32

6. Jedno okno ladění na proces 36
6.0.1. Procvičování otevírání okna ladění na proces.....36

7. Použití plovoucích oken.....38
7.1. Změna dialogu DenníProdeje na plovoucí okno 38
7.2. Další proměnné pro plovoucí okna.....39
7.3. Použití metody Při uzavření okna.....39
7.3.1. Zmenšení velikosti dialogu DenníProdeje 40

8. Zabránění vytvoření nadbytečných procesů 42
8.1. Jedinečné parametry pro New Process 42
8.2. BRING TO FRONT (Process ID) 42
8.2.1. Omezení otevírání více oken téže tabulky pomocí New Process {*}. 42
8.3. Více než jedno okno může snížit integritu dat 43
8.3.1. Omezení oken DenníProdeje na jedno.....43

9. Uschování a ukázání procesů.....45
9.1. PAUSE PROCESS(Process ID) 45
9.2. RESUME PROCESS(Process ID).....45
9.3. HIDE PROCESS (Process ID) 45
9.4. SHOW PROCESS (Process ID) 46
9.5. Process number (Process name) -> Process ID 46
9.6. REDUCE SELECTION ([Tabulka];Číslo).....46
9.7 Uschování ukázání a přerušení obnovení procesů.....47

10. Ovládání spouštění procesů 51
10.1. Použití Lokálních semaforů.....51
10.2. Použití meziprocessních array 52
10.3. Find in array (NázevArray; Hodnota; {Start}) -> Číslo.....52
10.4. Size of array (NázevArray) -> Číslo.....53
10.5. PROCESS PROPERTIES (IDProcesu; Název; Stav; Čas) 53
10.6. Get pointer(NázevObjektu) -> Ukazatel.....53
10.7. Tickcount -> Longint.....53
10.8. Testování na False 53
10.8.1. Metoda PROCESS_LSpawnProcess.....54

11. Proces pro import/export běžící na pozadí.....62
11.1. Příkazy pro práci se strukturou.....62
11.1.1. Table (UkazatelNaTabulku) -> Číslo.....62
11.1.2. Count fields (UkazatelNaTabulku) -> Číslo 62
11.1.3. Field (ČísloTabulky; ČísloPole) -> Ukazatel.....62
11.1.4. Fieldname (UkazatelNaPole) -> Řetězec 62
11.1.5. GET FIELD PROPERTIES (UkazatelNaPole; Typ; {Délka}; {Index}) 63
11.1.6. Type (Pole nebo Proměnná).....63
11.2. Zásuvné metody (plug in).....63
11.2.1. ACI Pack 63
11.2.2. GetFieldInfos (LTableNum;LFieldNum;LRelateTable;LRelateFld;LAttribs;sList) ->Integer 64
11.3. Více příkazů pro práci s array.....64

11.3.1. INSERT ELEMENT (NázevArray; Umístění; {PočetPrvků})....64
11.3.2. DELETE ELEMENT (NázevArray; Umístění: {PočetPrvků})....65
11.3.3. COPY ARRAY (ZdrojovéArray; CílovéArray)....65
11.4. Prvky jazyka pro práci s dokumenty....65
11.4.1. Open document (Dokument; {Typ}) -> Odkaz na dokument....65
11.4.2. SET CHANNEL (Operace; Dokument)....66
11.4.3. SEND PACKET ({OdkazNaDokument}; Data)....66
11.4.4. RECEIVE PACKET ({OdkazNaDokument}; Proměnná; (ZnakKonce nebo PočetZnaků))....66
11.4.5. SEND RECORD ({Tabulka})....67
11.4.6. RECEIVE RECORD ({Tabulka})....67
11.4.7. CLOSE DOCUMENT (OdkazNaDokument)....67
11.4.8. Proměnné dokumentu....67
11.5. Jiné příkazy k vytvoření procesu pro Import/Export....68
11.5.1. BUTTON TEXT (NázevTlačítka; TextTlačítka)....68
11.5.2. COPY NAMED SELECTION ({Tabulka; Název)....68
11.5.3. CUT NAMED SELECTION ({Tabulka}; Název)....68
11.5.4. USE NAMED SELECTION ({Tabulka}; Název)....68
11.5.5. CLEAR NAMED SELECTION ({Tabulka}; Název)....68
11.5.6. Substring (Zdroj; PrvníZnak {;PočetZnaků}) -> Řetězec....69
11.5.7. Get pointer (NázevProměnné) ->Ukazatel....69
11.5.8. CREATE RECORD ({Tabulka})....69
11.5.9. CREATE EMPTY SET ({Tabulka}; Název)....69
11.5.10. ADD TO SET (Název)....70
11.5.11. Time (Řetězec) -> Čas....70
11.5.12. FLUSH BUFFERS....70
11.5.13. POST KEY (AsciiKódZnaku; Modifikátor)....70
11.5.14. DRAG AND DROP PROPERTIES (srcObject; srcElement; srcProcess)....71
11.5.15. Drop position -> Číslo....71
11.5.16. VARIABLE TO BLOB (MáProměnná; NázevBlobu{; offset | *})....71
11.5.17. BLOB TO VARIABLE (NázevBlob; MáProměnná{; offset})....72
11.5.18. SET BLOB SIZE (blob; Velikost{; Výplň})....73
11.5.19. Při poklepnutí....73
11.5.20. Bitové operace....73
11.6. Předávání parametrů procesům....74
11.7. Vytvoření procesu na pozadí....74
11.7.1. Vytvoření procesu, který může být spuštěn na pozadí....74
11.8. Jak udělat názvy polí a tabulek uživatelsky přívětivější....100
11.8.1. Umístění mezer do názvů polí k zpřehlednění rozhraní....101

12. Ovládání více oken procesů....104
12.1. Abs (Číslo) -> Číslo....104
12.2. Frontmost process ({*}) -> Integer....105
12.3. Více plovoucích palet potřebuje přepínací záhlaví....105
12.4. Vytvoření palety tabulek....105
12.4.1. Vytvoření palety se seznamem, reprezentujícím dostupná okna....105
Extra kredit Diskuze....114
12.4.2. Synchronizování Palety tabulek s oknem na popředí....114
12.5. Uschování procesu a okna....118
Extra kredit Diskuze....118
12.5.1. Provedení změny v tlačítkách výstupního formuláře....119

Obsah

14. Vše co potřebujete znát o zamykání záznamů 120
14.1. Automatická upozornění na zamknuté objekty 120
14.2. Definice týkající se záznamů 120
14.2.1. Platný záznam 121
14.2.2. Zavedený záznam 121
14.2.3. Uzamčený záznam 121
14.3. Příkazy s vlivem na stav (mód) tabulky 121
14.3.1. Příkaz READ ONLY 121
14.3.2. Příkaz READ WRITE 122
14.3.3. Příkazy, které se snaží zavést a uzamknout záznam 122
14.3.4. Příkazy, které dočasně mění stav tabulky na pouze číst 123
14.4. Zabránění nechtěnému zamykání záznamů a inicializace nového procesu 124
14.4.1. Nastavení všech tabulek READ ONLY, pouze číst 125
14.5. Příkazy, které pomáhají při testování a zacházení s zamčenými záznamy 128
14.5.1. Locked ([Tabulka]) -> Logické 128
14.5.2. LOAD RECORD ([Tabulka]) 128
14.5.3. UNLOAD RECORD ([Tabulka]) 128
14.5.4. Testování před změnou stavu, zda tabulka je nastavena na pouze číst 129
14.6. Testování před prováděním změn, zda záznam nebyl zatím vymazán 129
14.7. Zavedení metod k úpravám vztažených záznamů 129
14.7.1. Dočasné nastavení tabulky READ WRITE 130
14.8. Někdy chybí varování 135
14.9. Texty automatických tlačítek 135
14.10. Sada LockedSet 135
14.10.1. Kontrola LockedSet po provedení DELETE SELECTION 136
14.11. Making sequence number handle locked records 138
14.11.1. Upravte metodu LNextSequence na uzamykání tabulek a zázmamů 138
14.12. Simulace více uživatelů procesy 141

15. 4th Dimension product line 142
15.1. Produkty pro jednoho uživatele 142
15.2. Produkty pro více uživatelů 142
15.3. Moduly připojení: 142
15.4. Další zásuvné moduly (plug-ins) 142

16. Technologie sdílení souborů 144
16.1. Víceuživatelské verze 4th Dimension 2.x 144
16.1.1. Data nemohou být vyrovnávána do paměti 144
16.1.2. Všechny operace s daty se musí vykonat na pracovn stanici 144

17. 4D Server technologie klient/server 146
17.1. Úlohy prováděné 4D Server 146
17.1.1. Vyrovnávání dat do paměti (caching) 146
17.1.2. Provádění operací nad daty 146
17.1.3. Provádění indexace 146
17.1.4. Provádění triggeru 146
17.1.5. Provádění uložených procedur 147
17.2. 4D Client 147
17.3. Provádění paralelního zpracování 147

Více procesové & Víceuživatelské programování

Obsah

18. Požadavky na hardware a úvahy o něm 148
18.1 Hardware 4D Server 148
18.1.1. Pevný disk serveru 148
18.1.2. Paměť serveru 148
18.1.3. Rychlost CPU serveru 148
18.2. Hardware 4D Client 149
18.2.1. Rychlost CPU klienta 149
18.2.2. Paměť klienta 149
18.2.3. Pevný disk klienta 149
18.3. Úvahy o síti 149

19. Zatížení sítě způsobené 4D Server 150
19.1. Při přihlášení uživatele 150
19.2. Když uživatel otevře formulář jsou zavedeny následující položky: 150
19.3. Když uživatel provádí dotaz 150
19.4. Když uživatel tiskne 150
19.5. Když se uživatel odhlašuje 150
19.6. Když je uživatel náhodně odpojen (problémy na síti, v počítači klienta) 150

20. Transakce 152
20.1. Příkazy transakce 152
20.2. Transakce a zamykání záznamů 152
20.3. Automatické transakce během vstupu dat 153
20.3.1. Vytvoření transakcí příkazy jazyka místo automaticky 154
20.4. Transakce a 4D Backup. 158
20.5. Tipy a rady pro transakce 158
20.5.1. Nepoužívejte UNLOAD RECORD v transakci 158
20.5.2. Příkaz DISTINCT VALUES se chová speciálním způsobem 158
20.6. Záměrné uzamčení záznamů transakcí 158

21. Komunikace mezi pracovními stanicemi 160
21.1.Použití globálních semaforů 160
21.2. Příklad použití globálního semaforu 160
21.3. Použití záznamů v databázi ke komunikaci 162

22. Otázky licencí a 4D Server 163
22.1. Ošetřování událostí 165
22.1.1. Odpojení neaktivních uživatelů 165
22.2. Ztracení uživatelé 170

23. Předvedení 4D Server 171
23.1. Demonstrace použití vyrovnávací paměti aplikace 171
23.2. Předvedení serveru pro více uživatelů 171
23.3. Předvedení automatických hlášení v Prostředí návrháře 171
23.4. Kontrola uzamčenosti formulářů v Prostředí návrháře 171
23.5. Úprava formuláře zatímco jiní uživatelé tento formulář používají pro zobrazení dat 171

24. Techniky optimalizace 172
24.1. Kompilace vaší databáze 172
24.1.1. Vytvoření metody COMPILER 174

24.2. Pište kód, který je optimalizován pro 4D Server, kdekoliv je to možné....179
24.3. Vyhněte se příkazům, které se provádějí pouze na klientu....180
24.4. Nahrazení příkazů, které se provádějí pouze na klientu jinými příkazy....181
24.4.1. Náhrada APPLY TO SELECTION....181
24.5. Nezatěžujte zbytečně server....185
24.6. Používejte správně sady....186
24.7. Nepřetěžujte provádění kompilovaného kódu....187
24.8. Řízení dotazování....188
24.9. Řízení třídění....188
24.10. Prázdná první tabulka....188
24.11. Použití automatických relací a výstupní formuláře....189
24.12. Vstupní formuláře....189
24.12.1. Umístěte všechny společné objekty vícestránkového formuláře na stránku 0....189
24.12.2. Snižte co nejvíce počet objektů na stránce formuláře....190
24.12.3. Překonvertujte všechny statické objekty ve formuláři do jednoho obrázku....190
24.12.4. Rozdělte velké vícestránkové formuláře do oddělených formulářů....190
24.13. Nastavte tabulky READ ONLY....190
24.14 Řízení procesu....191
24.14.1. Úpravy LSpawnProcess k vyvolání pozastavených procesů....191
24.14.2. Prověrka okna Seznam procesů na lokální a globální procesy....197
24.14.3. Nepoužívejte proces Uživatel/Aplikace....198
24.14.4. Pozastavení procesu Uživatel/Aplikace....198
24.15. Ovládání výběrů....199
24.16. Používejte array ukazatelů místo častého volání Get pointer....199
24.16.1. Funkce zacházející s datumy....199
24.16.2. Demonstrace použití array ukazatelů....201
24.17. Vyhněte se častým voláním operačního systému....203
24.17.1. Optimalizace dotazů pro kompilovaný kód....204
24.18. Používejte RELATE ONE místo LOAD RECORD....210
24.19. Vyhněte se podtabulkám....210
24.20. Zdroje (resource)....210
24.21. Jestliže mažete větší množství záznamů proveďte kompaktaci....211
24.22. Vyhněte se použití grafiky v oblasti obsahu výstupních formulářů....211

26. Příkazy a metody, které přidávají speciální funkce pro 4D Server....212
26.1. Count users -> Integer....212
26.2. Sequence number -> Longint....212
26.3. Current date (*)....212
26.4. Current time (*)....212
26.5. On Server Startup....212
26.5.1. Použití Current date a time účinněji pro síť....212

27. Uložené procedury....215
27.1. Stručný přehled....215
27.2. Proměnné a jejich rozsah na serveru....216
27.3. Výměna dat mezi procesy klientů a serveru....217
27.3. Jak provést uloženou proceduru?....218
27.3.1. Provedením v Prostředí uživatele....219
27.3.2. Execute on server (Procedura ; VelikostStack { ; NázevProcesu { ; Parametr1 { ; Parametr2 ;... } } } { ; * }) -> ČísloIDProcesu....219
27.4. Meziprocessní komunikace....220

27.4.1. SET PROCESS VARIABLE (IDProcesu ; CílováProm1 ; Výraz1 { ; CílováProm2 ; Výraz2; ... }) 220
27.4.2. GET PROCESS VARIABLE (IDProcesu ; ZdrojováProm1 ; CílováProm1 { ; ZdrojováProm2 ; CílováProm2 ; ... }) 220
27.4.3. VARIABLE TO VARIABLE (IDProcesu; CílováProm1; ZdrojováProm1 {; CílováProm2; ZdrojováProm2; ...; CílováPromN; ZdrojováPromN}) 220
27.5. Správné použití uložených procedur 221
27.6. Na co nepoužívat uložené procedury 222
27.7. Rychlý pohled do databáze s uloženými procedurami 222
27.8. Začínáme s uloženými procedurami 223
27.8.1. Testování uložených procedur na rychlost a účinnost 224
27.8.2. Tickcount -> Číslo 232

28. Ovládání diskových souborů nastavení 233
28.1. Pokud možnost umístěte složku s nastaveními na server jiný než počítač 4D Server 233
28.2. Vzory dokumentů pro moduly 233

29. Zabezpečení systému pomocí hesel 234
29.1. Soubor cesty 234
29.2. Vytváření hesel 234
29.3. Přístup do Prostředí návrháře 235
29.3.1. Nastavení systému hesel 235
29.4. Přístup do Prostředí uživatele 236
29.4.1. Zakázání přístupu uživatele do Prostředí uživatele 237
29.5. Přístup k modulům 237
29.6. Příkaz EDIT ACCESS 237
29.7. Příkaz CHANGE ACCESS 238
29.8. Příkaz CHANGE PASSWORD 238
29.9 Rozhraní, které umožní uživatelům měnit svá hesla 238
29.9.1 Vytvoření systému přístupu na heslo 238
29.10. Vymazání uživatele 240
29.10.1. DELETE USER(IDUživatele) 240
29.10.2. GET USER LIST (asJménaUživatele; asČíslaUživatele) 240
29.10.3. Is user deleted(IDUživatele) -> Logické 240

29. Použití sítě, které může ovlivnit chování 241
29.1. Tisk 241
29.2. Přenosy souborů 241
29.3. Elektronická pošta 241
29.4. Přístup k síťovým zařízením 241
29.5. Nesprávně ukončené uzly nebo poškozené kabely 241

30. Okno 4D Server 242
30.1. User Interface 242
30.2. Cache Manager 243
30.3. Client Manager 243
30.4. Indexing 243
30.5. Cache/Hit Ratio 243
30.6. Okno Cache/Hit Ratio 243

Obsah

31. Použití paměti na serveru 245
31.1. Uživatelé, procesy a výběry 245
31.2. Nastavení předvoleb databáze, keré má vliv na paměť serveu a chování 246
31.2.1. Scheduler (Řízení volání) 247
31.2.2. Vyrovnávací paměť databáze 248
31.2.3. 4D hlavní paměť(Main Memory- pouze Windows) 248
31.2.4. Tradiční nastavení paměti pro Macintosh 248
31.3. Halda aplikace 4DServeru 249
31.4. To že můžete ještě neznamená, že máte 251
31.5. Vyrovnávací paměť systému 251
31.6. Velikost Stack pro procesy klientů na serveru 251
31.7. Sady 251

32. Použití paměti na klientu 252
32.1. Procesy 252
32.2. Prvky struktury: Nabídky, Metody, Formuláře, Metody formulářů, objektů, Seznamy atd. 253
32.3. Nastavení použití paměti 254
32.4. Poznámky k souboru .rex 254
32.5. Rychlejší překreslování obrazovky 255

33. Jiné produkty 4D používané se 4D Server 256
33.1. 4D Backup 256
33.2. 4D Open 256
33.3. Moduly produktivity 257
33.4. Moduly připojení 257

34. Použití našich vlastních nástrojů k rozvíjení aplikace 258

35. Závěr 260

Příloha A: Optimalizace Windows NT pro zlepšení chování 4D Server ... 261
Úvod 261
Upozornění 261
Použití systému souborů NT (NTFS) 261
Optimalizace Pagefile Size Settings 261
Dialog Virtuální paměti z Ovládacích panelů v systému 262
Komprese složky obsahující datový soubor 4D 263
Spuštění 4D Server z batch souboru v REALTIME Mode 264
Konfigurace obrazovky 264
Display control panel 265
Nastavení plochy 265
Desktop control panel 266
Závěr 267

Rejstřík 268

4th Dimension

Příručka školení

Víceprocesové & Víceuživatelské programování

Verze 6.0.3

Vytvořeno:

Jméno	E-Mail
Kent Wilbur	kent@acius.com

Upraveno:

Jméno	E-Mail
Jaroslav Macháček	inforce@mbox.vol.cz

ACI Univerzita

1. Úvod

1.1. Vítejte

Tento kurz je zaměřen na lidi, kteří mají zkušenosti s programováním ve 4th Dimension ("4D") a chtějí se dozvědět o výhodách více procesů ve 4D a jak optimalizovat kód pro užití se 4D Serverem. Kurz pokrývá koncept, kerý Vám umožní ovládat prostředí více uživatelů a více současných procesů pro jednoho uživatele..

1.2. Jak budete v kursu pracovat

Délka trvání kursu je dva dny. Způsob práce je následující:

- Instruktor diskutuje některé rysy a demonstruje je s pomocí promítání na plátno, vy provádíte cvičení a příklady z této příručky, které procvičují diskutované rysy.
- Vy provádíte cvičení a příklady z této příručky, které procvičují diskutované rysy.
- Instruktor při cvičeních odpovídá na dotazy a poskytuje pomoc těm, kteří ji potřebují.

1.3. Scénář

Při práci v tomto kurzu budete přidávat nové rysy k zjednodušené databázi pro distributora videokazet. Je to tatáž databáze, kterou jste vytvořili v kurzu „Úvod do 4thDimension". Budete sledovat zákazníky, jejich faktury, položky těchto faktur a seznam prodejních produktů.

Databáze je pojmenována po vaší fiktivní společnosti "ACI Video." Na konci kurzu bude databáze zahrnovat tabulky a pole ze seznamu níže, je tatáž struktura jako v kurzu „Programování ve 4D".

[zDialogy]

Název pole	Typ	Vlastnosti
Povinnné	Logické	

[Zákazníci]

Název pole	Typ	Vlastnosti
IDzákazníka	Alfa 10	Nutný vstup; Pouze zobrazit; Indexované; Jedinečné
Jméno	Alfa 20	Indexované
Příjmení	Alfa 20	Indexované
Firma	Alfa 25	Indexované; Nutný vstup
Adresa	Alfa 25	
Město	Alfa 20	Indexované
Stát	Alfa 2	Indexované
PSČ	Alfa 10	Indexované
Telefon	Alfa 10	
PlátceDaně	Logické	
Důležitý	Logické	
CelkovéProdeje	Real	
DatumVytvoření	Datum	Pouze zobrazit; Neviditelné
ČasVytvoření	Čas	Pouze zobrazit; Neviditelné
DatumÚpravy	Datum	Pouze zobrazit; Neviditelné
ČasÚpravy	Čas	Pouze zobrazit; Neviditelné

[Faktury]

Název pole	Typ	Vlastnosti
IDZákazníka	Alfa 10	Nutný vstup, Neměnné; Indexované; Jedinečné
ČísloFaktury	Long Integer	Indexované; Jedinečné
DatumFaktury	Datum	Indexované
FakturaCelkem	Real	
Placeno	Logické	
ZpůsobPlatby	Alfa 15	
DatumVytvoření	Datum	Pouze zobrazit; Neviditelné
ČasVytvoření	Čas	Pouze zobrazit; Neviditelné
DatumUpravení	Datum	Pouze zobrazit; Neviditelné
ČasUpravení	Čas	Pouze zobrazit; Neviditelné

[PoložkyFaktur]

Název pole	Typ	Vlastnosti
ČísloFaktury	Long Integer	Indexované
IDZbožíProdukty	Long Integer	Indexované
Neužito	Logické	Neviditelné
Cena	Real	
Množství	Integer	
JednCena	Real	
CenaCelkem	Real	

[Produkty]

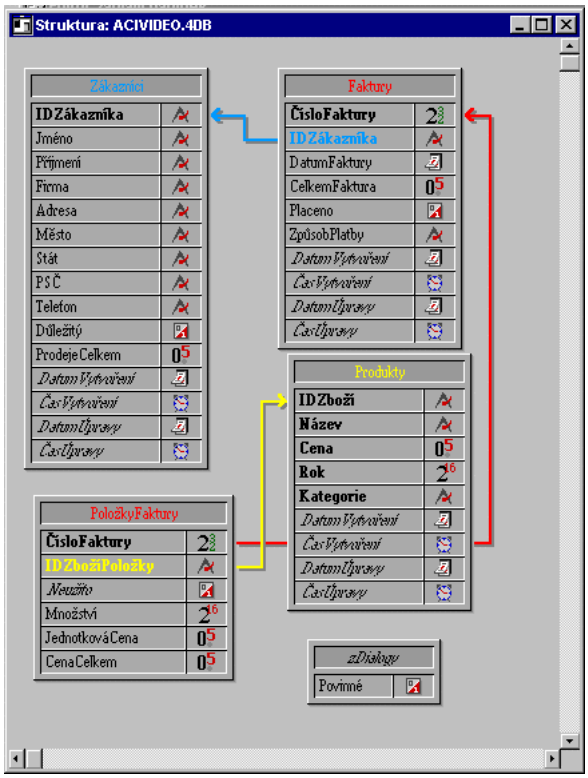
Název pole	Typ	Vlastnosti
IDZboží	Alfa 10	Nutný vstup; Neměnné; Indexované; Jedinečné
Název	Alfa 30	Indexované
Cena	Real	Indexované
Rok	Integer	Indexované
Kategorie	Alfa 15	Indexované
DatumVytvoření	Datum	Pouze zobrazit; Neviditelné
ČasVytvoření	Čas	Pouze zobrazit; Neviditelné
DatumUpravení	Datum	Pouze zobrazit; Neviditelné
ČasUpravení	Čas	Pouze zobrazit; Neviditelné

[zNavracenáČísla]

Název pole	Typ	Vlastnosti
NázevSekvence	Alfa 20	Indexované
Hodnota	Long integer	

[zSekvence]

Název pole	Typ	Vlastnosti
NázevSekvence	Alfa 20	Indexované
Hodnota	Long integer	



1.4. Konvence

- “Programování” a “tvorba kódu” jsou synonyma pro instrukce, které budete vkládat.
- To co budete zapisovat je v tomto tvaru.

  Příklad: Napište Johnson do pole Příjmení.
- Speciální klávesy na klávesnici jsou uvedeny následovně:

  Stiskněte Enter.
- Jestliže je požadováno, abyste stiskli více než jednu klávesu současně je tento požadavek zapsán následovně:

  Stiskněte CTRL + Shift + 3.
  Vysvětlení: Stiskněte současně klávesy CTRL, Shift a 3.

Macintosh™ ekvivalent bude okamžitě následovat po Windows™ v kurzívě, jak je ukázáno níže.
Stiskněte: Ctrl + Shift + 3 *(Comma + Shift + 3)*

- Volba položky z nabídky je popsána následovně:

  Zvolte Soubor ¬ Otevřít (Ctrl + O) *(_ + O)*
  Vysvětlení: Z nabídky Soubor, zvolte položku Otevřít. Na Windows™ stiskněte Control + O.
  Na Macintosh™ stiskněte Comma + O.

- Položky 4D kódu jsou zobrazovány stejně jako jsou v 4D Editoru metod. Například:

[Zákazníci]Stát	Pole
ALERT	Příkaz
MyProcedure	Metoda (procedura)

- Sekce označené "Extra Credit" jsou výběrové. Jesliže skončíte svůj příklad dříve můžete pracovat v sekci "Extra Credit" a naučit se více.

1.5. Copyright

20883 Steven Creek Blvd.

Cupertino,. CA 95014

(408) 252-4444

Příklady kódu používané v této příkladu jsou považovány za veřejné, mohou být použity v libovolné databázi, kterou vytvoříte.

1.6. Příkazy

V tomto kurz se naučíte následující příkazy z jazyka 4D. Některé z těchto příkazů jsou podrobně vysvětleny jen v dalších rozšiřujících materiálech a jsou určeny pro

samostudium a některé z příkazů nejsou použity v kódu a jsou jenom diskutovány v materiálech kurzu.

Abs	DELAY PROCESS	ON SERIAL PORT CALL
Activated	DELETE ELEMENT	ONE RECORD SELECT
ADD TO SET	DELETE USER	Open document
Append document	DRAG AND DROP PROPERTIES	ORDER BY FORMULA
ARRAY BOOLEAN	Drop position	Outside call
ARRAY DATE	EDIT ACCESS	PAUSE PROCESS
ARRAY INTEGER	Execute on server	POST KEY
ARRAY LONGINT	EXPORT DIF	PREVIOUS RECORD
ARRAY PICTURE	EXPORT SYLK	Process number
ARRAY POINTER	Field	PROCESS PROPERTIES
ARRAY REAL	Field name	Process state
ARRAY TEXT	Find in array	READ ONLY
ARRAY TO LIST	FLUSH BUFFERS	Read only state
ARRAY TO SELECTION	Frontmost process	READ WRITE
Ascii	GET FIELD PROPERTIES	RECEIVE PACKET
Average	Get pointer	RECEIVE RECORD
BEEP	GET PROCESS VARIABLE	REDUCE SELECTION
BLOB TO VARIABLE	GET USER LIST	RESUME PROCESS
BRING TO FRONT	GOTO RECORD	SCAN INDEX
BUTTON TEXT	GOTO SELECTED RECORD	QUERY BY FORMULA
CALL PROCESS	HIDE PROCESS	QUERY SELECTION BY FORMULA
CANCEL TRANSACTION	IMPORT DIF	SELECTION TO ARRAY
CHANGE ACCESS	IMPORT SYLK	Semaphore
CHANGE PASSWORD	INSERT ELEMENT	SEND RECORD
CLEAR SEMAPHORE	Int	Sequence number
CLOSE DOCUMENT	In header	SET BLOB SIZE
Compiled application	In footer	SET PROCESS VARIABLE
COPY ARRAY	Is a variable	SHOW PROCESS
COPY NAMED SELECTION	Is user deleted	Size of array
Count fields	Last object	START TRANSACTION
Count tasks	LAST RECORD	Std deviation
Count user processes	LIST TO ARRAY	Substring
Count users	LOAD RECORD	Sum squares
Create document	Locked	Table

CREATE RECORD	LOCKED ATTRIBUTES	Tickcount
Current process	Max	Time
Current user	Min	Type
C_BLOB	Month of	UNLOAD RECORD
C_TEXT	New process	VALIDATE TRANSACTION
C_TIME	NEXT RECORD	VARIABLE TO BLOB
Date	Not	VARIABLE TO VARIABLE
Day number	Num	Variance
DEFAULT TABLE	ON EVENT CALL	Year of

1.7. Konvence názvů pro proměnné

Pro přehlednost kódu je důležitá konvence názvů proměnných pro metody a funkce. ACI nezavedla žádnou povinnou konvenci, prezentujeme zde však jednu, která je poměrně široce používána v různých publikacích a kurzech. Všechna písmena jsou malá, kromě L označující LongInteger, kde se může plést 1 a malé l.

Všechny array (pole v paměti) začínají“a”. Dvoudimenzionální array začínají “a2” nebo “aa“.

Proměnné mají následující předpony , které jsou rovněž používána pro array po písmenu “a”.

Typ	Proměnná	1D Array	2D Array
String	s	as	a2s
Text	t	at	a2t
Integer	není	ai	a2i
Longint	L	aL	a2L
Real	r	ar	a2r
Date	d	ad	a2d
Time	h	není	není
Boolean	f	af	a2f
Picture	g	ag	a2g
Pointer	p	ap	a2p
BLOB	o	není	není

1.8. Aktivní objekty

Aktivní objekty jsou téměř vždy číselné. Pro odlišení však pro ně používáme vlastní předpony.

Objekt	Předpona
Tlačítko/Button	b
Zaškrtávací políčko/Checkbox	ck
Teploměr/Thermometer	th
Přepínač/Radio Button	rb1; rb2; rb3; sb1; sb2; sb3, etc.

1.9. Konvence názvů pro metody a příkazy a funkce

Vestavěné příkazy jazyka 4D jsou uváděny následovně :

- Vše velkými písmeny – pro příkazy vykonávající konkrétní akci jako CANCEL (provádí akci storno).
- Smíšeně velká a malá písmena – pro příkazy, které vrací hodnotu jako např. Uppercase.

Příkazy se smíšenou konvencí jsou vždy vpravo od operátoru přiřazení.

V žargonu programátorů je tato metoda navracející hodnotu nazývána funkce

Většina programátorů volí následující konvenci. Metody jsou uváděny s velkými písmeny na počátku každého slova „MojeProcedura“. Speciální metody nejčastěji

používané začínají „aa“, aby v seznamu metod byly na počátku. Další dělení je pro metody zvláštního účelu:

- “M_” pro metodu spouštěnou z nabídek
- “P_” pro metodu začínající nový proces
- “E_” pro metodu, která je navržena k ručnímu spouštění pomocí příkazu nebo nabídky Execute.

Kromě toho jsme vytvořili následující konvenci pro segmentaci kódu. Vše přímo přiřazené některým účelům (moduly, proměnné, názvy metod, formulářů) začíná následujícími předponami:

- “ALIAS_” pro komponenty, které zacházejí s tabulkami, nebo zastupují pole.
- “CHO_” pro metody, které jsou pro výběrové seznamy
- “GEN_” pro metody, které jsou plně generické
- “IMPEXP” pro komponenty účastné při exportu a importu
- “INITIALIZE” pro komponenty, které inicializují promměnné nebo procesy
- “KEY_” pro komponenty, které zachytávají klávesy
- “LOCK_” pro komponenty, které obsahují zamykání a odemykání záznamů
- “PROCESS_” pro komponenty účastné při řízení procesů
- “SPE_” pro komponenty zacházející s uloženými procedurami
- “WEB_” pro komponenty zacházející s WEB rutinami
- “WIN_” pro komponenty provádějícími manipulaci s okny

Příklady:
- M_Customers
- P_Customers
- sConstants
- GEN_MyMethod

Když píšete vaše vlastní metody, které jsou funkcemi, je často praktické začínat název písmenem, které reprezentuje typ navracené hodnoty. Tyto písmena jsou pak stejná jako v tabulce výše.

Q. QuickCode Pro™

QuickCode Pro™ (“QCP”) je produkt třetí strany, který může být zakoupen a „zasunut do 4D“. Tento produkt přidává k 4D mnoho dalších rysů, které urychlují psaní procedur a metod. QCP vám poskytne úplnou kontrolu editoru metod pomocí klávesových zkratek.

Natural Intelligence, Inc. věnovatla kopii QCP ACI US, Inc pro účely školení. Tato kopie může být používána pouze pro školící centra. V našem kurzu použijeme několik rysů k urychlení vývoje databáze.

Začneme importem maker při vývoji databáze, umožní nám to nezdržovat se samotným psaním a budeme mít více času na komentáře.

Q.1. Import maker do QCP

1. Vyberte Import/Export Macros… z nabídky the QCP Macros  .
2. Importujte tabulku nazvanou QCP Macros umístěnou ve složce Start With.
3. Uzavřete dialog Import/Export.
4. Vyberte Configure Macros… z nabídky QCP Macros  .
5. Otevřete makro Designer Name a nehraďte, “Your name here” here” vaším jménem.
6. Otevřete makro Designer Identifier. Nahraďte "J" iniciálou vašeho jména. Nahraďte slovo name vaším příjmením
   <> + Iniciála+ _ + vaše příjmení + := False
   Příklad: <>fJ_Steinman := True.

Pozn.: QuickCode Pro je k dispozici jen při školeních s lektorem

2. Prostředí více procesů 4D

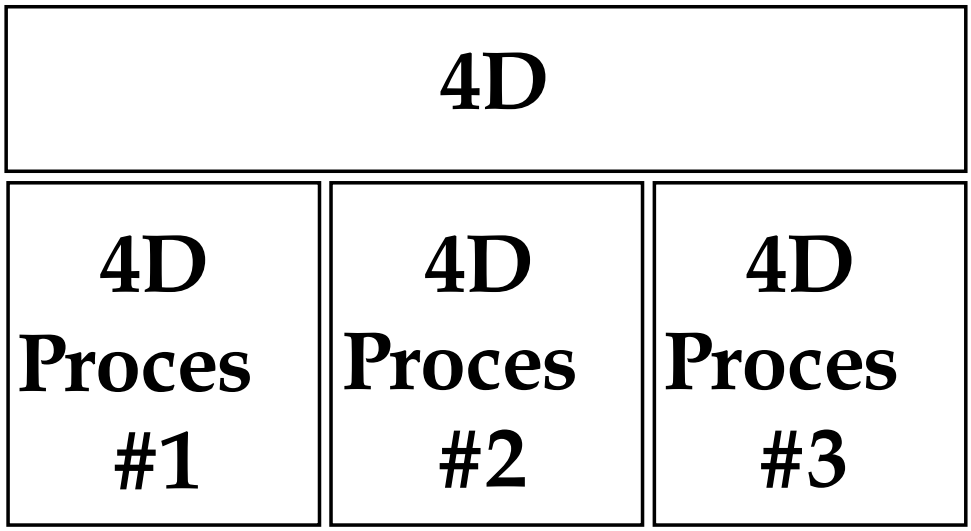
2.1. Procesy

Od verze 3 zavádí 4D koncept více úloh. Schopnost více úloh znamená provádění více než jedné úlohy současně. Koncept více úloh ve 4D závadí cosi nazvané proces.

Proces je jeden výskyt kompletního prostředí 4D. V podstatě více než jedna kopie 4D spuštěná na tomtéž stroji. Tento koncept umožňuje zlepšit využití jednoho počítače, ale přenáší problémy z prostředí více uživatelů na počítač s jedním uživatelem. Každý spuštěný proces musí být uvažován jako oddělený uživatel databáze.

Každý proces ve 4D obsahuje své vlastní prostředí, které zahrnuje následující:

- Platný výběr pro každou tabulku databáze.
- Ukazatel na platný záznam pro každou tabulku databáze
- Platný vstupní a výstupní formulář pro každou tabulku databáze
- Stav číst/psát nebo pouze číst pro každou tabulku databáze
- Výchozí platnou tabulku
- Platné záhlaví nabídek
- Běžící kód (t.j., právě spuštěnou metodu)
- Kompletní sadu proměnných procesu
- Okna (jedno z nich aktivní)
- Dokument nebo sériový port otevřený pomocí SET CHANNEL



2.2. Proměnné a jejich rozměr rozsah

Počítače mají dva typy ukládání dat: trvalý a dočasný. Trvalé ukládání zahrnuje disky, magnetické pásky a další podobná média. Dočasné ukládání obsahuje data zavedená do RAM. 4D má tentýž koncept:

- Data uložená v polích/slouúcích jsou trvalá a jsou uložena na disku
- Proměnné jsou uloženy pouze v RAM

Často potřebujete používat data, která jsou uložena pouze v paměti. Následující je příklad požití proměnné:

```
dDate := Current Date
```

4th Dimension má uloženy všechny názvy a adresy polí používaných v databázi. Když 4D zařazuje název dDate podívá se nejdříve do tabulek. Jestliže dDate není v tabulce, pak 4D ví, že musí vytvořit proměnnou tohoto názvu.

Proměnná je pojmenovaná oblast paměti, do které se dočasně ukládá nějaká hodnota. Hodnota může být kdykoliv změněna.

Proměnné mají další dodatečnou vlastnost, rozsah (scope). Rozsah proměnné odpovídá tomu, kde můžete danou proměnnou využít.

2.2.1. Lokální (místní) proměnné

Lokální proměnné existují pouze v metodě, ve které byly vytvořeny. Mají tu přednost, že zmizí z paměti spolu s metodou po jejím dokončení. Takže jejich použití šetří paměť.

2.2.2. Proměnné procesu

Proměnné procesu jsou dostupné pro více metod jednoho procesu, ale zabírají paměť i když již nejsou potřeba

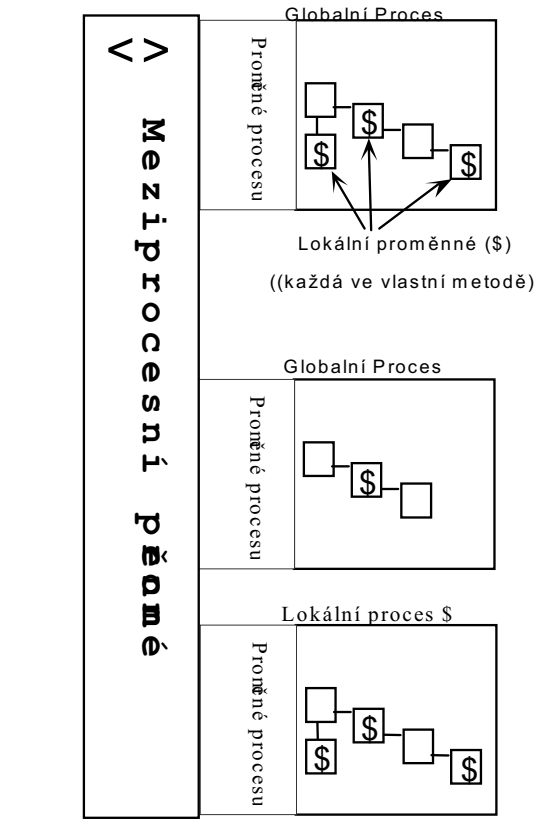
Jestliže se chystáte použít kompilátor, je třeba vědět, že všechny proměné jsou předdeklarovány. Znamená to, že všechny proměnné procesu jsou vytvořeny v době startu/spuštění procesu. Nejsou z paměti vymazány dokud proces neskončí. Proto existují i pokud již nejsou potřeba. Kompletní sada je vytvořena pro každý běžící proces. Tento typ (rozsah) proměnných je pro paměť ejvíc zatěžující.

Každý proces má různou hodnotu pro každou proměnnou procesu. Tak například v procesu uživatele/aplikace dDate muže mít hodnotu !25.1.99! a v jiném uživatelském procesu může mít hodnotu !12.12.98!.

2.2.3 Meziprocesní proměnné

Procesy často potřebují mezi sebou sdílet informace. Například. určité hodnoty mohou zůstávat konstantní dostatečně dlouho, tak že mohou být využity ve více procesech. Meziprocesní proměnné vyplňují tento požadavek. Mají tutéž hodnotu pro všechny procesy. Jestliže jeden proces změní hodnotu, všechy ostatní procesy vidí tuto změněnou hodnotu.

Pro každý počítač, je pouze jeden výskyt meziprocesní proměnné a proto šetří paměť téměř tak jako lokální proměné. Meziprocesní proměnné jsou uloženy v paměti pouze jednoho počítače, proto nejsou sdíleny mezi uživateli 4D Serveru.



2.3. Úlohy pro sledování s použitím meziprocesních proměnných

K sledování některých důležitých věcí, jako například číslo verze, kdy byla procedura změněna, je možno použít meziprocesní proměnné. Jestliže použijete Logockou proměnou, můžete ji nastavit na False (Nepravda) a umístit ji do kódu každé metody. Pomocí 4D Insider je můžete vyhledat a uvidíte tak přehled, v které verzi jste měnili jaký kód a kdo na něm pracoval.

Zahrnuli jsme několik proměnných k rozlišení verze a člověka, který na kódu pracoval.

<>f_Version6x20 pro verzi

a

<>f (Iniciála jména)_ (Příjmení) pro identifikaci osoby

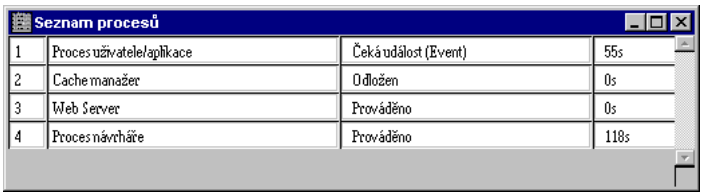
t.j.

<>fK_Wilbur
<>fB_Scott

3. Začínáme s procesy

3.1. Okno Procesu návrháře

1. Spusťte "ACI Video".
2. Přpněte se do Prostředí návrháře.
3. Zvolte Nástroje ¬ Procesy (Průzkumník provádění).



Pro každý proces se v seznamu procesů dozvíte následující informace:

- Číslo ID procesu
- Název procesu
- Současný stav procesu
- Celkový čas provádění procesu v sekundách (využití procesoru) od spuštění procesu

3.2. Číslo ID procesu

Výchozí procesy jsou Proces uživatele/aplikace, Proces vyrovnávací paměti (Cache manager).a Proces návrháře. Tyto procesy jsou obvykle uvedeny na počátku a číslovány 1,2 a 3 dle uvedeného pořadí.

Když spustíte svůj vlastní proces, nebo 4D spustí automatický proces, objeví se tento proces v seznamu procesů s dalším pořadovým číslem, nebo zabere číslo již ukončeného procesu. Čísla ukončených procesů se zabírají odspodu nahoru.

3.3. Název procesu

Proces uživatele/aplikace je výchozí proces při spuštění 4D. Je vždy číslován jako 1. Tento proces nelze ukončit dokud není ukončena samotná 4D. Obsahuje úvodní obrazovku a prostředí uživatele. Jetliže nespustíte nový proces bude celý váš kód prováděn v tomto procesu.

Cache manager je vždy proces číslovaný 2 v prostředí jednouživatelské aplikace. Na 4DClient tento proces neexistuje. Tento proces řídí čtení a zápisy na disk a ukládá data do vyrovnávací paměti samotné 4D.

Proces návrháře existuje pouze tehdy je-li spuštěno Prostředí návrháře.

Další procesy jsou pojmenovány “M_x” nebo “ML_x” pro procesy spuštěné automaticky z nabídky a “P_x” pro procesy spuštěné pomocí Provést metodu. Procesy spštěné procesurálně pomocí příkazu New proces jsou pojmenovány názvem, vámi použitým v příkazu.

3.4. Stav procesu

Stav procesu je jednoduše řečeno odrazem toho co proces právě provádí.Následující tabulka uvádí potenciální stavy procesů a hodnoty těchto stavů.

Hodnota	Stav/Constants	Popis
0	Prováděn	Metoda procesu je prováděna
1	Odložen	Provádění bylo odloženo
2	Čeká událost	Typicky vstup dat
3	Čeká na vstup/výstup	Záznamy převáděny do či z vyrovnávací paměti
4	Čeká semafor	Čeká na přístup do databáze
5	Přerušen	Proces byl přerušen
6	Uschované modální okno	Skrytý proces s modálním oknem
-1	Odstraněn	Proces byl odstraněn
-100	Neexistuje	

3.5. Čas procesu

4D dělí čas CPU mezi existující procesy, takže žádný proces není souvisle vykonáván příliš dlouho. Tento způsob se nazývá časové dělení více úloh.

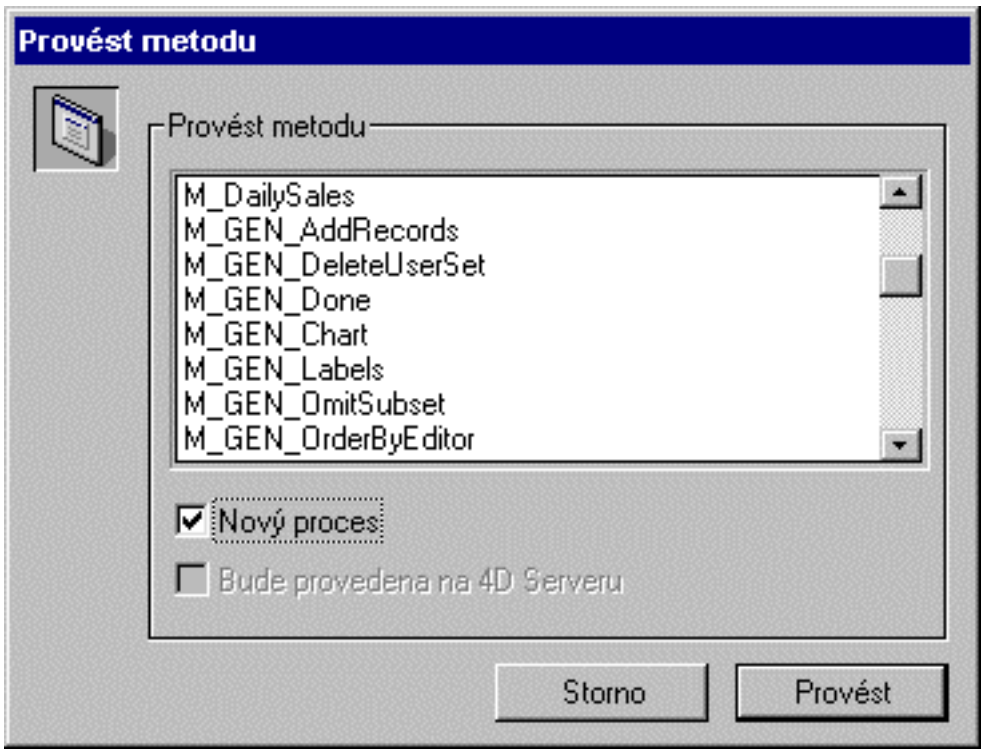
3.6. Spuštění procesu

Spustit proces lze jedním z následujících způsobů.
První je provést něco, čím 4D automaticky zahájí nový proces. Zahrnuje to následné procesy či akce:

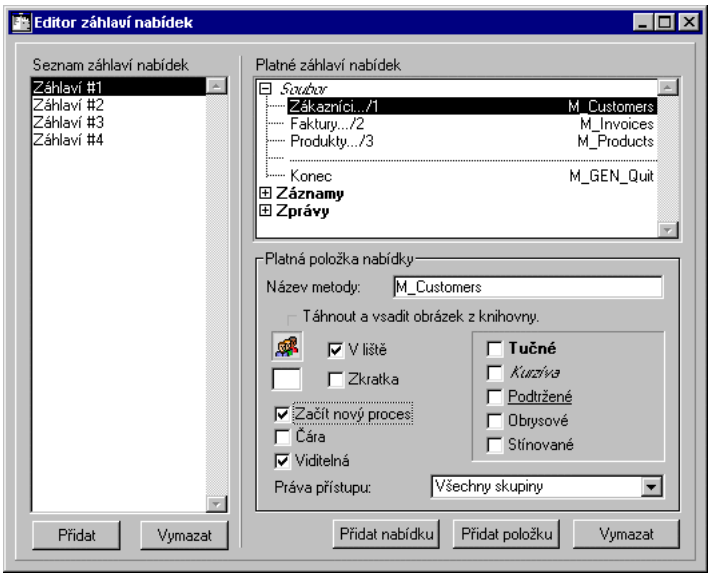
- Proces uživatele/aplikace
- Proces Cache Manager
- Proces Web serveru
- Proces návrháře
- Proces indexování
- Proces Apple Events Manager
- Proces při připojování Web

- Proces ON SERIAL PORT CALL
- Proces ON EVENT CALL (Pozor: nikdy neužívejte příkaz TRACE v tomto procesu)

Druhý, můžete vybrat začít nový proces, při provádění metody z dialogu Provést metodu.



Třetí, můžete vybrat začít nový proces zatržením políčka při vytváření položky nabídky v Editoru nabídek.



Čtvrtý, můžete začít nový proces kódem. Tento poslední způsob vyžadje více práce, než předchozí tři, je však nedůležitější, jestliže chcete dosáhnout plnou kontrolu nad možnostmi 4D ve spouštění a provádění více procesů.

3.7. Není dost místa v paměti ke spuštění procesu

Předpokládejme, že se pokoušíte spustit nový proces buď dialogem Provést metodu nebo pomocí zatržení políčka v Editoru nabídek. Jestliže není dost paměti, nestane se nic. Není zádné varování, žádné zvukové upozornění, nic kromě překreslení obrazovky. Toto chování neodpovídá doporučením Windows Interface a Human Intrface Guidline. Uživatelé Windows a Macintosh očekávají nejakou odezvu pokaždé, když zvolí nějaký příkaz.

První výhoda při spuštění procesu procedurálně je tedy kontrola, zda byl proces spuštěn. Jestliže nebyl spuštěn, můžete informovat uživatele, že není dost místa v paměti.

3.8. Spuštění procesu při spuštění aplikace

Často je vhodné začít plnit určité úlohy, jakmile uživatel spustí aplikaci a přihlásí se úspěsně do databáze. Jestliže se tak nestane, je vhodné uživatele upozornit, že musí ukončit jiné aplikace, nebo zvětšit paměť přidělenou 4D.

3.9. Použití přikazu BEEP k upozornění uživatele.

Když se stane něco neočekávaného, měl by program upozornit uživatele, že je něco nesprávně. Obvykle se tak děje příkazem ALERT. Je-li však něco skutečně v nepořádku

je často používána zvuková výstraha. 4D může tento rys zavést použitím příkazu BEEP. Tento příkaz poskytne zvukovou výstrahu a rovněž tak i vizuální. Pozor: Časté použití zvukových výstražných znamení snižuje citlivost uživatele k tomuto druhu varování. Budeme používat tento druh upozornění k oznámení, že není dost paměti.

3.10. Funkce New Process

Nový proces spustíte procedurálně použitím funkce New process. Když je tento příkaz prováděn 4D spustí nový proces a spustí určenou metodu. Po provedení příkazu se v mateřském procesu pokračuje v provádění dalších příkazů.

New process (NázevMetody; VelikostStack; {NázevProcesu};{*})

Funkce New proces má dva povinné parametry a budeme diskutovat první nepovinný parametr:

- NázevMetody – je název metody, kterou nový proces zahájí provádění.
- VelikostStack – je množství paměti stack přidělené procesu. Stack je místo v paměti používané k uložení objektů procesu, jako volání metod, místní proměnné, parametry volaných metod, záznamy uložené do stack paměti a rekurzivní procedury. Je vyjadřován v bytech a záleží na počtu objektů vytvořených procesem. Velikost od 16000 do 32000 je dostatečná pro většinu procesů- Minimální velikost je 16000 pro lokální procesy, keré nemají přístup k datům databáze a 32000 pro procesy globální s přístupem k datům databáze. Velikost stack nemůže být pod 16000. Jestliže zvolíte číslo menší než 16000, bude stejně použito 16000. (Horní hranice není omezena, ale doporučujeme používat maximálně 256K. Je-li to málo je doporučeno přepsat kód a oplimalizovat jej). Velkost stack je možno zvyšovat po 4000 bytech, i když použijete jiná čísla. Doporučuje, ale zvolit krok po 16000 bytech k zlepšení chování aplikace. V interretované databázi je potřeba větší Stack, v kompilované databázi je využití paěti optimalizováno a je tedy potřeba menší Stack.
- NázevProcesu – je volitelný parametr, kerý by však neměl být považován za volitelný. Je to parametr, který umožňuje identifikovat proces v okně seznamu procesů. Je to rovněž jediný jedinečný parametr, kterým můžete nalézt tento konkrétní proces a řídit jej. Jestliže neurčíte název proces je bez názvu. Název procesu může mít délku do 31 znaků.

3.10.1. Změna okna Denní proeje, tak že se otevře po spuštění

1. Vytvořte novou metodu projektu nazvanou MyStartup, nebo ji importujte ze souboru MyStartup pomocí textového editoru:

```
      ` Metoda: MyStartup
      ` ACI Universita
      ` Vytvořeno: Jim Steinman
      ` Datum:  1/15/97

      ` Účel: Poskytuje proceduru Při spuštění
      ` spustitelnou z dialogu Provést metodu.
      ` potřeba při vývoji a častých změnách metody Při spuštění
C_LONGINT ($Lpid)
C_STRING (80; $sMessage)

   ` rozeznání verzí a autorů
<>f_Version6x00 :=True   ` z kurzu Úvod do 4D
<>f_Version6x10 :=True   ` z kurzu Programování 4D
<>f_Version6x20 := True  `Kurs Více uživatelů, více procesů
<>fJ_Steinman :=True     ` z ACI University
      `<>fGeneric          ` indikace generické procedury

   ` Modified: 1/16/97
<>fK_Wilbur := True

WIN_IsWindows                    ` rozeznat zda běží pod Windows
WIN_InitalizeWindowVariables     ` inicializovat proměnné pro okna

$Lpid := New process ("M_DailySales"; 32000; "DailySalesDialog")

If ($Lpid = 0)
   BEEP
   $sMessage := "Zkuste ukončit jiné aplikace, nebo zvýšit paměť přidělenou této aplikaci."
   ALERT ("K řádnému běhu této databáze není dost paměti. " + $sMessage)
End if

SET WINDOW TITLE ("ACI Video, Inc.")
SET ABOUT ("Co je ACI Video…"; "About")

      `konec metody
```

2. Otevřete metodu databáze Při spuštění a upravte její kód následovně.

```
.  ` Metoda: Při spuštění
   ` ACI Universita
   ` Vytvořeno: Jim Steinman
   ` Datum:  1/15/97
   ` Účel: Nastaví prostředí na výchozí hodnoty uživatele.

MyStartup
   `konec metody
```

3. Přepněte se do prostředí uživatele a zvolte tabulku [Faktury].
4. Vyberte několik faktur a zvolte Dotazy ¬ Vybrat označené.

5. Užijte následující výraz pomocí položky nabídky Vstup ¬ Užít výraz a změňte datum faktury na dnešní datum:

   [Fakrury]DatumFaktury := Current date

6. Restartujte program

4. Události formuláře

Metody formuláře a metody objektu se spouštějí, když nastane určitá událost na úrovni formuláře nebo objektu. Obdobně v prostředí více procesů je umožněno jednomu procesu volat jiný proces. Jestliže chcete, aby určitá metoda formuláře, nebo objektu byla spštěna při určité události musíte provést dvě věci. Zkontrolovat ve vlastnostech objektu, že je zatržena patřičná událost. Musíte napsat kód, který bude při této události spouštěn.

Následující události byly již probrány v předchozích kurzech, zmíníme je zde stručně pro úplnost.

- On Load (Při zavedení) 4th Dimension se chystá zobrazit formulář, nebo při listování záznamy ve vstupním formuláři se chystá zobrazit záznam. (V3 Before)
- On Data Change (Při aktualizaci) Když byla data objektu změněna. (V3 During)
- On Validate (Při potvrzení vstupu) Po přijetí záznamu uživatelem. (V3 After)
- On Close Details (Při uzavření obsahu) 4th Dimension opustila vstupní formulář a chystá se znovu zobrazit výstupní formulář.
- On Printing Break (Při tisku zlomu) Oblast zlomu formuláře se bude tisknout.

Následující události budou probrány v tomto kurzu:

- On Clicked (Při klepnutí) Uživatel klepl na objekt. (V3 During)
- On Double-clicked (Při poklepnutí) Uživatel poklepal na objekt. (V3 During)
- On Drop (Při vsazení) Když danmý objekt přijímá nějaký tažený objekt.
- On Drag Over (Při odtažení) Když nějaký objekt je tažen přes daný objekt.
- On Outside Call (Při vnějším volání) Když formulář obdrží volání přes CALL PROCESS.

Existuje ještě řada dalších událostí. Pokud bychom je chtěli všechny podrobně probrat museli bychom se jim věnovat v samostatném kurzu. Zmíníme je pouze stručně a doporučujeme věnovat se jim v samostudiu.

- On Unload (Při vyvedení) Formulář se bude uzavírat a bude uvolňován žz paměti.

- On Getting Focus (Při získání fokusu) Když objekt získává fokus (t.j. uživatel přešel do objektu pomocí Tab nebo klepnutím myši).
- On Losing Focus (Při ztrátě fokusu) Když objekt formuláře ztrácí fokus (t.j. uživatel klepl na Tab k přejití do dalšího objektu dle pořadí vstupů, nebo klepl na jiný objekt formuláře).
- On Activate (Při aktivaci) Když okno formuláře se stává oknem na popředí (aktivním oknem).
- On Deactivate (Při deaktivaci) Když okno formuláře nebude již dále aktivním oknem (t.j. jiné okno se stává oknem na popředí).
- On Menu Selected (Při výběru nabídky) Byla vybrána nějaká položka nabídky.
- On Keystroke (Při úhozu na klávesu) Když uživatel klepl na klávesu v objektu, kerý má fokus.
- On Plug-In (V oblasti plug-in) V externí oblasti, zajistí spuštění metody této oblasti.
- On Close Box (Při uzavření okna) Bylo klepnuto na uzavírací box okna.
- On Display Details (Při zobrazení obsahu) Záznam bude zobrazen ve výstupním formuláři nebo podformuláři.
- On Open Details (Při otevření obsahu) Záznam ve výstupním formuláři byl poklepán a cyhstá se zobrazit ve vstupním formuláři.

Metody formuláře a objektu jsou prováděny v každé události, kerá je zaškrtnuta pro tuto metodu. Poznámka: Pro jednotlivé typy metod nejsou vhodné všechny typy událostí. Například metody objektů nemohou odpovídat na událost Při vnějšín volání (Konstanta On Outside Call).

Existují rovněž událost pro tisk

- On Printing Header (V záhlaví) Oblast záhlaví formuláře se chystá tisknout.
- On Printing Details (Při tisku obsahu) Oblast obsahu formuláře se chystá tisknout, děje se pro každý tisknutý záznam.
- On Printing Footer (Při tisku zápatí) Oblast zápatí formuláře se chystá tisknout.

4.1. Události a metody

Když nastane nějaká událost, 4th Dimension provede následující akce:

- Nejdříve, projde všechny objekty formuláře a zavolá metody objektů, pro objekty, které byly zvoleny tyto události, a které jsou v události zahrnuty.
- Potom, zavolá metodu formuláře zda byla zvolena odpovídající událost.

Nepředpokládejme, že metody objektů budou volány v nějakém určitém pořadí. Jediné si můžeme pamatovat je, že metody objektů jsou vždy volány před metodou formuláře. Jestliže je objekt podformulář jsou nejdříve provedeny metody objektů podformuláře a pak metoda samotného podformuláře (v seznamu jeden záznam po druhém) a pak teprve 4D pokračuje ve volání metod objektů rodičovského formuláře. Jinými slovy pokud je objekt podformulář platí zde stejné pravidlo jako pro metody formuláře a objektů normálního formuláře.

Jestliže pro určitou událost není zaškrtnuta událost ve formuláři, nezabrání to volání metod objektů pokud je u nich tato událost zaškrtnuta. Jinými slovy zaškrtnutí či odškrtnutí události na úrovni formuláře nemá vliv na události vybrané ve vlastnostech jednotlivých objektů.

Počet objektů zahrnutých do události závisí na vlastnosti události samotné. Například:

- Pro událost On Load jsou to všechny objekty formuláře u nichž je zaškrtnuta událost Při zavedení, a to na libovolné stránce formuláře. Při zatržení ve vlastnostech formuláře události Při zavedení je vždy při této události spouštěna metoda formuláře.
- Při události On Activate nebude spuštěna žádná metoda objektu, protože tato událost se týká formuláře jako celku a ne jeho jednotlivých objektů. Pokud bude zatržena událost ve fromuláři bude se spouštět pouze metoda formuláře.
- Při události On Drag Over budou zahrnuty pouze objekty jejichž vlastnost je vsaditelné a spustí se ty, keré mají tuto událost zaškrtnutu. Metoda formuláře nebude nikdy spuštěna, protože tato událost na úrovni formuláře nemá smysl.

Tabulka níže sumarizuje, pro každý tyo události, jak jsou metody objektů a formulářů volány.

Událost	**Metoda objektu(ů)**	**Formuláře**	**Jaký objekt(y)**
On Load	Ano	Ano	Všechny objekty
On Unload	Ano	Ano	Všechny objekty
On Validate	Ano	Ano	Všechny objekty
On Clicked	Ano (klepnutelné) (*)	Ano	Pouze dotčený
On Double Clicked	Ano (klepnutelné) (*)	Ano	Pouze dotčený

On Keystroke	Ano (dostupné) (*)	Ano	Pouze dotčený
On Getting Focus	Ano (dost.tabelátorem) (*)	Ano	Pouze dotčený
On Losing Focus	Ano (dost.tabelátorem) (*)	Ano	Pouze dotčený
On Activate	Nikdy	Ano	Žádný
On Deactivate	Nikdy	Ano	Žádný
On Outside Call	Nikdy	Ano	Žádný
On Drop	Ano (vsaditelné) (*)	Ano	Pouze dotčený
On Drag Over	Ano (vsaditelné) (*)	Nikdy	Pouze dotčený
On Menu Selected	Nikdy	Ano	Žádný
On Data Change	Ano (dostupné, změny) (*)	Ano	Pouze dotčený
On Plug-in Area	Ano	Ano	Pouze dotčený
On Printing Header	Ano	Ano	Všechny objekty
On Printing Details	Ano	Ano	Všechny objekty
On Printing Break	Ano	Ano	Všechny objekty
On Printing Footer	Ano	Ano	Všechny objekty
On Close Box	Nikdy	Ano	Žádný
On Display Details	Ano	Ano	Všechny objekty
On Open Details	Nikdy	Ano	Žádný
On Close Details	Nikdy	Ano	Žádný

TIP pro optimalizaci: Pamatujte si vždy, že pro libovolnou událost metoda formuláře a objektu je volána vždy když je zatržena odpovídající událost. Výhodou odškrtnutí události v prostředí návrháře (Vlastnosti objektu, stránka události) je, že můžete výrazně snížit počet volání metod a proto výrazně optimalizovat rychlost provádění vašich formulářů.

4.2. Události, objekty a vlastnosti

Metoda objektu je volána, jestliže událost pro daný objekt skutečně může nastat , v závislosti na jeho přirozenosti a vlastnostech. Diskuze níže shrnuje, které události můžete obecně použít pro ovládání různých typů objektů.

4.2.1. Klepnutelné objekty

Klepnutelné objekty jsou objekty, které jsou hlavně ovládány myší. Zahrnují následující objekty:

- Dostupná logická pole a logické proměnné
- Obrázková pole a proměnné, jejichž formát zobrazení byl vybrán Na pozadí.
- Tlačítka, voliče, zaškrtávací políčka, síť tlačítek
- 3D tlačítka, 3D voliče, 3D políčka zaškrtávání
- Sloupcové nabídky, Hierarchický seznam, Obrázkový seznam
- Rozevírací seznam, Nabídka/Seznam
- Posuvné oblasti, Hierarchické seznamy
- Neviditelná tlačítka, Zvýrazněná tlačítka, Obrázková tlačítka
- Teploměry, Pravítka, Výseče
- Řízení karty

Když jsou zaškrtnuty události On Clicked a On Double Clicked pro některý z těchto objektů, můžete rozeznat a ovládat klepnutí na nebo v objektu pomocí příkazu Form event, který vrátí On Clicked nebo On Double Clicked v závislosti na typu klepnutí.

Pro všechny tyto objekty se událost On Clicked vyskytne při uvolnění tlačítka myši s dvěma vyjímkami:

- U neviditelného tlačítka nastane událost On Clicked jakmile je provedeno klepnutí a nečeká se na uvolnění tlačítka.
- U posuvných objektů (Teploměry, Pravítka a Výseče) musí být metoda objektu zavolána okamžitě při posunu řídícího prvku, takže událost On Clicked nastane okamžitě po klepnutí.

Poznámka: Některé z těchto objektů mohou být aktivovány z klávesnice, např. zaškrtávací políčko, pokud má fokus, může být přepínáno pomocí mezerníku. I v takovýchto případech nastává událost On Clicked.

Upozornění: Seznamy s textem (Combo-box) nejsou považovány za klepnutelné objekty. Seznam s textem je považován za vstupní textovou oblast, pro kterou přiřazený seznam slouží jako seznam výchozích hodnot. Proto musíte vstup dat pomocí Seznamu s textem ovládat událostmi On Keystroke a On Data Change.

4.2.2. Objekty dostupné z klávesnice

Objekty dostupné z klávesnice jsou objekty, do kterých vkládáte data pomocí klávesnice a pro které můžete chtít filtrovat vstup dat na nejnižší možné úrovni pomocí události On Keystroke. Jsou to následující objekty:

- Všechny dostupné objekty polí (kromě Obrázek, Podtabulka a BLOB)
- Všechny dostupné proměnné (kromě Obrázek, BLOB, Ukazatel a Array)
- Seznamy s textem
- Hierarchické seznamy

Když zaškrtnete událost objektu On Keystroke pro jeden z těchto objektů, můžete určit a ovládat úhoz na klávesnici uvnitř tohoto objektu za pomoci příkazu Form event, který v tomto případě vrátí On Keystroke .

4.2.3. Změnitelné objekty

Změnitelné objekty jsou objekty, keré mají datový zdroj, který může být změněn s použitím myši nebo klávesnice, a které nejsou považovány za řídící prvky uživatelského rozhraní ovládané přes událost On Clicked. Zahrnují následující objekty:

- Všechny dostupné objekty polí (kromě Podtabulka a BLOB)
- Všechny dostupné proměnné (kromě BLOB, Ukazatel a Array)
- Seznamy s textem
- Externí oblasti (pro které celý vstup dat není přijímán přes 4D Extension)

Toto jsou objekty, které je možno řídit přes událost On Data Change. Když je zaškrtnuta událost On Data Change pro něktrý z těchto objektů můžete určit a ovládat změny dat ve zdroji dat pomocí příkazu Form event, kerý v tomoto případě vrací On Data Change .

4.2.4. Objekty dostupné tabelátorem

Objekty dostupné tabelátorem jsou objekty, které mohou získat fokus při přechodu mezi objekty formuláře pomocí tabelátoru (tyto objekty mohou získat fokus i při klepnutí na ně pomocí myši). Objekty mající fokus jsou objekty, které přijímají znaky napsané z klávesnice (kromě znaků akcelerátorů a klávesových zkratek k nabídkám či tlačítkům) .

Všechny objekty jsou dostupné tabelátorem KROMĚ těchto uvedených níže:

- Nedstupná pole a proměnné
- Tlačítka (používaná pro platformu MacOS)
- Síť tlačítek
- 3D tlačítka, 3D voliče, 3D políčka označení
- Pop-Up Menus, Hierarchical Pop-Up Menus
- Menus/Drop-Down Lists (when used on MacOS platform)
- Obrázkové nabídky

- Posuvné oblasti
- Neviditelná tlačítka, Zvýrazněná tlačítka, Obrázkové voliče
- Diagramy
- Externí oblasti (pro které celý vstup dat není přijímán přes 4D Extension)
- Ovládání karty

Když zaškrtnete událost objektu On Getting Focus anebo On Losing Focus pro jeden z těchto objektů, můžete určit a ovládat změny fokusu uvnitř tohoto objektu za pomoci příkazu Form event, který v tomto případě vrátí On Getting Focus nebo On Losing Focus podle případu.

4.3. Kategorie událostí

Události formuláře mohou být klasifikovány do následujících kategorií:

- obecné události

On Load, On Unload, On Validate, On Display Details

- události příslušející formuláři

On Activate, On Deactivate, On Outside Call, On Close Box, On Menu Selected, On Open Details, On Close Details

- události se vztahem k činnosti uživatele

On Clicked, On Double Clicked, On Keystroke, On Getting Focus , On Losing Focus, On Data Change, On Plug-in Area

- události potáhnout a vsadit

On Drop, On Drag Over

- tiskové události

On Printing Header, On Printing Details, On Printing Break, On Printing Footer

5. Vytvoření Ohlašovacího procesu

Předpokládejme, že chceme neustále zobrazovat formulář, který periodicky, automaticky a pravidelně obnovuje svůj obsah na základě změn v datech.
Když se zobrazí formulář, je proces ve stavu Čeká událost, Waiting for user event. Je-li proces v tomto stavu není vlastní iniciativou schopen provádět nic. K pokračování provádění vyžaduje buď akci uživatele, nebo jiného procesu. Na druhé straně, pokud by jste v tomto procesu v metodě formuláře vytvořili nekonečnou smyčku, která neustále obnovuje data, nebudou tato data na obrazovce nikdy zobrazena, dokud tato smyčka neukončí běh. Jeden způsob jak toto omezení vyřešit je vytvořit jiný proces, jehož zodpovědností je neustálý běh a obnova dat a periodické volání procesu zobrazujícího formulář. Když formulář obdrží vnější volání, nastane událost Při vnějším volání Outside call a formulář se zachová korektně podle požadavků. Abychom to mohli provést musíme procesy řídit vlastními silami.

5.1. Typy procesů

Existují dva typy procesů:

- Lokální proces - je proces prováděný pouze na jednom stroji a dotýkající se svými důsledky a požadavky pouze tohoto stroje (klienta a nebo jednouživatelkého počítače). Tento proces může obsahovat pouze prvky rozhraní a nemá přístup k datům v tabulkách. Data v tabulkách nemohou být lokálním procesem samostatně dotčena. Lokální proces sdílí tatáž data s procesem uživatele/aplikace. To znamená, jestliže jeden lokální proces změní platný výběr, je platný výběr změněn ve všech dalších lokálních procesech a rovněž v procesu uživatele/aplikace. Procesy ON EVENT CALL a ON SERIAL PORT CALL mohou být lokální procesy. Při víceuživatelském běhu se 4D Server neexistuje na 4D server proves odpovídající lokálnímu procesu.

- Globální proces – všechny procesy, keré mají přístup k datům v tabulkách. Při víceuživatelském běhu se 4D Server má každý globální proces svůj protějšek a proces spuštěný na 4D Server.

5.2. DELAY PROCESS(Proces ID; Trvání)

Jestliže vytvoříme proces, který bude fungovat jako budík a necháme jej běžet celý čas, bude 4D dávat tomuto procesu více času CPU, než je potřeba, protože proces běží celou dobu. Abychom tomuto zabránili, můžeme použít příkaz DELAY PROSESS. Když je proces odložen nezabírá žádný čas CPU ani na 4D Serveru ani na uživatelské stanici. Jestliže byl proces již odložen je resetován čítač na požadované trvání. Odložení procesu urychlí zbývající procesy, protože jim bude přiděleno více času CPU.

Odložení procesu je jeho dočasné uspání. Odložený proces se sám probudí na konci požadovaného času. Jestliže proces zobrazuje prvky rozhraní, jsou tyto prvky nefunkční po dobu odložení. Proto není zcela vhodné odkládat procesy zobrazující rozhraní k uživateli. Nejříve tento proces uschovejte.

Všechny příkazy, které vám dovolí řídit proces jsou závislé na číslce ID procesu. Když se proces vytváří je mu přiděleno další dostupné číslo v čítači procesů. Proto se nemůžete nikdy spolehnout na to, že proces bude mít nějaké specifikované zvláštní (vyjma procesu Uživatel/Aplikace). Nikdy se totiž nemůžete spolehnout na to, že uživatel bude provádět vše přesně v tomtéž pořadí pokaždé, když užívá databázi.

Takže nám vzniká určitý problém. Jestliže chcete pracovat s určitým procesem, musíte znát jeho ID. Pro zjištění tohoto ID nabízí 4D čtyři způsoby. Za prvé, můžete toto ID uložit do proměnné v době vytváření procesu funkcí New process. Tato funkce navrací číslo ID právě vytvářeného procesu. Za druhé, kdykoliv vytváříte proces můžete mu přidělit jedinečný název. Pak lze použít způsob vyhledání ID procesu podle jeho názvu. Za třetí, v běžícím procesu můžete zjistit jeho ID pomocí funkce Current process.

Ne všechny procesy mohou být odloženy. Např. hlavní proces, proces Uživatel/Aplikace, nemůže být nikdy odložen. Jestliže potřebujete nutně odložit či zpozdit tento proces, musíte cyklicky provádět něco co zabere čas.

5.3. Current process -> Process ID

Tato funkce navrací ID procesu pro proces, ze kterého je tento příkaz volán. Je to obdoba otázky „Kdo jsem?“. Touto funkcí nezjistíte žádný proces vytvořený v dané metodě, ale zjistíte ID procesu, ve kterém je tato metoda prováděna. Jestliže potřebujete zjistit ID procesu jiného než Current process (tento proces), musíte použít jinou výše zmiňovanou metodu jako např. uložení ID procesu do meziprocesní proměnné.

5.4. CALL PROCESS (Process ID)

Tento příkaz můžeme přirovnat k vytočení čísla na telefonu. To znamená, že tímto příkazem voláme zcela určitý proces. Jestliže není tento proces právě zaneprázdněn a je v něm zobrazen formulář, nastane v něm událost Outside call. Jestliže formulář ve volaném procesu obsahuje patřičný kód, který se má provést ve fázi vnějšího volání, provede se patřičná akce. Tento příkaz nemá žádnou schopnost k přenášení dat a zpráv. Je to pouze schopnost zavolat (zaktivnit) jiný proces. Tento volaný proces pak může provést libovolnou úlohu, která byla naprogramována jako odpověď na událost Při vnějším volání. Jestliže je ID procesu hodnota –1, budou všechny procesy, které zobrazují meziprocessní proměnné, překresleny.

5.5. Process state (Process ID) -> Integer

Příkaz Process state (Stav procesu) navrátí některý z následujících stavů:

Hodnota	Stav/Constants	Popis
0	Prováděn	Metoda procesu je prováděna
1	Odložen	Provádění bylo odloženo
2	Čeká událost	Typicky vstup dat
3	Čeká na vstup/výstup	Záznamy převáděny do či z vyrovnávací paměti
4	Čeká semafor	Čeká na přístup do databáze
5	Přerušen	Proces byl přerušen
6	Uschované modální okno	Skrytý proces s modálním oknem
-1	Odstraněn	Proces byl odstraněn
-100	Neexistuje	

5.5.1. Vytvoření Ohlašovacího procesu k obnově dialogového okna DenníProdeje

1. Nahraďte metodu projektu M_DailySales následující metodou, nebo exportujte pomocí textového editoru:

```
If (False)
      ` Metoda:  M_DailySales
      ` ACI Univerzita kurzy programování
      ` Vytvořeno: Jim Steinman.
      ` Datum:  15/1/97

      ` Účel: Vyhledá všechny dnešní faktury, sečte je a zobrazí výsledek v dialogovém okně.

   <>f_Version6x10 :=True
   <>f_Version6x20 := True
   <>fJ_Steinman :=True

      ` Modified: 16/1/97
   <>fK_Wilbur := True
End if

C_LONGINT ($Lpid; $LWidth; $LHeight; $LWindowID)

MENU BAR (4)

$Lpid := New process ("P_Notifier"; 16000; "$Notifier")  ` Local process w/o interface or
table data

If ($Lpid = 0)
   BEEP
   ALERT ("Není dostatk paměti k otevření okna Denní prodeje.")
Else
   <>LSalesPid := Current process

   $LWidth := 190  `    Výchozí výška.
   $LHeight := 80  `    Výchozí délka.
   $LWindowID:=WIN_LNewWindow ($LWidth; $LHeight; <>WIN_LUpper;
<>WIN_LStandardNoResize)

   GOTO XY (0; 2)
   MESSAGE ("Prosím čekejte." + Char(Carriage return) + "Probíhá výpočet dnešních
prodejů.")

   CalculateDailySales

   DIALOG ([zDialogy]; "Denníprodeje")
   CLOSE WINDOW

End if
```

```
DELAY PROCESS ($Lpid; 120)  ` Nastaví čas prodlení procesu Notifier

` Konec metody.
```

2. Vytvořte novou metodu projektu P_Notifier s následujícím obsahem nebo importujte pomocí textového editoru:

```
If (False)
    ` Metoda: P_Notifier
    ` ACI Univerzita kurzy programování
    ` Vytvořeno: Jim Steinman
    ` Datum: 16/1/97

    ` Účel: Volá proces DailySalesDialog a vyzve ho tak k obnově celkového součtu prodejů.

  <>f_Version6x20 := True
  <>fJ_Steinman :=True
End if

C_LONGINT ($LOneSecond; $LDelay)

$LOneSecond := 60  ` 60 tiků = 1 sekunda
$LDelay := 30

Repeat
  DELAY PROCESS (Current process; $LDelay * $LOneSecond)
  CALL PROCESS (<>LSalesPid)
Until (Process state (<>LSalesPid) = Aborted)

  ` Konec metody
```

První věcí, kterou M_DailySales provede je vytvoření procesu $Notifier, který následně volá DailySalesDialog. Při ukončení procedury M_DailySales tj. při uzavření dialogu DailySalesDialog budeme chtít, aby rovněž skončil proces $Notifier. V opačném případě by tento proces zkoušel neustále volat proces, který již neexistuje. Abychom to zajistili použijeme v procesu $Notifier smyčku Repeat, která neustále testuje stav procesu M_DailySales pomocí ID procesu <>LsalesPid a budeme tuto smyčku opakovat dokud stav procesu není –1 což indikuje že proces s DailySalesDialog byl odstraněn. V tomto okamžiku skončí i proces $Notifier.

DELAY PROCESS ($Lpid; 120) je dobrým způsobem pro ovládání ukončení procesu, který je potomkem daného procesu. Co kdyby se náš proces s DailySalesDialog ukončoval právě tehdy, když proces $Notifier zahájí své odložení. Bude trvat dvě sekundy než provede další kontrolu. Pamatujeme si, že jestliže je DELAY PROCESS voláno na proces, který je již odložen, je použita nová doba trvání. Takže výsledek je, že proces $Notifier bude odložen na 120 tiků (2 sekundy). Tato hodnota byla použita pro to, že procesu trvá určitý čas než se ukončí, takže použitá hodnota odloží proces $Notifier na dostatečně rozumnou dobu pro rychlé ukončení.

3. Vytvořte metodu formuláře pro [zDialogy]; "DenníProdeje" a napište následující kód:

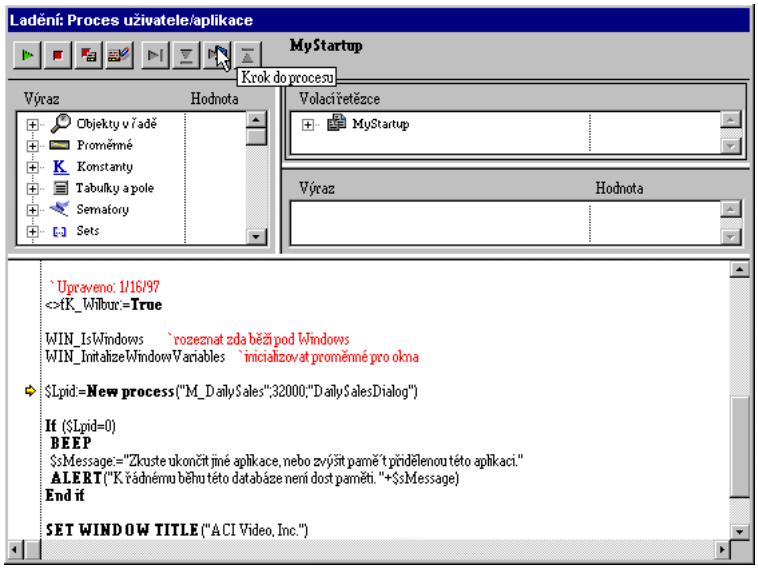
```
If (False)
      ` Form Method.: [zDialogs] "DailySales
      ` ACI Univerzita kurzy programování
      ` Vytvořeno:  Jim Steinman.
      ` Datum: 16/1/97

      ` Účel: Odpovídá na volání Ohlašovacího procesu a obnoví dnešní celkové prodeje.

   <>f_Version6x20 := True
   <>fJ_Steinman :=True
End if

If (Form event = On Outside Call)
   CalculateDailySales
End if

   ` Konec metody
```

4. Upravte metodu projektu CalculateDailySales a odkomentujte příkaz MESSAGES OFF před příkazem QUERY a odkomentujte příkaz MESSAGES ON po příkaze QUERY.

5. Přepněte se do provstředí Vlastní nabídky to. Uzavřete a znovu spusťte okno Denní prodeje. Potom zkuste přidat fakturu.
6. Upravte metodu projektu CalculateDailySales a znovu zaktivněte příkazy MESSAGES OFF před příkazem QUERY a příkaz MESSAGES ON po příkazu QUERY.
7. Přepněte se do prostředí Vlastní nabídky.

6. Jedno okno ladění na proces

Každý proces může mít své vlastní okno ladění. Nové okno ladění otevřete klepnutím na tlačítko Krok do procesu, když je provádění na řádku obsahujícím příkaz New process.



6.0.1. Procvičování otevírání okna ladění na proces

1. Upravte metodu projektu MyStartup následujícím způsobem:

```
` Metoda: MyStartup
` ACI Univerzita kurzy programování
` Vytvořeno: Jim Steinman
` Datum:  15/1/97

` Účel: Poskytuje proceduru Při spuštění
` spustitelnou z dialogu Provést metodu.
` potřeba při vývoji a častých změnách metody Při spuštění

C_LONGINT ($Lpid)
C_STRING (80; $sMessage)

   ` rozeznání verzí a autorů
<>f_Version6x00 :=True    ` z kurzu Úvod do 4D
<>f_Version6x10 :=True    ` z kurzu Programování 4D
<>f_Version6x20 := True  `Kurs Více uživatelů, více procesů
<>fJ_Steinman :=True       ` z ACI University
      `<>fGeneric             ` indikace generické procedury

   ` Modified: 1/16/97
```

```
<>fK_Wilbur := True

WIN_IsWindows                    ` rozeznat zda běží pod Windows
WIN_InitalizeWindowVariables     ` inicializovat proměnné pro okna
```

❖ TRACE

•

```
$Lpid := New process ("M_DailySales"; 32000; "DailySalesDialog")

…
```

2. Proveďte metodu projektu MyStartup z Prostředí uživatele.
3. Po provedení příkazu TRACE se otevře okno ladění. Rozšiřte seznam Procesy v části Výrazy. Tímto způsobem můžete sledovat čísla ID procesu a stavy jednotlivých běžících procesů.
4. Krokujte postupně kódem a na řádce New process klepněte na tlačítko Krok do procesu.
5. Když se objeví další okno ladění opět rozšiřte seznam Procesy.
6. Pokračujte v krokování kódu a zopakujte předchozí postup pro případné další okno ladění.
7. Neprovádějte žádný řádek, který závisí na proměnných, které jsou přiřazeny v jiném procesu před tím než v tomto procesu budou tyto proměnné skutečně přiřazeny.
8. Po skončení krokování odstraňte příkaz TRACE z metody MyStartup.

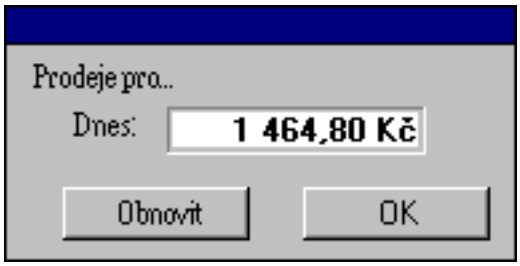
7. Použití plovoucích oken

Jak již název naznačuje jsou to speciální okna, která „plavou“ nad běžnými okny. To znamená, že tento typ okna je vždy aktivní dokonce i tehdy existuje-li jiné aktivní okno.

Jestliže otevřete více oken s plovoucím oknem, máte okna ve dvou různých vrstvách. Plovoucí okna a běžná okna. Plovoucí okno se používá jako paleta nástrojů a ne pro vstup dat. Nikdy nezahrnujte do plovoucích oken oblasti vstupu, protože vstup z klávesnice je odesílán do běžného okna na popředí.

V našem příkladu by bylo dobré, kdyby okno s denními prodeji zůstávalo stále na popředí a nebylo překrýváno okny jiných procesů.

Plovoucí palety mají typ okna –720. Protože pro ovládání oken máme předdefinovány meziprocessní proměnné, použijeme naše vlastní konstanty a meziprocessní proměnnou <>WIN_LfloatingPalette, která má hodnotu –720. Tuto proměnnou budeme používat pro přehlednost kódu a pro otevírání oken typu plovoucích palet.



7.1. Změna dialogu DenníProdeje na plovoucí okno

1. Upravte metodu projektu M_DailySales a existující řádek s WIN_LNewWindow následujícím způsobem:

❖ $LWindowID:=WIN_LNewWindow ($LWidth;$LHeight;<>WIN_LUpper;<>WIN_LFloatingPalette)

2. Přepněte se do prostředí Vlastní nabídky a otestujte plovoucí okno.

7.2. Další proměnné pro plovoucí okna

Existují ještě další typy oken s doplňkovými vlastnostmi.

Typ okna	Popis	Meziprocessní proměnná	Hodnota
-721	Vytvoří přepínací plovoucí okna. Tento typ použijte, když existuje více než jedno plovoucí okno.	<>WIN_LFloatingToggleableCoef	-1
-722	Vytvoří plovoucí okno s nadpisem.	<>WIN_LFloatingTitleCoefficient	-2
-724	Vytvoří plovoucí okno s prvkem pro změnu velikosti.	<>WIN_LFloatingScrollCoefficient	-4
-728	Vytvoří plovoucí okno s posuvníky.	<>WIN_LFloatingZoomCoefficient	-8

Hodnoty lze skládat dohromady. Typ okna se všemi rysy bude tedy -735.

7.3. Použití metody Při uzavření okna

4D vám dovolí vytvořit uzavírací políčko tím, že vložíte název metody jako parametr do příkazu Open window. Tato metoda bude provedena, když uživatel klepne na uzavírací políčko.

Pro přidání uzavíracího políčka do okna může být několik důvodů. Za prvé, vaše aplikace ve 4D bude konzistentní s jinými aplikacemi pro Windows a Macintosh. Za druhé, přidá dodatečné funkce do aplikace. Za třetí, můžete zmenšit velikost okna pokud použijete uzavírací políčko místo tlačítek uzavírajících okno.

7.3.1. Zmenšení velikosti dialogu DenníProdeje

1. Upravte metodu projektu M_DailySales následujícím způsobem:

```
If (False)
    ` Metoda: M_DailySales
    ` ACI Univerzita kurzy programování
    ` Vytvořeno:  Jim Steinman.
    ` Datum:  15/1/97

    ` Účel: Vyhledá všechny dnešní faktury, sečte je a zobrazí výsledek v dialogovém okně.

  <>f_Version6x10 :=True
  <>f_Version6x20 := True
  <>fJ_Steinman :=True

❖   ` Upraveno: 1/16/97 na menší okno s použitím zavíracího políčka a názvu
  <>fK_Wilbur := True
End if

C_LONGINT ($Lpid; $LWidth; $LHeight; $LWindowID)
•   C_STRING (31; $sWindowTitle)          ` Název okna
•   C_STRING (31; $sCloseBoxMethod)    ` Metda zavíracího políčka

MENU BAR (4)

$Lpid := New process ("P_Notifier"; 16000; "$Notifier") ` Místní proces w/o interface nebo
table data

If ($Lpid = 0)
  BEEP
  ALERT ("Není dostatek paměti k otevření okna Denní prodeje.")
Else

  <>LSalesPid := Current process

❖   $LWidth := 140 `    Výchozí delka
❖   $LHeight := 20 `    Výchozí šířka.
❖   $sWindowTitle := "Dnešní prodeje"
❖   $sCloseBoxMethod := "CloseForm"
•   $LWindowID:=WIN_LNewWindow ($LWidth; $LHeight; <>WIN_LUpperLeft;
            <>WIN_LFloatingPalette + <>WIN_LFloatingTitleCoefficient;
            $sWindowTitle; $sCloseBoxMethod)

❖   GOTO XY (2; 1)
•   MESSAGE ("Výpočet prodejů")

  CalculateDailySales

•   DIALOG ([zDialogy]; "DailySales.rev")
```

```
   CLOSE WINDOW
End if

DELAY PROCESS ($Lpid; 120) `Nastaví prodlení procesu Notifier

   ` Konec metody.
```

2. Vytvořte novou metodu projektu CloseForm a napište následující kód:

```
If (False)
      ` Metoda: CloseForm
      ` ACI Univerzita kurzy programování
      ` Generic ACI Shell Programming
      ` Vytvořeno:  Jim Steinman.
      ` Datum: 1/16/97

      ` Účel: Uzavře dialog DenníProdeje.

      ` Called by: DailySales dialog

   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True
End if

ACCEPT

   ` Konec metody
```

3. Přepněte se do prostředí Vlastní nabídky a spusťte metodu DailySales.

8. Zabránění vytvoření nadbytečných procesů

Když uživateli poskytnete možnost více procesů brzy zjistí, že tato aplikace podstatně zvyšuje produktivitu. Avšak několik otevřených oken se stejným názvem může zmást dokonce i zkušeného uživatele. Začátečník bude určitě zmaten již jen možností otevření dvou nebo tří oken najednou, i když budou mít různé názvy. Kromě toho pokud uživatel otevírá okna bez omezení, může brzy vyčerpat dostupnou paměť.

Nejjednodušší způsob jak vyřešit tento problém je dovolit uživateli otevřít pouze jedno okno na tabulku současně. Pokud jsme pečlivě pojmenovali naše procesy, můžeme si poradit bez použití meziprocessních proměnných. Pouze se jednoduše podíváme zda existuje proces s patřičným názvem. Pokud existuje přeneseme jej na popředí. Jestliže neexistuje vytvoříme nový.

8.1. Jedinečné parametry pro New Process

Vytváření nových procesů můžete omezit tak, že zajistíte pouze jeden výskyt procesu s jedinečným parametrem použitým ve funkci New process. Když je tento příkaz prováděn 4D se pak podívá zda již existuje proces s tímto názvem předtím než vytvoří nový proces. Jestliže proces je již spuštěn, funkce navrátí ID tohoto spuštěného procesu.

```
New process (NázevMetody; VelikostStack; {NázevProcesu};{*})
```

Funkce New process má volitelný parametr, který zajistí jedinečnost procesu:

- * Tento volitelný parameter je užitečný pokud chceme zajistit jedinečnost názvu procesu. Jestliže název procesu již existuje pak nový nebude nový proces vytvořen.

8.2. BRING TO FRONT (Process ID)

Jestliže proces zobrazuje okno, které je za jinými okny, je toto okno převedeno na popředí. Je to totéž jak, kdybyste klepli do okna uschovaného na pozadí.

8.2.1. Omezení otevírání více oken téže tabulky pomocí New Process {*}.

1. Upravte název metody projektu M_Customers na P_Customers. Upravte komentář v P_Customers tak, aby odrážel nový název metody.
2. Vytvořte novou metodu projektu M_Customers a napište následující kód: M:

```
If (False)
    ` Metoda: M_Customers
    ` ACI Univerzita kurzy programování
    ` Vytvořeno: Jim Steinman
```

```
      ` Datum: 16/1/97

      ` Účel: Tato metoda vytváří nebo přenáší na popředí proces zákazníků.

   <>f_Version6x20 :=True
   <>fJ_Steinman :=True
End if

C_LONGINT ($Lpid)

$Lpid := New process ("P_Customers"; 32000; "Zákazníci";*)

BRING TO FRONT ($Lpid)

   ` Konec metody
```

3. Otevřete záhlaví nabídek #1 a klepněte na položku Zákazníci v nabídce Soubor.
4. Odškrtněte políčko Začít nový proces.
5. Změňte metody M_Invoices a M_Products tímtéž způsobem a opakujte pro ně kroky 2-4.
6. Přepněte se do prostředí Vlastní nabídky a otestujte nové rysy.

8.3. Více než jedno okno může snížit integritu dat

Zkuste otevřít více než jedno okno DenníProdeje. Přidejte novou fakturu a dočkáte se překvapení. Bylo obnoveno pouze poslední okno DenníProdeje. Je to proto, že používáme meziprocessní proměnnou, abychom řekli obnovovacímu procesu $Notifier, který proces z DailySalesDialog má volat. Pokud je otevřeno více než jedno okno denních prodejů je samozřejmě otevřen stejný počet procesů $Notifier, ale všechny tyto procesy volají jeden proces denních prodejů a to ten poslední. Jestliže toto okno uzavřete, budou ukončeny všechny procesy $Notifier včetně těch, které byly vytvořeny předchozími procesy denních prodejů. Potom, kdy jsou procesy $Notifier ukončeny, neexistuje již žádný obnovovací proces pro obnovu celkových prodejů ve zbývajících otevřených oknech.

Okno DenníProdeje je příkladem procesu, který nesmí mít spuštěnu více než jednu kopii. Jestliže je spuštěno více kopií procesu, uživatel dostává nekorektní informace.

8.3.1. Omezení oken DenníProdeje na jedno

1. Změňte název metody projektu M_DailySales na P_DailySales
2. Změňte název k metodě projektu P_DailySales, aby odrážel změnu názvu.
3. Vytvořte novou metodu projektu M_DailySales a napište do ní následující kód:

```
If (False)
   ` Metoda: M_DailySales
   ` ACI Univerzita kurzy programování
```

```
      ` Vytvořeno: Jim Steinman
      ` Datum: 1/16/97

      ` Účel: Tato metoda vytvoří nebo přenese na popředí proces DailySalesDialog.

   <>f_Version6x20 :=True
   <>fJ_Steinman :=True
End if

C_LONGINT ($Lpid)

$Lpid := New process ("P_DailySales"; 32000; "DailySalesDialog";*)

   ` Konec metody
```

4. Upravte metodu projektu MyStartup na řádku volání M_DailySales a to následovně:

❖ `$Lpid := New process ("P_DailySales"; 32000; "DailySalesDialog";*)`

5. Otevřete záhlaví nabídek #1 a klepněte na položku Denní prodeje v nabídce Zprávy.
6. Odškrtněte políčko Začít nový proces.
7. Otevřete záhlaví nabídek #3 a klepněte na položku Denní prodeje v nabídce Zprávy.
8. Odškrtněte políčko Začít nový proces.
9. Otevřete záhlaví nabídek #4 a klepněte na položku Denní prodeje v nabídce Zprávy.
10. Odškrtněte políčko Začít nový proces.
11. Přepněte se do Prostředí uživatele a proveďte metodu MyStartup.

9. Uschování a ukázání procesů

Spuštění nového procesu zabírá určitý čas. Jestliže máte proces, který bude často používán je podstatně výhodnější uschovat proces a pak jej opět ukázat než jej vždy spustit a ukončit. Tento způsob pak podstatně vylepší chování datbáze, když se proces objeví okamžitě vždy, když je potřeba.

Jestliže daný proces není potřeba pro akce na pozadí lze jej přerušit na dobu kdy je uschován, takže nebude spotřebovávat žádný čas CPU. Pamatujte sei, že provádění přerušeného procesu musí být obnoveno jiným procesem, protože tento krok přerušený proces sám o sobě nemůže udělat. Nakonec si vzpomeňte, že výběry zabírají paměť. Jestliže proces nepotřebuje pracovat se záznamy, měli by jste vymazat výběry z paměti.

9.1. PAUSE PROCESS(Process ID)

PAUSE PROCESS je funkcí velice podobný DELAY PROCESS, zabraňuje procesu využít čas CPU. Rozdíl je v tom, že proces odložený DELAY PROCESS se vzbudí sám po uplynutí požadované doby trvání. Proces přerušený PAUSE PROCESS musí být vzbuzen jiným procesem příkazem RESUME PROCESS.

9.2. RESUME PROCESS(Process ID)

RESUME PROCESS obnoví provádění procesu jehož provádění bylo přerušeno. Volání procesů, příkazem RESUME PROCESS jehož provádění nebylo přerušeno nemá žádný efekt.

9.3. HIDE PROCESS (Process ID)

Funkce tohoto příkazu je velice podobná funkci „Minimalizovat“ ve Windows nebo uschovat aplikaci pod Mac OS. Tento příkaz uschová všechna okna nabídky a další prvky rozhraní určitého procesu. Spustit proces zabírá určitý čas. Jestliže spouštíte proces ve víceuživatelské verzi se 4D Server, musí se rovněž spustit proces na serveru. Jestliže se váš proces chystá zobrazit dialog i otevření okna zabírá určitý čas (výpočet pozice na obrazovce atd.) Jestliže se chystáte často používat tento proces, můžete proces pouze uschovat a vyhnout se tak dodatečným požadavkům na čas při spouštění procesu. Tento rys zajistí příkaz HIDE PROCESS. Kdykoliv budete proces opět potřebovat, použijete přikaz SHOW PROCESS a jediný čas, který ztratíte je čas na překreslení obrazovky. Jestliže nechcete, aby se proces zobrazoval v době svého vytváření, příkaz HIDE PROCESS by měl být prvním příkazem v metodě procesu.

9.4. SHOW PROCESS (Process ID)

Tento příkaz zobrazí proces, který byl uschován. Použití příkazů HIDE PROCESS a SHOW PROCESS nezpůsobuje, že by příkaz byl současně odložen nebo přerušen. Rovněž nemá vliv na přidělený čas procesu na CPU. Proces je stále spuštěn na pozadí, pouze není viditelný. Není neobvyklé, že se uschovává proces, který byl předtím přerušen, aby nezabíral čas CPU.

9.5. Process number (Process name) -> Process ID

Tato funkce může být použita k zjištění zda daný proces je přítomen a spuštěn. Podobně jako rys jedinečnosti, funkce New process jednoduše vrátí ID procesu jestliže proces existuje nebo 0 jestliže neexistuje. V metodě s DailySalesDialog nepotřebujeme provádět nadbytečné řádky kódu, jestliže proces není spuštěn. Na druhé straně, jestliže je spuštěn potřebujeme provést několik kroků kódu. Použijeme funkci Process number ke kontrole zda proces je spuštěn. Jestliže není spustíme jej, jestliže je spustíme kód k jeho zobrazení a provedení naležitých funkcí.

9.6. REDUCE SELECTION ([Tabulka];Číslo)

Tento příkaz zredukuje počet záznamů v platném výběru na zadaný počet. Jestliže se chystáme uschovat proces. Proč by měl zabírat i paměť a pamatovat si platný výběr, když jej budeme současně i přerušovat. Proto použijeme příkaz REDUCE SELECTION k uvolnění paměti zabrané platným výběrem.

9.7 Uschování ukázání a přerušení obnovení procesů

1. Upravte metodu projektu M_DailySales následujícím způsobem.

```
If (False)
    ` Metoda: M_DailySales
    ` ACI Univerzita kurzy programování
    ` Vytvořeno: Jim Steinman
    ` Datum: 16/1/97

    ` Účel: Tato metoda vytvoří nebopřenese na popředí proces DailySalesDialog

  <>f_Version6x20 :=True
  <>fJ_Steinman :=True
End if

C_LONGINT ($Lpid)

$Lpid := Process number ("DailySalesDialog")

If ($Lpid <=0)
  $Lpid := New process ("P_DailySales"; 32000; "DailySalesDialog";*)
Else
  RESUME PROCESS ($Lpid)
  SHOW PROCESS ($Lpid)
  RESUME PROCESS (<>LNotifyPid)
End if

  ` Konec metody
```

2. Vytvořte novou metodu projektu HideDailySales a napište následující kód:

```
If (False)
    ` Metoda: HideDailySales
    ` ACI Univerzita kurzy programování
    ` Vytvořeno: Jim Steinman
    ` Datum: 16/1/97

    ` Účel: Uschová a přeruší proces a okno DailySalesDialog.

    ` Voláno: uzavíracím okénkem DenníProdeje

  <>f_Version6x20 :=True
  <>fJ_Steinman :=True
End if

REDUCE SELECTION ([Invoices]; 0)  `     Uvolní paměť k použití jinde
HIDE PROCESS (Current process)  `    Skryje okno
PAUSE PROCESS (<>LNotifyPid)  `          Pozastaví proces
PAUSE PROCESS (Current process)  ` Poauza k volným tikům
```

```
` Konec metody
```

3. Upravte metodu projektu nazvanou P_DailySales následujícím způsobem:

```
If (False)
      ` Metoda: P_DailySales
      ` ACI Univerzita kurzy programování
      ` Vytvořeno:  Jim Steinman.
      ` Datum:  1/15/97

      ` Účel: Vyhledá všechny dnešní faktury, sečte je a zobrazí výsledek v dialogovém okně.

   <>f_Version6x10 :=True
   <>f_Version6x20 := True
   <>fJ_Steinman :=True

` Upraveno: 1/16/97 na menší okno s použitím zavíracího políčka a názvu
   <>fK_Wilbur := True
End if

C_LONGINT (~~$LPid;~~$LWidth; $LHeight; $LWindowID)
C_STRING (31; $sWindowTitle)          ` Název okna
C_STRING (31; $sCloseBoxMethod)    ` Metoda zvíracího políčka

MENU BAR (4)

<>LNotifyPid := New process ("P_Notifier"; 16000; "$Notifier")

If (<>LNotifyPid = 0)
   BEEP
   ALERT ("Není dostatek paměti k otevření okna Denní prodeje.")
Else

   <>LSalesPid := Current process

   $LWidth := 140 `     Výchozí šířka.
   $LHeight := 20 `     Výchozí výška.
   $sWindowTitle := "Dnešní prodeje"
   $sCloseBoxMethod := "HideDailySales"
   $LWindowID:=WIN_LNewWindow ($LWidth; $LHeight; <>WIN_LUpperLeft;
                  <>WIN_LFloatingPalette + <>WIN_LFloatingTitleCoefficient;
                  $sWindowTitle; $sCloseBoxMethod)

   GOTO XY (2; 1)
   MESSAGE ("Calculating Sales")

   CalculateDailySales

   DIALOG ([zDialogy]; "DailySales.rev")
   CLOSE WINDOW
End if
```

- DELAY PROCESS (<>LNotifyPid ; 120) ` Nastaví časovou prodlevu procesu

  ` Konec metody.

4. Přepněte se do prostředí Vlastní nabídky a proveďte testování.

10. Ovládání spouštění procesů

V našich procesech máme několik řádek vícenásobného kódu. Naše metody M_Customers, M_Invoices, M_Products, a M_DailySales. Kromě toho jestliže proces není spuštěn, metody neupozorní uživatele, že se tak nestalo. Jinými slovy porušujeme pravidla, že je vždy potřeba dát uživateli odezvu na akci.

Nyní již máme vyřešen problém s neomezeným otvíráním oken a otevíráme pouze jedno okno na tabulku. Předpokládejme však že chcete, aby uživateli bylo umožněno otevřít více než jedno okno na tabulku. Tímto způsobem uživatel může provádět porovnání mezi skupinami z téže tabulky na jedné obrazovce v tentýž čas. Např. uživatel může chtít nalézt všechny zákazníky, kteří mají prodeje přes 30 000 a porovnat je se zákazníky, kteří objednali něco během posledních třiceti dní.

Aby jste umožnili více oken na tabulku, potřebujete řídit vytváření procesů tak, že přidělíte každému procesu jedinečný název a název okna.

10.1. Použití Lokálních semaforů

Lokální semafory jsou přepínače sdílené jednotlivými procesy, na tomtéž počítači. Semafor není proměnná, je to pouze přepínač, který existuje nebo neexistuje (zapnuto, vypnuto). Lokální semafor je dostupný všem procesům na téže pracovní stanici. Jeho název musí vždy začínat předponou znaku dolaru ($). Lokální semafory lze použít k sledování operací mezi procesy prováděnými na téže pracovní stanici. Např. lokální semafor může být použit k sledování přístupu k meziprocessnímu array sdílenému všemi procesy jedné pracovní stanice.

Velice často jsou semafory účinnější než meziprocessní proměnné. Semphore je příkaz, který navrací hodnotu True, jestliže semafor existuje. Jestliže semafor neexistuje je vytvořen a funkce vrátí hodnotu False. Jestliže Semaphore vrátí False znamená to, že semafor neexistoval, ale rovněž to znamená, že byl právě vytvořen. Semafory jsou účinnější, protože v době kdy kód kontroluje existenci semaforu jsou automaticky vytvořeny v dalším tiku hodin. To zabraňuje dvěma procesům kontrolovat semafor v tentýž čas.

Meziprocessní proměnné nemohou být použity k řízení provádění kódu při více úlohách, protože se procesy dělí o čas.

Process 1:

```
If (<>Posting)
  ALERT("Nyní nelze potvrdit. Zkuste později.")
Else
<>Posting:=True
  POSTING    `Spustí proceduru potvrzení
<>Posting:=False
End if
```

Process 2:

```
If (<>Posting)
  ALERT("Nyní nelze potvrdit. Zkuste později.")
Else
<>Posting:=True
  POSTING    ` Spustí proceduru potvrzení
<>Posting:=False
End if
```

Jestliže budete krokovat tyto paralelní segmenty kódu současně, objevíte, že proměná <>Posting bude False v obou procesech současně, a že lze v obou procesech projít k provádění metody POSTING. To je hlavní problém při časově děleném provádění více úloh a vyřešíme jej použitím semaforu.

10.2. Použití meziprocessních array

K ukládání informací, které budou potřeba ve více procesech můžete použít meziprocessní array. Vytvoříme několik array k uložení další polohy okna pro tabulku tj. [Zákazníci], [Faktury] nebo [Produkty]. Protože array spotřebuje určtý čas na své vytvoření je běžné, že array jsou zakládána dopředu před svým prvním použitím. Provedeme to umístěním deklarace array do metody MyStartup nebo metody, kterou budeme volat při spuštění libovolného nového procesu. Běžně jsou array zakládány s nulovým počtem prvků. Příkazy pro deklaraci array jsou:

```
ARRAY BOOLEAN (NázevArray; Velikost 1; {Size 2})
ARRAY DATE (NázevArray; Velikost 1; { Velikost 2})
ARRAY INTEGER (NázevArray; Velikost 1; { Velikost 2})
ARRAY LONGINT (NázevArray; Velikost 1; { Velikost 2})
ARRAY PICTURE (NázevArray; Velikost 1; { Velikost 2})
ARRAY POINTER (NázevArray; Velikost 1; { Velikost 2})
ARRAY REAL (NázevArray; Velikost 1; { Velikost 2})
ARRAY STRING (DélkaŘětězce; NázevArray; Velikost 1; { Velikost 2})
ARRAY TEXT (NázevArray; Velikost 1; { Velikost 2})
```

10.3. Find in array (NázevArray; Hodnota; {Start}) -> Číslo

Velice často potřebujete vyhledat v array určitý prvek nebo hodnotu. Velice často umístění hodnoty není známo. K nalezení prvku můžete použít smyčku For, ale 4D nabízí mnohem chytřejší způsob: příkaz Find in array. Tento příkaz vrátí číslo prvku následujícího prvku, který obsahuje hodnotu, kterou jsme zadali v hledání. Jestliže parametr není určen parametr Start, funkce vrátí první výskyt hodnoty v array. Jinak vrátí první výskyt hodnoty za prvkem určeným parametrem start.

10.4. Size of array (NázevArray) -> Číslo

Tato funkce vrátí počet prvků v array.

10.5. PROCESS PROPERTIES (IDProcesu; Název; Stav; Čas)

Příkaz PROCESS PROPERTIES vrátí do třech proměnných název, stav a čas hodnoty určeného procesu. Je podobný funkci Process state, ale vrací všechny tři důležité hodnoty. Obvykle použijete lokální proměnné k uložení těchto dat. Příkaz PROCESS PROPERTIES budete používat k získání názvu procesu, aby jste mohli upravit naše metody, které obnovují názvy oken, kde použijete raději název procesu než název tabulky. Tento způsob dá uživateli určité upozornění, že to jsou odlišná okna a ne totéž okno zobrazené několikanásobně.

10.6. Get pointer(NázevObjektu) -> Ukazatel

Příkaz Get pointer navrátí ukazatel na proměnnou jejíž název jste zadali jako parametr NázevObjektu.

Tento příkaz nepotřebujete pro provedení tohoto úkolu, ale naučím vás váš první trik ve 4D. Příkaz Get pointer je nejpomalejším příkazem. Budeme používat příkaz Get pointer k zpomalení procesu Uživatel/Aplikace.

10.7. Tickcount -> Longint

Tickcount vrátí počet tiků (1/60 sekundy) spotřebovaný od doby spuštění procesu.

10.8. Testování na False

Provést testování na hodnotu True je jednoduché. Napíšeme jednoduše If a logickou hodnotu a jsme hotovy. Testování na hodnotu False je však trochu obtížnější, nejčastěji je používáno If (Hodnota = False), ale tento způsob je občas matoucí v závislosti na tom jaká hodnota je testována (testovanou hodnotou může být např. i výraz A=B). 4D nabízí pro testování na hodnotu False funkci. Touto funkcí je Not. Pro testování je pak použito Not (Hodnota). Výraz If (Not(Hodnota)) navrátí True jestliže hodnota je False a False, jestliže hodnota je True. Je to proto, že funkce Not převrací logické hodnoty.

10.8.1. Metoda PROCESS_LSpawnProcess

1. Vytvořte novou metodu projektu PROCESS_MyDelay následujícím způsobem nebo ji importujte pomocí textového editoru:

```
If (False)
      ` Metoda: PROCESS_MyDelay (longint;longint)
      ` ACI Univerzita kurzy programování
      ` Generic ACI Shell Programming
      ` Vytvořeno: Jim Steinman
      ` Datum: 16/1/97

      ` Účel: Odloží proces o určený proces tiků

   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True
End if

      ` Vytvoření parametrů
C_LONGINT($1;$LProcessID)          ` ProcessID k odložení
C_LONGINT($2;$LDelay)                  ` trvání odložení

      ` Vytvoření místních proměnných
C_LONGINT($LTickcount)
C_POINTER($pDelayTrick)

      ` Přiřazení parametrů k místním proměnným
$LProcessID := $1
$LDelay := $2

If ($LProcessID # <>LMainProcess)
   DELAY PROCESS ($LProcessID;$LDelay)
Else
   $LTickcount := Tickcount+$LDelay
   Repeat
      $pDelayTrick := Get pointer ("LBox1")
   Until (Tickcount>$LTickcount)
End if
  `Konec metody
```

2. Vytvořte novou metodu projektu PROCESS_LNextView nebo ji importujte pomocí textového editoru:

```
If (False)
      ` Metoda: PROCESS_LNextView (string) -> longint
      ` ACI Univerzita kurzy programování
      ` Generic ACI Shell Programming
      ` Vytvořeno: Jim Steinman
      ` Datum: 16/1/97

      ` Účel: Zjistí další pořadové číslo zobrazovaného okna a vrátí jej

   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True
End if

      ` Vytvoření parametrů
C_LONGINT($0)                          ` Next View číslo k vytvoření
C_STRING (15; $1; $sProcessName)    ` Název procesu k vytvoření

      ` Vytvoření místních proměnných
C_LONGINT($LLocation)

      ` Přiřazení parametrů k místním proměnným
$sProcessName := $1

$LLocation := Find in array (<>PROCESS_asViewName; $sProcessName)

If ($LLocation = -1)

   While (Semaphore ("$ModViewArrays"))
      PROCESS _MyDelay(Current process; 1)
   End while

   $LLocation := Size of array (<>PROCESS_asViewName) + 1
   ARRAY STRING (31; <>PROCESS_asViewName; $LLocation)
   ARRAY INTEGER (<>PROCESS_aiViewNumber; $LLocation)
   <>PROCESS_asViewName{$LLocation} := $sProcessName
   CLEAR SEMAPHORE ("$ModViewArrays")
End if

<>PROCESS_aiViewNumber{$LLocation} := <>PROCESS_aiViewNumber{$LLocation} +
1
$0 := <>PROCESS_aiViewNumber{$LLocation}

   ` Konec metody
```

3. Vytvořte novou metodu projektu PROCESS_LSpawnProcess následujícím způsobem nebo importujte pomocí textového editoru:

```
If (False)
      ` Metoda: PROCESS_LSpawnProcess (string; integer; string; bool; bool) -> longint
      ` ACI Univerzita kurzy programování
      ` Generic ACI Shell Programming
      ` Vytvořeno: Jim Steinman
      ` Datum: 16/1/97

      ` Vrací -> longint - proces id, nula poklud proces nebyl vytvořen
      ` ----------------------------------------------------

      ` Účel: Zde je volána funkce New process.
   ` Brání vytvoření procesu, pokud existuje <>fQuit flag.
   ` Tato metoda vytvoří nový proces kontrolovaným způsobem.
   ` Je zde vlastnost která upozorní uživatele že proces nemůže být vytvořen
   ` z důvodu nedostatku paměti. Vývojář kontroluje kdy přiložená
   ` metoda může být spuštěna ve více procesech (multiple views).

   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

End if

      `  Vytvoření parametrů a přeřezení místních proměnných
C_LONGINT($0;$Lpid)                  ` Proces id, nula pokud nebyl vytvořen..
C_STRING(31;$1;$sMethod)             ` Metoda ke spuštění v novém procesu.
C_LONGINT($2;$LStack)                ` Stack velikost (paměť pro procesové proměnné)
C_STRING(31;;$3;$sProcessName)       ` Název vytvořeného procesu.
C_BOOLEAN($4;$fInformUser)           ` Informuje uživatele, že není dostatek paměti k vytvoření
procesu.
C_BOOLEAN($5;$fAllowMultipleViews)   ` (Volitelné) Umožnění více spuštění..

C_STRING(255;$sMessage)              ` Zpráva uživateli o nedostatku paměti..

If (Not(<>fQuit))  ` Nezačít proces, pokud se vypíná databáze
   $sMethod := $1
   $LStack  := $2 * 1024
   $sProcessName  := $3
   $fInformUser :=$4
   if (Count parameters =5)
      $fAllowMultipleViews  :=$5
   Else
      $fAllowMultipleViews  := False
   End if

   $Lpid := Process number ($sProcessName)

   If ($Lpid= 0)  ` Již neběží žádný proces
```

```
      $Lpid := New process ($sMethod; $LStack; $sProcessName)

   Else

      If ($fAllowMultipleViews )  ` Is the multiview flag set
         $Lpid := New process ($sMethod; $LStack; $sProcessName +
            String (PROCESS_LNextView ($sProcessName)))
      Else
         RESUME PROCESS ($Lpid)
         SHOW PROCESS ($Lpid)
         BRING TO FRONT ($Lpid)
      End if

   End if

   If ($Lpid=0)  ` Něco je špatně
      If ($fInformUser )
         $sMessage := " Nedostatek paměti k splnění akce. "
         $sMessage := $sMessage + " Zkuste uzavřít nějaká okna k uvolnění paměti."
         BEEP
         ALERT ($sMessage)
      End if

   End if
   $0 := $Lpid

Else
   ALERT ("Databáze se vypíná.  Nelze začít nový proces.")
   $0 := 0
End if
   ` Konec metody
```

4. Upravte metodu projektu MyStartup následujícím způsobem:

```
   ` Metoda: MyStartup
   ` ACI Univerzita kurzy programování
   ` Vytvořeno: Jim Steinman
      ` Datum:  15/1/97

      ` Účel: Poskytuje proceduru Při spuštění
      ` spustitelnou z položky nabídky Provést metodu.
      potřeba při vývoji a častých změnách metody Při spuštění

C_LONGINT ($Lpid)
C_STRING (80; $sMessage)


   ` rozeznání verzí a autorů
<>f_Version6x00 :=True    ` z klurzu Úvod do 4D
```

```
<>f_Version6x10 :=True   ` z kurzu programování ve 4D
<>f_Version6x20 := True  `kurz víceuživtelský, více procesů
<>fJ_Steinman :=True     ` z ACI Univerzity
     `<>fGeneric         ` indikaze generické procedry

  ` Upraveno: 1/16/97
<>fK_Wilbur := True

WIN_IsWindows                    ` tozeznat, zda běží pod Windows
WIN_InitalizeWindowVariables     ` inicializovat proměnné pro okna
```

- `<>fQuit := False`
- `<>LMainProcess := Current process`

```
$Lpid := New process ("P_DailySales"; 32000; "DailySalesDialog";*)

If ($Lpid=0)
  BEEP
  $sMessage := "Zkuste ukončit jiné aplikace nebozvýšit paměť přidělenou této aplikaci."
  ALERT ("K řádnému běhu této aplikace není dostatek peměti. " + $sMessage)
End if

SET WINDOW TITLE ("ACI Video, Inc.")
SET ABOUT ("About ACI Video…"; "About")
```

- `  Následujcící array jso využity peo více zobrazení`
- `ARRAY STRING (31; <>PROCESS_asViewName; 0)`
- `ARRAY INTEGER(<>PROCESS_aiViewNumber; 0)`
-

```
  ` Konec metody
```

5. Upravte metodu projektu M_Customers následujícím způsobem:

```
If (False)
     ` Metoda: M_Customers
     ` ACI Univerzita kurzy programování
     ` Vytvořeno: Jim Steinman
     ` Datum: 1/16/97

     ` Účel: Tato metoa vytvoří nebo přenese na poředí proces Zákazníci

  <>f_Version6x20 :=True
  <>fJ_Steinman :=True
End if

C_LONGINT ($Lpid)
```

❖ `$Lpid := PROCESS_LSpawnProcess ("P_Customers"; 32; "Zákazníci"; True; True)`

X ~~`BRING TO FRONT ($Lpid)`~~

X
` Konec metody

6. Upravte metodu projektu M_Products stejně jako M_Customers.
7. Upravte metodu projektu M_Invoices následujícím způsobem:

```
If (False)
    ` Metoda: M_Invoices
    ` ACI Univerzita kurzy programování
    ` Vytvořeno: Jim Steinman
    ` Datum: 16/1/97

    ` Účel: Tato metoda vytvoří nebopřenese na popředí proces Faktury

  <>f_Version6x20 :=True
  <>fJ_Steinman :=True
End if

C_LONGINT ($Lpid)

❖ $Lpid := PROCESS_LSpawnProcess ("P_Invoices"; 48; "Faktury"; True; True)

X ~~BRING TO FRONT ($Lpid)~~
X
  ` Konec metody
```

8. Nahraďte text metody WIN_OutputWindowTitle následujícím textem nebo jej importujte pomocí textového editoru:

```
If (False)
    ` Metoda: WIN_OutputWindowTitle
    ` ACI Univerzita kurzy programování
    ` Generic ACI Shell Programming
    ` Vytvořeno: Jim Steinman
    ` Datum: 15/1/97

    ` Účel: Dle potřeby obnovuje název okna

  <>fGeneric :=True
  <>f_Version6x10 :=True
  <>f_Version6x20 :=True
  <>fJ_Steinman :=True

    ` Upraveno: 1/16/97 k použití názvu procesu místo názvu tabulky
  <>fK_Wilbur :=True
End if

C_STRING (31; $sTableName; $sRecordsInSelection; $sRecordsInTable)
C_LONGINT ($LState; $LTime)

PROCESS PROPERTIES (Current process; $sTableName; $LState; $LTime)
```

```
$sRecordsInSelection:= String (Records in selection (pTable->))
$sRecordsInTable:= String (Records in table (pTable->))

SET WINDOW TITLE ($sTableName+ ": " + $sRecordsInSelection+ " z " +
$sRecordsInTable)

   ` Konec metody
```

9. Nahraďte text metody projektu WIN_InputWindowTitle následujícím textem nebo importujte pomocí textového editoru:

```
If (False)
      ` Metoda: WIN_InputWindowTitle
      ` ACI Univerzita kurzy programování
      ` Generic ACI Shell Programming
      ` Vytvořeno: Jim Steinman
      ` Datum: 15/1/97

   `Účel: Tato metoda mění záhlaví nabídek
   `a titul okna pro vstupní formulář.

   `CB: všechny metody vstupních formulářů

   <>fGeneric :=True
   <>f_Version6x10 :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

      ` Upraveno: 1/16/97 k použití názvu procesu místo názvu tabulky
   <>fK_Wilbur :=True
End if

C_STRING (81; $sWindowTitle)
C_STRING (31; $sTableName; $sSelectedRecord;$sRecordsInSelection;)
C_LONGINT ($LState; $LTime)

PROCESS PROPERTIES (Current process; $sTableName; $LState; $LTime)

If (Record number(pTable->)=-3)                    `Toto je nový záznam
   $sWindowTitle:=$sTableName+": Nový záznam"
Else      ` toto je již existující záznam
   $sSelectedRecord :=String(Selected record number(pTable->))
   $sRecordsInSelection :=String(Records in selection(pTable->))
   $sWindowTitle :=$sTableName+": # "+$sSelectedRecord+" z "+$sRecordsInSelection
End if

SET WINDOW TITLE($sWindowTitle )

MENU BAR(4)
```

```
`Konec metody
```

10. Přepněte se do Prostředí uživatele a proveďte metodu MyStartup.

Kvíz

- Potřebujeme nahradit volání New process v metodě M_DailySales za PROCESS_LSpawnProcess?
- Co by se stalo, kdyby uživatel zvolil M_DailySales a nebyl dostatek paměti k spuštění procesu?

11. Proces pro import/export běžící na pozadí

V tomto cvičení vytvoříme uživatelsky přívětivý dialog pro Import/Export, který dovolí uživateli manipulovat s daty téměř, tak jako obdobná funkce v prostředí uživatele. Po té, kdy uživatel zvolí možnosti a zahájí akci, spustíme proces na pozadí, který zvolenou akci provede.

Předtím, než budeme schopni zahrnout tyto rysy, musíme nejdříve probrat několik příkazů a funkcí 4D, které pro toto cvičení budeme potřebovat.

11.1. Příkazy pro práci se strukturou

Určení struktury databáze je obzvlášť důležité při vývoji metod a formulářů, které mohou být použity ve více databázích. Schopnost číst strukturu databáze vám dovolí vyvinout přenositelnou část kódu.

V prostředí vlastní aplikace není k dispozici editor pro Import a Export. Proto si jeden vlastní vytvoříme. Abychom jej mohli vytvořit musíme být schopni číst strukturu a na jejím základě vytvořit patřičná array.

11.1.1. Table (UkazatelNaTabulku) -> Číslo

Tento příkaz navrací pořadové číslo tabulky patřičné k ukazateli na tabulku. Table (Ukazatel na pole) provede totéž. Pořadové číslo tabulky potřebujeme k tomu, abychom byly schopni určit tabulku, pro kterou budeme provádět import či export.

Jiná forma tohoto příkazu je Table (ČísloTabulky), který navrací ukazatel na tabulku.

11.1.2. Count fields (UkazatelNaTabulku) -> Číslo

Tento příkaz navrací počet polí v tabulce určené ukazatelem na tabulku. Count fields (ČísloTabulky) provádí totéž. Tento příkaz je užitečný neboť nám sdělí kolik polí budeme potřebovat zavést do array.

11.1.3. Field (ČísloTabulky; ČísloPole) -> Ukazatel

Tento příkaz navrací ukazatel na pole. Jiná verze tohoto příkazu, Field (UkazatelNaPole), navrací pořadové číslo pole.

11.1.4. Fieldname (UkazatelNaPole) -> Řetězec

Tento příkaz navrací název pole příslušející k ukazateli na pole. Fieldname (ČísloTabulky;ČísloPole) provádí totéž. Tuto informaci potřebujeme, abychom uživateli zobrazili názvy polí, které může pro import či export použít.

11.1.5. GET FIELD PROPERTIES (UkazatelNaPole; Typ; {Délka}; {Index})

Tento příkaz navrací do určených proměnných typ pole, jeho délku (je-li to řetězec typu alfa) a zda je či není pole indexované. Budemé také používat jinou formu tohoto příkazu GET FIELD PROPERTIES (ČísloTabulky; ČísloPole; Typ; {Délka}; {Index})

11.1.6. Type (Pole nebo Proměnná)

Tento příkaz navrací tutéž informaci jako proměnná typ v příkaze GET FIELD PROPERTIES. Následující tabulka ukazuje seznam čísel navracených pro různé typy dat.

Typ dat	Číslo	Typ dat	Číslo
Alfa pole	0	Real array	14
Real	1	Integer array	15
Text	2	Longint array	16
Obrázek	3	Datum array	17
Datum	4	Text array	18
Nedefinováno	5	Obrázek array	19
Logické	6	Ukazatel array	20
Podtabulka	7	Alfa array	21
Integer	8	Logické array	22
Longint	9	Ukazatel	23
Čas	11	Alfa proměnná	24
Array 2D	13	BLOB	30

11.2. Zásuvné metody (plug in)

Zásuvné metody jsou metody, které byly vytvořeny vně 4D. Zásuvné metody jsou obvykle vytvářeny v obecných počítačových jazycích jako Pascal, C, C++. Jsou kompilovány a následně zpřístupněny aplikaci 4D. Zásuvné metody vám dovolí do vašich databází 4D přidat další rozličné funkce.

11.2.1. ACI Pack

Nejčastěji používány modulem plug in je balík zásuvných metod ACI Pack. Tento balík je standartní součástí 4th Dimension a existuje pro verze Windows, Macintosh a Power Macintosh. Výhodou použití ACI Pack je, že shrnuje nejčastěji vyhledávané funkce, které neobsahuje samotná 4D a při existenci verzí pro různé platformy umožňuje rychlou přenositelnost celého kódu na tyto různé platformy.

Obsah ACI Pack

ACI Pack obsahuje 65 zásuvných metod z celkem 10 témat. Následující seznam dává obecný přehled témat a typů funkcí dostupných pomocí ACI Pack.

• Prostředí	Informace o systému a 4th Dimension
• Systémové dokumenty	Ovládání souborů
• Obrázky	Ovládání obrázků
• Zdroje String	Manipulace se zdroji STR#
• Uživatelské rozhraní	Ovládání Uživatelského rozhraní
• Nápověda Windows	Ovládání nápovědy (pouze Windows)
• Utilities	Různé metody
• Písmo	Informace o dostupných písmech
• Bitové operace	Ovládání paměti po bitech
• Schránka	Operace se schránkou

11.2.2. GetFieldInfos (LTableNum;LFieldNum;LRelateTable;LRelateFld;LAttribs;sList) ->Integer

Procedura, kterou se chystáme vytvořit pro naplnění array názvy polí bude používat zásuvnou metodu GetFieldInfos ke kontrole zda je či není pole neviditelné. Potřebujeme ji proto, že i když 4D má vlastnost pole neviditelné, neexistuje uvnitř jazyka 4D příkaz, který můžeme použít k zjištění zda pole je či není neviditelné.

Volání GetFieldInfos provedem s předáním čísla tabulky a pole jako parametrů. Toto volání vrátí hodnotu do paramteru Attribs, který může vypadat takto 3168. Dokumentace k ACI Pack pro příkaz GetFieldInfos zmiňuje, že bit 8 v parametru Attribs bude 1 jestliže je pole neviditelné a 0 jestliže pole je viditelné.

Co nemusí být zřejmé je jak testovat osmý bit hodnoty. Použití operátoru Bit And (&) otestujete bit. Poznamenejme, že bitové operátory zacházejí s binárními čísly tak, že osmý bit je roven desetinné hodnotě 2 na osmou tj. 256. Binární reprezentace 256 bude obsahovat nuly kromě jedničky na osmém bitu. Takže způsob jak testovat hodnotu v bitu 8 je použít operátor BitAnd a jednu hodnotu 256. Jestliže je výsledek 0 pak je osmý bit 0. Jestliže je výsledek 256 pak je osmý bit 1 (viz níže).

11.3. Více příkazů pro práci s array

Jedním způsobem jak změnit velikost array je předeklarovat jej na jinou velikost a tak ji zvětšit či zmenšit. To samozřejmě zapůsobí pouze na prvky na konci array (byl-li array větší a zmenšujeme jej jsou prvky na konci pole zapomenuty a byl-li array menší a zvětšujeme jej přidají se na konec prázdné prvky). Protože však uživatel musí mít možnost určovat i pořadí polí pro Export/Import, je nutno umožnit i přidání prvků doprostřed array.

11.3.1. INSERT ELEMENT (NázevArray; Umístění; {PočetPrvků})Tento příkaz zvětší velikost array o patřičný počet prvků na specifikovaném místě. Jestliže není počet prvků

určen přidá se pouze jeden prvek. Jestliže je zvoleno umístění větší než je velikost array přidají se prvky na konec array.

Budeme používat tento příkaz k tomu, abychom umožnili uživateli vložit pole doprostřed seznamu pro Export/Import pomocí klepnutí na tlačítko. Patřičný kód je již umístěn v metodě objektu tlačítka v databázi.

11.3.2. DELETE ELEMENT (NázevArray; Umístění: {PočetPrvků})

Tento příkaz sníží velikost array o patřičný počet prvků na určeném místě. Jestliže není určen počet prvků bude vymazán pouze jeden prvek. Velikost array se zmenší o počet určených prvků.

Tento příkaz budeme používat k tomu, abychom umožnili uživateli vymazat pole ze seznamu pro Import/Export pomocí klepnutí na tlačítko. Patřičný kód je již umístěn v metodě objektu tlačítka v databázi.

11.3.3. COPY ARRAY (ZdrojovéArray; CílovéArray)

Tento příkaz je používán k duplikování existujícího array. Vytvoří cílové array o stejném obsahu jako má zdrojové array.

Tento příkaz budeme používat pro kopírování konečného array ukazatelů polí určených pro Export/Import do meziprocesního array, které může být přečteno novým procesem. To nám umožní, že nový proces na pozadí bude vědět, která pole použít. Nový proces okamžitě zduplikuje hodnoty tohoto meziprocessního array do lokálního array, které následně použije. Po tomto kroku můžeme již původní meziprocesní array opět použít v jiném procesu.

11.4. Prvky jazyka pro práci s dokumenty

Dříve či později se setkáte s potřebou pracovat s dokumenty na disku. Tyto dokumenty nejsou data tabulek 4D, ale zcela nezávislé dokumenty vytvořené buď vaším programem nebo úplnně jinými aplikacemi.

11.4.1. Open document (Dokument; {Typ}) -> Odkaz na dokument

Tato funkce otevře existující dokument pro čtení či zápis na začátek dokumentu a vrátí číslo, které může být použito jako odkaz na dokument. Data zapsaná na počátek dokumentu, přepíší libovolná již existující data. Odkaz na dokument je datavý typ Čas.

Tento příkaz budeme používat pro otevření existujících dokumentů pro import dat. V jednom procesu může být tímto příkazem otevřeno současně více dokumentů.

Další obdobné příkazy, které můžete chtít postupně použít jsou Create document a Append document.

11.4.2. SET CHANNEL (Operace; Dokument)

Tento příkaz vám dovolí vytvořit, otevřít a uzavřít dokumenty. Z dokumentu je možno číst nebo do něj zapisovat. Oproti předcházejícím příkazům dokumentu může být tímto příkazem otevřen pouze jeden dokument na proces.

Parametr operace určuje akci, která bude s dokumentem prováděna.

Akce	Dokument	Výsledek
10	Řetězec	Otevře či vytvoří určený dokument.
10	"" (prázdný)	Zobrazí dialog pro otevření libovolného souboru
11	Žádný	Uzavře otevřený soubor
12	"" (prázdný)	Zobrazí dialogové okno pro uložení nového vytvořeného souboru
13	"" (prázdný)	Zobrazí dialogové okno pro otevření pouze textových souborů

Tento příkaz je také používán k otevření sériového portu. Pro tento účel jsou používány různé operace a parametry. Neváhejte a samostatně si vyzkoušejte všechny možnosti tohoto příkazu.

V našem kódu budeme ukazovat oba způsoby ovládání dokumentu. Tento příkaz budeme speciálně používat k otevření dokumentu pro export záznamů.

11.4.3. SEND PACKET ({OdkazNaDokument}; Data)

SEND PACKET odesílá určená data do dokumentu (nebo na sériový port). Jestliže byl k otevření dokumentu použit příkaz SET CHANNEL nebo je přímo použit sériový port, není potřeba použít odkaz na dokument, protože omezením příkazu SET CHANNEL je zajištěno, že je otevřen pouze jeden dokument či sériový port. Jestliže byl použit příkaz Open document, odkaz na dokument je vyžadován.

Tento příkaz budeme používat k exportu dat do textového dokumentu. k otevření

11.4.4. RECEIVE PACKET ({OdkazNaDokument}; Proměnná; (ZnakKonce nebo PočetZnaků))

Příkaz RECEIVE PACKET získá data z dokumentu (nebo sériového portu). Jestliže byl použit příkaz SET CHANNEL pro otevření dokumentu nebo sériového portu není

potřeba odkaz na dokument. Jestliže byl použit pro otevření dokumentu příkaz Open document je odkaz na dokument vyžadován. Tento příkaz čte buď všechny znaky do té doby než narazí na znak určený jako znak konce nebo přesně určený počet znaků. Data jsou čtena do proměnné. Vždy je nejlepší mít pro toto čtení použitu proměnnou typu text, protože nikdy nevíte kolik znaků bude tímto příkazem přečteno.

Tento příkaz budeme používat k importu dat z textového dokumentu.

11.4.5. SEND RECORD ({Tabulka})

Tento příkaz používá speciální exportní formát, kerý je čitelný pouze aplikacemi 4D. Bude exportovat záznamy dp dokumentu na sériový port otevřený pomocí SET CHANNEL. Odeslán je celý záznam to znamená, že jsou odeslány všechny podzáznamy a obrázky přiřazené k záznamu.

Soubor vytvořený tímto příkazem, může být čten pouze příkazem RECEIVE RECORD v databázi, která má identickou strukturu tabulky. To znamená, že tabulka musí mít tentýž počet polí, stejné typy polí a jejich pořadí.

Tento příkaz budeme používat s volitelným zaškrtávacím políčkem 4D format v našem dialogu Import/Export..

11.4.6. RECEIVE RECORD ({Tabulka})

Je to speciální formát pro import používaný jinými databázemi 4D. Tímto příkazem importujeme záznamy z dokumentu nebo sériového portu otevřeného pomocí SET CHANNEL. Pomocí tohoto příkazu je přijat celý záznam.

11.4.7. CLOSE DOCUMENT (OdkazNaDokument)

Tento příkaz uzavře dokument vytvořený pomocí Create document nebo otevřený pomocí Open document.

11.4.8. Proměnné dokumentu

Proměnné dokumentu budou obsahovat kompletní cestu k nedávno použitým dokumentům. Dovolíme uživateli vybrat soubor pro proces na pozadí, uzavřít tento soubor a předat obsah proměnných dokumentu do procesu na pozadí a umožníme tak tomuto procesu otevřít určený dokument. Tento způsob urychlí provádění operace se soubory a nebudeme muset čekat na spuštění procesu na pozadí, který by se dotázal na soubor.

11.5. Jiné příkazy k vytvoření procesu pro Import/Export

11.5.1. BUTTON TEXT (NázevTlačítka; TextTlačítka)

Tento příkaz vám umožní změnit viditelný text tlačítka. Text tlačítka může být buď alfanumerický řetězec uzavřený v uvozovkách nebo zdroj typu string.

Tento příkaz budeme používat k změně textu v tlačítku ACCEPT (Přijmout) ve formuláři pro Export/Import na text Import nebo Export v závislosti na tom co budeme provádět.

11.5.2. COPY NAMED SELECTION ({Tabulka; Název)

Tento příkaz vytváří kopii platného výběru tabulky. Pojmenovaný výběr je jednoznačně určován názvem. Příslušný platný výběr tabulky zůstane nedotčen.

Vždy se ujistěte, že máte dostatek paměti k provedení této operace. Paměť požadovaná k uložení platného výběru tabulky bude zdvojena. Tento příkaz nemusí být vždy úspěšně splněn.

Příkaz COPY NAMED SELECTION budeme používat k vytvoření meziprocessního pojmenovaného výběru záznamů, které mají být exportovány v procesu exportu. Tento pojmenovaný výběr předá výběr záznamů do exportního procesu.

11.5.3. CUT NAMED SELECTION ({Tabulka}; Název)

Tento příkaz přemístí platný výběr tabulky do pojmenovaného výběru určeného názvem. Pojmenovaný výběr je svým názvem jednoznačně určen. Po provedení tohoto příkazu je platný výběr patřičné tabulky prázdný.

Protože bude platný výběr tabulky prázdný nevyžaduje tento příkaz žádnou dodatečnou paměť. Proto může být tento příkaz vždy úspěšně dokončen.

11.5.4. USE NAMED SELECTION ({Tabulka}; Název)

Tento příkaz se chová odlišně v závislosti na tom jak pojmenovaný výběr vznikl. Jestliže byl vytvořen pomocí COPY NAMED SELECTION. Zůstane pojmenovaný výběr nadále v paměti nedotčen. Jestliže byl vytvořen pomocí CUT NAMED SELECTION pojmenovaný výběr bude z paměti vymazán. Jinými slovy platný výběr tabulky se změní tak, že bude stejný jako pojmenovaný výběr určený názvem.

11.5.5. CLEAR NAMED SELECTION ({Tabulka}; Název)

Připomeňme si, že pojmenovaný výběr zabírá místo v paměti. Jakmile již není potřeba měl by být z paměti odstraněn. K vymazání výběru z paměti použijete příkaz CLEAR

NAMED SELECTION. Tento příkaz jednoduše vymaže pojmenovaný výběr z paměti a paměť uvolní.

11.5.6. Substring (Zdroj; PrvníZnak {;PočetZnaků}) -> Řetězec

Tento příkaz navrací část obsahu zdrojového řetězce do jiného odděleného řetězce. Oddělený řetězec začne na znaku určeném jako první znak a jeho délka je určený počet znaků.

11.5.7. Get pointer (NázevProměnné) ->Ukazatel

Tato funkce navrátí ukazatel na proměnnou určenou názvem proměnné. Tabulky a pole nejsou jediné položky, které mohou být přístupné pomocí ukazatelů. Generické rutiny mohou také pracovat s proměnnými jestliže použijete jejich ukazatele.

Tuto funkci budeme používat k předávání ukazatelů na pole ukazatelů na pole, která budou importována nebo exportována v procesu na pozadí. Budeme slučovat velké meziprocessní proměnné s různou mírou informací a předávat tuto jednu výslednou meziprocessní proměnnou do nového procesu. Tento způsob omezí počet potřebných proměnných a rovněž sníží pravděpodobnost, že jiný proces bude používat naše meziprocessní proměnné. Následně budeme listovat touto sloučenou proměnnou a extrahovat potřebné informace. Budeme používat Get pointer pro získání ukazatele na správný array.

11.5.8. CREATE RECORD ({Tabulka})

Tento příkaz procedurálně vytvoří nový záznam pro tabulku. Může být použit pouze tehdy jestliže všechna nezbytná pole budou vyplňována pomocí programu. U tohoto příkazu není zobrazováno žádné vstupní okno.

Tento příkaz budeme používat během importu a to k vytvoření nových záznamů, které budou obsahovat importovaná data.

11.5.9. CREATE EMPTY SET ({Tabulka}; Název)

Tento příkaz vytvoří sadu s určeným názvem, která nebude obsahovat žádný záznam.

Budeme jej používat během importu záznamů a to k vytvoření sady všech záznamů, které byly importovány. Na počátku importu nebyl ještě importován žádný záznam, takže potřebujeme prázdnou sadu.

11.5.10. ADD TO SET (Název)

Tento příkaz přidá platný záznam do sady s názvem.

Při provádění importu záznamů nejdříve jeden záznam uložíme a pak jej přidáme do naší importní sady. Když bude import dokončen, můžeme tuto sadu převést do platného výběru tabulky a zobrazit.

11.5.11. Time (Řetězec) -> Čas

Když předáme do této funkce řetězec ve správném formátu HH:MM:SS tak překonvertuje řetězec do hodnoty typu čas. Budeme ji v případě potřeby používat v našich procesech importu a to ke konverzi patřičných řetězců na čas.

11.5.12. FLUSH BUFFERS

Tento příkaz není zcela nezbytný, ale je dobré jej použít při importu většího množství nových záznam. Tento příkaz způsobí, že datový buffer 4D bude zapsán na disk. Buffer je normálně zapisován na disk periodicky podle nastavení předvoleb v Prostředí návrháře nebo i častěji pokud je zaplněn. Vyvolání příkazu FLUSH BUFFERS způsobí, že buffer bude na disk zapsán okamžitě a všechny nové či upravené záznamy budou na disk uloženy.

Při použití tohoto příkazu buďte opatrní. Někteří vývojáři se domnívají, že musí použít tento příkaz pokaždém příkazu SAVE nebo ACCEPT v databázi. To není správně, jediné co tím ve skutečnosti dosáhnou je zpomalení databáze a to zvláště pokud běží pod 4D Server. Vyvolání tohoto příkazu po velkém počtu změn vám však zajistí, že všechny změny budou okamžitě uloženy na disk.

11.5.13. POST KEY (AsciiKódZnaku; Modifikátor)

Tato populární zásuvná metoda byla použita od verze 6 přímo ve 4D. Je používána k simulaci stisku kláves. Jako parametry se předává ASCII kód znaku a číslo odpovídající klávesám Ctrl, Alt, Caps Lock, stisknutí tlačítka myši nebo nic a příkaz vytvoří událost podobnou klepnutí na klávesnici se stisknutými příslušnými modifikátory jakou by vytvořil operační systém.

Tento příkaz budeme používat k simulaci klepnutí na klíčové klávesy pro tlačítka, abychom jednoznačně zajistili, že bylo klepnuto na správné tlačítko při opuštění dialogu pro Export/Import.

11.5.14. DRAG AND DROP PROPERTIES (srcObject; srcElement; srcProcess)

Příkaz DRAG AND DROP PROPERTIES vám umožní získat informaci o zdrojovém objektu, když nastane v nějakém objektu událost On Drop.

- Parametr srcObject je ukazatel na zdrojový objekt, jinými slovy objekt, který byl potažen a puštěn. Poznamenejme, že tento objekt může být různý, ale může být také tentýž jako cílový objekt, jinými slovy objekt, pro který se provádí událost On Drop.
- Jestliže jsou potažená a puštěná data prvkem array (prvek zdrojového objektu je array), pak je parametr srcElement roven číslu tohoto prvku. Pokud objekt není array je srcElement roven -1.
- Operace potáhnout a pustit se může provést mezi procesy. Parametr srcProcess je roven číslu procesu, do kterého patří zdrojový objekt. Je důležité testovat hodnotu tohoto parametru. Můžete odpovědět na „potáhnout a pustit uvnitř procesu“ jednoduše, zkopírováním zdrojových dat do cílového objektu. Na druhou stranu, když je provedeno „potáhnout a pustit mezi procesy“ budete muset použít příkaz GET PROCESS VARIABLE, aby jste získali zdrojová data ze zdrojového procesu. V tomto nebudeme tento případ probírat.

Jestliže použijete příkaz DRAG AND DROP PROPERTIES, když nenastala žádná událost potáhnou a pustit srcObject navrátí prázdný ukazatel, srcElement navrátí –1 a srcProcess navrátí 0.

Tento příkaz budeme používat, abychom umožnili přetažení položek z jedné strany dialogu Export/Import a abychom umožnili změnu pořadí pro export.

11.5.15. Drop position -> Číslo

Příkaz Drop position navrátí číslo prvku array do kterého byl objekt potažen a puštěn.

11.5.16. VARIABLE TO BLOB (MáProměnná; NázevBlobu{; offset | *})

Příkaz VARIABLE TO BLOB uloží proměnnou s názvem MáProměnná do BLOB NázevBlobu.

Jestliže uvedete volitelný parametr * , bude proměnná přidána k BLOB a velikost BLOB bude patřičně zvětšena. S použitím parametru * můžete postupně ukládat libovolný počet proměnných nebo seznamy v BLOB a to do té doby dokud se BLOB vejde do paměti.

Pokud neurčíte volitelný parametr * nebo proměnný parametr offset pak bude proměnná uložena na počátek BLOB a přepíše tak libovolný předchozí obsah. Velikost BLOB bude určena podle výsledků zápisu.

Jestliže předáte proměnný parametr offset, proměnná bude zapsána na patřičné místo vyjádřené číslem offset (počet bytů, číslování začíná od nuly) dovnitř BLOB. Velikost BLOB se změní především podle místa do kterého proměnnou vkládáte (plus velikost proměnné). Nově vytvořené byty před místem, do kterého jste zapsali jsou iniciovány na nulu.

Po volání příkazu se vrátí proměnný parametr offset zvětšený o počet bytů, které jste zapsali. Proto můžete tuto proměnnou znovu použít s jiným příkazem zápsi do BLOB a zapsat jinou proměnnou či seznam.

VARIABLE TO BLOB přijme libovolný typ proměnné (včetně jiných BLOB) kromě následujících:

- Ukazatel
- Array ukazatelů
- Dvourozměrný array
- Odkazové Long Integers (okna a hierarchické seznamy)

UPOZORNĚNÍ: Jestliže používáte BLOB pro ukládání proměnných musíte později pro zpětné čtení obsahu BLOB použít příkaz BLOB TO VARIABLE, protože proměnné jsou ukládány v BLOB v interním formátu 4D.

Po volání pokud byla proměnná úspěšně zapsána je systémová proměnná OK nastavena na 1. Jestliže operace nemůže být provedena je systémová proměnná OK nastavena na 0; např. není-li dostatek paměti.

11.5.17. BLOB TO VARIABLE (NázevBlob; MáProměnná{; offset})

Příkaz BLOB TO VARIABLE přepíše obsah proměnné daty uloženými uvnitř BLOB od bytu offset určeného parametrem offset.

Data v BLOB musí být konzistentní s cílovou proměnnou. Typicky budete tento příkaz používat pro BLOB naplněný pomocí příkazu VARIABLE TO BLOB.

Jestliže neurčíte volitelný parametr offset budou data pro proměnnou čtena od počátku BLOB (offset=0). Jestliže zacházíte s BLOB, ve kterém bylo uloženo několik proměnných musíte předat parametr offset a kromě toho jej musíte naplnit číselnou hodnotou. Před voláním příkazu nastavte číselnou hodnotu na správný offset. Po volání

tatáž proměnná vrátí číselnou hodnotu offset pro další proměnnou uloženou uvnitř BLOB.

Pokud byla proměnná úspěšně přepsána je po volání nastavena systémová proměnná OK na 1. Jestliže operace nemohla být provedena je systémová proměnná OK nastavena na 0; např. když nebyl dostatek paměti.

11.5.18. SET BLOB SIZE (blob; Velikost{; Výplň})

SET BLOB SIZE změní velikost BLOB na hodnotu předanou parametrem velikost.

Jako výchozí jsou nově vytvořené byty (jsou-li nějaké) v BLOB inicializovány na 0x00. Jestliže chcete, aby byly tyto byty inicializovány na jinou hodnotu, předejte pomocí parametru výplň hodnoty 0..255.

Když jste skončili práci s BLOB je dobré uvolnit zabranou paměť. Paměť uvolníte nastavením velikosti BLOB na nula.

11.5.19. Při poklepnutí

Událost formuláře On Double Click může být použita k určení zda uživatel na položku poklepnut.

Budeme ji používat v dialogu k určení zda uživatel poklepl na název pole a tím je zahrnul čí odstranil z array pro import/export.

11.5.20. Bitové operace

Bitové operátory zacházejí s výrazy Long Integer nebo hodnotami. My budeme potřebovat operaci BitAND k nalezení pole, které je neviditelné.

Poznámka: Jestliže předáte do bitového operátoru hodnotu Integer nebo Real, 4th Dimension vyhodnotí tuto hodnotu jako Long Integer a pak teprve provede operace s použitím bitového operátoru.

Když používáte bitové operátory, musíte přemýšlet o hodnotě Long Integer jako o array o 32 bitech. Bity jsou číslovány od 0 do 31 zprava doleva.

Protože každý bit může být roven 0 nebo 1, můžete přemýšlet o hodnotě Long Integer jako o hodnotě, která ukládá 32 logických hodnot. Bit rovný 1 znamená True a bit rovný 0 znamená False.

Výraz, který užívá bitové operátory vrací hodnotu Long Integer, kromě operátoru Bit Test, kde výraz vrátí logickou hodnotu. Následující tabulka uvádí seznam bitových operátorů a jejich syntaxe:

Operace	Operátor	Syntaxe	Vrací
Bitwise AND	&	Long & Long	Long

Každý výsledný bit je logické AND bitu ze dvou operandů.

Zde je uvedena tabulka výsledků operace logické AND:

1 & 1 1
0 & 1 0
1 & 0 0
0 & 0 0

Příklad:
0x0000FFFF & 0xFF00FF00 -> 0x0000FF00

11.6. Předávání parametrů procesům

Od 4D verze 6 mohou být procesům předávány parametry v době spuštění procesu. Protože uživatel nastavuje celou řadu možností a parametrů v procesu na popředí, přeneseme tyto předvolby do BLOB a předáme je do nového procesu jako parametr v době spuštění tohoto procesu. Předávání parametrů novému procesu má svá omezení. Musíte použít hodnoty. Jinými slovy nejsou povoleny žádné odkazy. Co jsou to odkazy? Ukazatele, odkazy okna, odkazy na hierarchické seznamy atd.

11.7. Vytvoření procesu na pozadí

Mnoho úloha na počítači zabere poměrně významný čas. Úlohy těchto typů zahrnují import, export a některé druhy zpráv. Tyto úlohy obvykle uzamknou počítač a znemožní jeho použití pro cokoliv jiného. S použitím více procesů a tím, že zajistíme, že některé z těchto procesů budou spuštěny na pozadí umožníme, že část času CPU může být uvolněna a uživatel může pokračovat v práci na počítači s jinými úlohami než se tyto čas zabírající procesy dokončí na pozadí.

11.7.1. Vytvoření procesu, který může být spuštěn na pozadí

1. Vytvořte novou metodu projektu INITIALIZE_Characters následovně a nebo ji importujte pomocí textového editoru:

```
If (False)
    ` Metoda: INITIALIZE_Characters
```

```
   ` ACI Univerzita kurzy programování
   ` Generic ACI Shell Programming
   ` Vytvořeno: Jim Steinman
   ` Datum: 16/1/97

   ` Účel: Přiřadí běžně používané znaky do meziprocessních proměnných

  <>fGeneric :=True
  <>f_Version6x20 :=True
  <>fJ_Steinman :=True

End if

<>sComma := Char (44)  ` ,
<>sQU := Char (34)  `    "
<>sSP := " "  `              Space

  ` Konec metody
```

2. Přidejte následující kód do metody projektu MyStartup.

```
❖                INITIALIZE_Characters
a                <>LBackground := 0  ` Počet procesů běžících na pozadí tak že nemůžeme skončit dokud běží
```

3. Vytvořte novou metodu projektu GEN_FieldsToArray následovně nebo ji importujte pomocí textového editoru:

```
If (False)
    ` Metoda: GEN_FieldsToArray (table ptr; array ptr; array ptr; boolean; boolean)
    ` ACI Univerzita kurzy programování
    ` Generic ACI Shell Programming
    ` Vytvořeno: Jim Steinman
    ` Datum: 16/1/97

    ` Účel: Kopíruje názvy polí do array
    ` Kopíruje ukazatele k polím do druhého array
    ` Obrázková pole, BLOBy a podtabulky nejsou kopírovány.

  <>fGeneric :=True
  <>f_Version6x20 :=True
  <>fJ_Steinman :=True

End if

    ` Vytvoření parametrů a přeřazení místních proměnných
C_POINTER($1;$pTable)              ` Ukazatel k tabulce ve které jsou pole.
C_POINTER($2)                      ` Ukazatel k array s názvy polí.
C_POINTER($3)                      ` Ukazatel k array s ukazateli k polím.
C_BOOLEAN($4;$fIncludeInvisible)   ` (Volitelné) Pokud pravda, obsahuje neviditelná
pole.
C_BOOLEAN($5;$fOnlyIndexed)        ` (Volitelné) pokud pravda, pouze kopie
indexovaných polí.

    ` vytvoření místních proměnných
C_POINTER($pField)                     ` ukazatel k polím
C_LONGINT($i)                      ` opakovací počítadlo
C_LONGINT($LType)                      ` druh pole
C_LONGINT($LLength)                    ` délka pole (pokud je řetězec)
C_LONGINT($LTableNumber)           ` číslo platné tabulky
C_LONGINT($LParameters)                ` počet parametrů přiřazených k metodě
C_LONGINT($LSizeOfArray)               ` velikost array
C_BOOLEAN($fFieldIndexed)              ` pole indexované
C_BOOLEAN($fFieldInvisible)            ` pole neviditelné

    ` vyžadované parametry pro External GetFieldInfo
C_LONGINT($LProperties)            ` předvolb yo poli obsahující neviditelné
C_LONGINT($LRelatedTable)          ` číslo vzatžené tabulky
C_LONGINT($LRelatedField)              ` číslo vztaženého pole
C_LONGINT($LResult)                    ` výsledek funkce
C_STRING(30;$sChoiceList)              ` Chvýběrový seznam přiřazený k poli

    ` definování místních array
$LSizeOfArray:=0
```

```
ARRAY STRING(31;$asFieldNames;$LSizeOfArray)          ` nastavení array na nula prvků
ARRAY POINTER($apFieldPointers;$LSizeOfArray)
ARRAY INTEGER($aiFieldNumbers;$LSizeOfArray)

    ` nastavení výchozích hodnot
$fIncludeInvisible:=False
$fOnlyIndexed:=False

$LParameters:= Count parameters

    ` přiřazení parametrů k proměnným
$pTable := $1
If ($LParameters>3)
  $fIncludeInvisible :=$4
  If ($LParameters>4)
    $fOnlyIndexed :=$5
  End if
End if

$LTableNumber:= Table ($pTable )  `číslo tabulky kterou používáme

For ($i; 1; Count fields ($pTable ))

  $pField := Field ($LTableNumber; $i)  `    Ukazatel k poli

  GET FIELD PROPERTIES($pField;$LType;$LLength;$fFieldIndexed)

  If (($LType # Is Picture) & ($LType # Is BLOB) & ($LType # Is Subtable))
                                                ` ne obrázkové pole, BLOB nebo
podtabulky

    If ((Not ($fOnlyIndexed)) | (($fOnlyIndexed) & ($fFieldIndexed)))` jestliže indexovaný
nebo obsahuje všechna pole bez ohledu na indexy

      $LResult:=GetFieldInfos
($LTableNumber;$i;$LRelatedTable;$LRelatedField;$LProperties;
                $sChoiceList)
      $fFieldInvisible :=(($LProperties & 256)=256)          ` bitAND  256=
neviditelné 0=Normální

      If (($fIncludeInvisible) | (Not($fFieldInvisible)))          ` obsahuje jestliže není
neviditelný nebo not checking
        $LSizeOfArray:=$LSizeOfArray+1
        ARRAY STRING(31;$asFieldNames;$LSizeOfArray)
        $asFieldNames{$LSizeOfArray} := Field name ($pField)
        ARRAY POINTER($apFieldPointers;$LSizeOfArray)
        $apFieldPointers{$LSizeOfArray} := $pField
        ARRAY INTEGER($aiFieldNumbers;$LSizeOfArray)
        $aiFieldNumbers{$LSizeOfArray} := $i
      End if
```

```
      End if

   End if

End for

COPY ARRAY($asFieldNames;$2->)          ` přeřadí místní array zpět k originálu
$LType := Type($3->)
If ($LType=Pointer array)                    ` vrací array ukazatelů k polím
   COPY ARRAY($apFieldPointers;$3->)
Else                                    ` vrací array čísel polí
   COPY ARRAY($aiFieldNumbers;$3->)
End if
   ` Konec metody
```

4. Vytvořte novou metodu projektu GEN_HandleArraysDragAndDrop následovně nebo ji importujte pomocí textového editoru:

```
If (False)
    ` Metoda: GEN_HandleArraysDragAndDrop (Pointer) -> Boolean
    ` ACI Univerzita kurzy programování
    ` Generic ACI Shell Programming
    ` Vytvořeno:  Kent Wilbur
    ` Datum: 16/1/97

  <>fGeneric :=True
  <>f_Version6x20 :=True
  <>fK_Wilbur :=True

End if

    ` vytvoření parametrů
C_BOOLEAN($0)                   ` Drag & Drop je povoleno
C_POINTER($1)                   ` ukazatel k výsledné array

    ` vytvoření místních proměnných
C_LONGINT($LSourceElement)      ` vybraný zdrojový prvek
C_LONGINT($LDestinationElement) ` cílový prvek
C_LONGINT($LWorkingElement)     ` pracující cílový prvek
C_LONGINT($LSourcePid)          ` ID zdrojového procesu
C_LONGINT($i)                   ` opakovací počitadlo
C_LONGINT($LNumberofArrays)     ` počet array které používáme

$LNumberofArrays := Int(Count parameters/2)+1  ` počet párů array

ARRAY POINTER($apSourceObject; $LNumberofArrays)  ` ukazatel k zdrojovým objektům
ARRAY POINTER($apDestinationObject; $LNumberofArrays)  ` ukazatel k cílovým
objektům

  ` přiřazení parametrů k proměnným
$apDestinationObject{1} := $1
If ($LNumberofArrays > 1)
  For ($i;2;$LNumberofArrays)
    $apSourceObject{$i} := ${($i-2)*2+2}
    $apDestinationObject{$i} := ${($i-2)*2+3}
  End for
End if

    ` informace o zdrojovém objektu potažení
DRAG AND DROP PROPERTIES ($apSourceObject{1};$LSourceElement;$LSourcePid)

    ` číslo cílového prvku
$LDestinationElement :=Drop position
    ` Je drag an drop ze stejného array ve stejném procesu?
```

```
$0 :=($LSourcePid = Current process) & ($apSourceObject{1} = $apDestinationObject{1})

If ($0)
        ` jestliže prvek nebyl položen sámna sebe
  If ($LDestinationElement # $LSourceElement)

        ` uložit přetažený prvek do prvku nula
    For ($i;1;$LNumberofArrays)
        ` přiřadit cílový prvek k proměnné pracujícího prvku
      $LWorkingElement := $LDestinationElement

      $apDestinationObject{$i}->{0} := $apDestinationObject{$i}->{$LSourceElement}

        ` vymazat přetažený prvek
      DELETE ELEMENT ($apDestinationObject{$i}->;$LSourceElement)

        ` jestliže byl cílový prvek nad přetaženým prvkem
      If ($LWorkingElement > $LSourceElement)

        ` zmenšení čísla cílového prvku
        $LWorkingElement := $LWorkingElement-1
      End if

       ` jestliže se tažení a položení stalo nad posledním prvkem
      If ($LWorkingElement = -1)

        ` nastavení čísla cílového prvku na nový prvek na konci array
        $LWorkingElement:= Size of array($apDestinationObject{$i}->)+1
      End if

        ` vložit tento nový prvek
      INSERT ELEMENT ($apDestinationObject{$i}->;$LWorkingElement)

        ` nastavit hodnotu prvku, která byla před tím uložena v prvku nula
      $apDestinationObject{$i}->{$LWorkingElement} := $apDestinationObject{$i}->{0}

        ` tento prvek bude nový označený prvek v array
      $apDestinationObject{$i}-> := $LWorkingElement

    End for
  End if

Else

  If ($LSourcePid =  Current process)        ` prvek pochází z jiného array
        ` byl prvek přiřazen k array
    For ($i;1;$LNumberofArrays)
        ` přiřadit cílový prvek k proměnné pracujícího prvku
      $LWorkingElement := $LDestinationElement

      If ($LWorkingElement<1)
```

```
            $LWorkingElement := Size of array($apDestinationObject{$i}->)+1
         End if

               ` vložit tento nový prvek
         INSERT ELEMENT($apDestinationObject{$i}->;$LWorkingElement )
         $apDestinationObject{$i}->{$LWorkingElement } := $apSourceObject{$i}-
>{$LSourceElement}
      End for

   End if
End if
   ` Konec metody
```

5. Vytvořte novou metodu projektu IMPEXP_ImportExportDialog následovně nebo ji importujte pomocí textového editoru:

```
If (False)
      ` Metoda: IMPEXP_ImportExportDialog (string)
      ` ACI Univerzita kurzy programování
      ` Generic ACI Shell Programming
      ` Vytvořeno: Jim Steinman
      ` Datum: 16/1/97

      ` Účel: připraví Import Export.
      ` všechny záznamy v platném výběru jsou exportovány.

   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

End if

      ` Přiřazení parametrů
C_STRING(6;$1;$sÚčel)     ` "Import" nebo "Export"

      ` Vytvoření místních proměnných
C_LONGINT($Lpid)                  ` ProcesID
C_LONGINT($LWidth)                ` Šířka okna
C_LONGINT($LHeight)                     ` Výška okna
C_LONGINT($LWindowID)                   ` ID okna
C_LONGINT($LFormat)                     ` Druh formátu použitého při exportu
C_LONGINT($LTableNumber)
C_STRING(31;$sProcessName)        ` Název procesu ke spuštění
C_BLOB($oParameters)                    ` Blob do kterého se přiřadí parametry exportu

      ` Přiřazení parametrů k proměnným
$sÚčel := $1

   ` Vytvoření array k uložení názvů polí
ARRAY STRING (31; asFields; 0)
```

```
   ` Vytvoření array k uložení ukazatelů k polím
ARRAY INTEGER (aiFields; 0)

   ` Nastavení array exportních polí
ARRAY STRING (31; asImportExportFields; 0)
ARRAY INTEGER (aiImportExport; 0)
GEN_FieldsToArray (pTable; ->asFields; ->aiFields; False;False)

   ` Výchozí hodnota pro formát dialogu.
if (<>WIN_fIsWindows)
   sFieldDelimiter := "44"
   sRecordDelimiter := "10"
   ckCommasQuote := 1
Else
   sFieldDelimiter := "9"
   sRecordDelimiter := "13"
   ckCommasQuote := 0
End if
ckInclude := 0
sDlogText1 := $sÚčel + " Dat…"
sDlogText2 := Table name (pTable) + " Pole"
sDlogText3 := $sÚčel + " Pořadí"

$LWidth := 390 ` cílová šířka
$LHeight := 245 `    cílová šířka
$LWindowID :=WIN_LNewWindow ($LWidth; $LHeight; <>WIN_LCentered;
<>WIN_LModal)
DIALOG ([zDialogs]; "IMPEXP_ImportExport")
CLOSE WINDOW

If (bExport = 1)

   Case of

      : (ckForm = 1)                        ` Užít existujíc formulář
         $LFormat := 3
      ARRAY INTEGER (aiImportExport; 0)

      : (ck4DFormat = 1)                    ` Použít vnitřní formát 4D
         $LFormat := 2
      ARRAY INTEGER (aiImportExport; 0)

      : (ckInclude = 1)        ` Vložit záhlaví záznamu
         $LFormat := 1

      Else
         $LFormat := 0
   End case

   If ($sÚčel = "Export")  `Výběr použít pouze pro export
```

```
    SET CHANNEL(12;"")`Vytvořit nový soubor
    COPY NAMED SELECTION (pTable->; "<>ExportSelection")
  Else
    SET CHANNEL(10;"")`Otevřít existující soubor pro import
  End if

  If (OK=1)
    SET CHANNEL(11)`    Pro nyní zavřít soubor

      ` Připravit BLOB pro přiřazení exportovaných polí do nového procesu
    $LTableNumber := Table(pTable)
    VARIABLE TO BLOB ($LTableNumber;$oParameters;*)
    VARIABLE TO BLOB(aiImportExport;$oParameters;*)
    VARIABLE TO BLOB($LFormat;$oParameters;*)
    VARIABLE TO BLOB(ckCRLF;$oParameters;*)
    VARIABLE TO BLOB(ckCommasQuote;$oParameters;*)
    VARIABLE TO BLOB(sFieldDelimiter;$oParameters;*)
    VARIABLE TO BLOB(sRecordDelimiter;$oParameters;*)
    VARIABLE TO BLOB(Document;$oParameters;*)

        ` Začít export v odděleném procesu

    $sProcessName :=$sÚčel+"ing "+Substring(Table name(pTable);1;4)
    $Lpid :=Process number($sProcessName)
    If ($Lpid>0)                        ` Umožnit více procesům pracovat ve stejnou chvíli
    $sProcessName :=Substring ($sProcessName;1;29)+String(PROCESS_LnextView
($sProcessName))
                        ` Limit na 29 znaků pro vložení čísla pokud je potřeba
    End if

    $Lpid :=New process("P_IMPEXP_"+$sÚčel+"Data";32*1024;$sProcessName;
$oFieldPointers;True)
    SET BLOB SIZE ($oParameters;0)

    If ($Lpid=0)

      If ($sÚčel="Export")          ` Výběr použít pouze pro export
        CLEAR NAMED SELECTION("<>ExportSelection")    ` vymazat jestliže jiné
procesy nebyli vytvořeny
      End if

      ALERT("Not Enough memory to do the "+$sÚčel)
    End if

  End if                ` OK

End if                  ` bExport

  ` Konec metody
```

6. Vytvořte novou metodu projektu P_IMPEXP_ExportData následovně nebo ji importujte pomocí textového editoru:

```
If (False)
    ` Metoda: P_IMPEXP_ExportData (BLOB; boolean)
    ` ACI Univerzita kurzy programování
    ` Generic ACI Shell Programming
    ` Vytvořeno: Jim Steinman
    ` Datum: 16/1/97

  <>fGeneric :=True
  <>f_Version6x20 :=True
  <>fJ_Steinman :=True

End if

    ` Vytvoření parametrů
C_BLOB ($1)                          ` BLOB parametrů exportu
C_BOOLEAN($2;$fBackground)           ` Proces na pozadí (volitelné; True = pozadí)

    ` vytvoření místních proměnných
C_POINTER($pTable)                       ` uakaztel k tabulce
C_LONGINT($LFieldType)                   ` Součet parametrů
C_LONGINT($LFormat)                      ` Umístění okne
C_BOOLEAN($fCRLF)                        ` Vložit řádek po return
C_BOOLEAN($fCommasQuotes)            `  Použít čárky
C_STRING(2;$sFieldDelimiter)         ` Oddělovač polí
C_STRING(2;$sRecordDelimiter)        ` Oddělovač záznamů
C_STRING(2;$sDelimiter)                  ` dočasný oddělovač
C_STRING(2;$sQU)                     ` Uvedení znaků pro tečky & uvozovky Export
C_LONGINT($LNumberOfFields)          ` Počet polí k exportu
C_LONGINT($LNumberOfRecords)         ` Počet záznamů k exportu
C_LONGINT($LPosition)                ` Pozice znaků
C_LONGINT($LWindowID)                ` ID okna pro zprávy
C_LONGINT($LTable)                   ` Číslo tabulky
C_LONGINT($LOffset)                  ` Offset pro rozluštění BLOB
C_LONGINT($i;$j)                         ` opakovcí počítadlo
C_TIME($hDocumentReference)          ` Okdaz k dokumentu
C_TEXT($tBuffer)                     ` beffer dat
C_TEXT($tField)                          ` možná data polí
ARRAY INTEGER($aiFieldNumbers;0)

    ` definovat výchozí hodnoty
$fBackground:=False

    ` přiřazení BLOB parametrů k proměnným
$LOffset := 0
BLOB TO VARIABLE ($1;$LTable;$LOffset)
BLOB TO VARIABLE($1;$aiFieldNumbers;$LOffset)
BLOB TO VARIABLE($1;$LFormat;$LOffset)
```

```
BLOB TO VARIABLE($1;$i;$LOffset)
$fCRLF := ($i=1)                          ` 1 značí ano
BLOB TO VARIABLE($1;$i;$LOffset)
$fCommasQuotes := ($i=1)                  ` 1 značí ano
BLOB TO VARIABLE($1;$sFieldDelimiter;$LOffset)
$sFieldDelimiter := Char(Num($sFieldDelimiter))     ` převedení čísel řetězce na char
BLOB TO VARIABLE($1;$sRecordDelimiter;$LOffset)
$sRecordDelimiter:=Char(Num($sRecordDelimiter)) ` převedení čísel řetězce na char
BLOB TO VARIABLE($1;Document;$LOffset)
SET BLOB SIZE($1;0)
If (Count parameters=2)
   $fBackground := $2
End if

$LNumberOfFields := Size of array($aiFieldNumbers)
ARRAY POINTER ($pFields;$LNumberOfFields)
For ($i;1;$LNumberOfFields)
   $pFields{$i} := Field($LTable;$aiFieldNumbers{$i})   ` ukazatele k polím
End for

If ($fCRLF)
   $sRecordDelimiter := $sRecordDelimiter  + Char(Line feed)
End if
$pTable :=Table($LTable)`Get a pointer to the table being exported.

If ($fBackground )                        ` ExportData běží ve vlastním procesu

   MENU BAR (1)  ` nastavit nabídku pro proces tak, že nabídka bude prázdná
      `              když bude zobrazené okno exportu.

   <>LBackground := <>LBackground + 1
   USE NAMED SELECTION ("<>ExportSelection")
   CLEAR NAMED SELECTION ("<>ExportSelection")
   SET CHANNEL (10; Document)

Else                                      ` ExportDat byl volán procesem který otevřel soubor

   SET CHANNEL (12; "")

End if

If (OK = 1)

   If ($fBackground )
      HIDE PROCESS (Current process)      ` Přesunout napozadí
   End if

   FIRST RECORD ($pTable->)
   $LNumberOfRecords := Records in selection ($pTable->)

   $sTableName := Table name ($pTable)
```

```
   $LWindowID :=WIN_LNewWindow (200; 30; <>WIN_LCentered;
<>WIN_LStandardNoResize; "Export")

   GOTO XY (6; 0)
   MESSAGE ("Export " + $sTableName + "…")

   Case of

      : (($LFormat = 0) | ($LFormat = 1))` Text or Merge format

         ` Build an array of the field types
      ARRAY LONGINT (aLFieldTypes; $LNumberOfFields)

      For ($i; 1; $LNumberOfFields)
         GET FIELD PROPERTIES ($pFields{$i}; $LType)
         aLFieldTypes{$i} := $LType
      End for

      If ($fCommasQuotes)              ` Děláme tečky a uvozovky
         $sQU :=<>sQU
      Else
         $sQU :=""
      End if

      If ($LFormat = 1)                ` Merge Format. Nejdšíve exportovat názvy polí
         $sDelimiter := $sFieldDelimiter
         $tBuffer := ""

         For ($i; 1; $LNumberOfFields)

            If ($i = $LNumberOfFields) ` exportujeme poslední název pole
               $sDelimiter := $sRecordDelimiter
            End if

            $tBuffer := $tBuffer +$sQU + Field name ($pFields{$i}) +$sQU+ $sDelimiter
         End for

         SEND PACKET ($tBuffer)
      End if

         ` nyní export záznamů
      $tBuffer := ""

      For ($i; 1; $LNumberOfRecords)
         $sDelimiter := $sFieldDelimiter

         For ($j; 1; $LNumberOfFields)

            If ($j = $LNumberOfFields) ` exportujeme poslední pole v záznamu
               $sDelimiter := $sRecordDelimiter
```

```
      End if

      Case of
         : ((aLFieldTypes{$j} # Is Alpha Field) &  (aLFieldTypes{$j} # Is Text) &
               (aLFieldTypes{$j} # Is Boolean)) ` překonvertovat na řetězec

            If ((aLFieldTypes{$j}=Is Real) | (aLFieldTypes{$j}=Is Integer) |
                  (aLFieldTypes{$j}=Is Longint))  ` jestliže to je číslo
               $tData :=(String(($pFields{$j})->)+$sDelimiter)
            Else
               $tData :=$sQU+(String(($pFields{$j})->)+$sQU+$sDelimiter)
            End if

         : (aLFieldTypes{$j} = Is Boolean)  `      je to logické

            If ($pFields{$j}->)
               $tData := $sQU+"True" +$sQU+ $sDelimiter
            Else
               $tData := $sQU+"False" +$sQU+ $sDelimiter
            End if

      Else

         $tData := $sQU+ (($pFields{$j}->) +$sQU+ $sDelimiter)
      End case

      If ((Length ($tBuffer) + Length ($tData)) > 16000)
         SEND PACKET ($tBuffer)
         $tBuffer := $tData
         $tData :=""
      Else
         $tBuffer := $tBuffer + $tData
      End if

   End for

   NEXT RECORD ($pTable->)

   If ($i % 25 = 0)                  ` Nehrnout dopředu věci pouze k zobrazení
      GOTO XY (6; 1)
      MESSAGE ("Zznam " + String ($i) + " z " + String ($LNumberOfRecords))
   End if

End for

If ($tBuffer # $tData)
   SEND PACKET ($tBuffer)
End if

: ($LFormat = 2)          ` 4D Formát
```

```
      For ($i; 1; $LNumberOfRecords)

        SEND RECORD ($pTable->)
        NEXT RECORD ($pTable->)

        If ($i % 25 = 0)                    ` Nehrnout dopředu věci pouze k zobrazení
          GOTO XY (6; 1)
          MESSAGE ("záznam " + String ($i) + " z " + String ($LNumberOfRecords))
        End if

      End for

    : ($LFormat = 3)                        ` Použít formulář ImportExport
        ` Účel: Umožňit exportovaným záznamům do platné tabulky použít formulář
        ` Poznámka: Vývojář musí mít vytvořený formulář ImportExport pro tuto práci
      FldDelimit := Num ($sFieldDelimiter)
      RecDelimit := Num ($sRecordDelimiter)
      OUTPUT FORM ($pTable->; "ImportExport")
      EXPORT TEXT ($pTable->; "")
      OUTPUT FORM ($pTable->; "Output")

  End case

  CLOSE WINDOW
  SET CHANNEL (11) ` Zavřít dokument

  If ($fBackground )                        ` Vymazat čítač procesů na pozadí
    <>LBackground := <>LBackground - 1
  End if

End if                          ` OK = 1

  ` Konec metody
```

7. Vytvořte novou metodu projektu P_IMPEXP_ImportData následovně nebo ji importujte pomocí textového editoru:

```
If (False)
    ` Metoda: P_IMPEXP_ImportData (BLOB; boolean)
    ` ACI Univerzita kurzy programování
    ` Generická procedura
    ` Vytvořeno: Jim Steinman
    ` Datum: 16/1/97

    ` Účel: Importuje ukazatele polí do array ukazatelů.
    ` Všechny záznamy v platném výběru jsou exportovány.

  <>fGeneric :=True
  <>f_Version6x20 :=True
  <>fJ_Steinman :=True
```

```
End if

    ` Vytvořit parametry
C_BLOB($1)                          ` BLOB exportovaných záznamů
C_BOOLEAN($2;$fBackground)          ` Procesy v pozadí (volitelné; True = pozadí)

    ` Vytvoření místních proměnných
C_POINTER($pTable)                      ` Ukazatel k tabulce
C_LONGINT($LFieldType)                  ` parametr čítače
C_LONGINT($LFormat)                     ` Umístění okna
C_BOOLEAN($fCRLF                        ` Vložit řádek po return
C_BOOLEAN($fCommasQuotes)           ` Použít formát teček a uvozovek
C_STRING(2;$sFieldDelimiter)        ` Oddělovač polí
C_STRING(2;$sRecordDelimiter)       ` Oddělovač záznamů
C_LONGINT($LNumberOfFields)         ` Počet polí k importu
C_LONGINT($LPosition)                   ` Pozice znaků
C_LONGINT($LTable)                  ` číslo tabulky
C_LONGINT($LOffset)                 ` Offset pro rozluštění BLOB
C_LONGINT($i;$j)                        ` opakovací čítač
C_TIME($hDocumentReference)         ` Odkaz  dokumentu
C_TEXT($tBuffer)                    ` buffer dat
C_TEXT($tField)                         ` možná data polí
ARRAY INTEGER($aiFieldNumbers;0)

    ` definovat výchozí hodnoty
$fBackground:=False

    ` přiřazení BLOB parametrů k proměnným
$LOffset := 0
BLOB TO VARIABLE($1;$LTable;$LOffset)
BLOB TO VARIABLE($1;$aiFieldNumbers;$LOffset)
BLOB TO VARIABLE($1;$LFormat;$LOffset)
BLOB TO VARIABLE($1;$i;$LOffset)
$fCRLF := ($i=1)                        ` 1 Značí Ano
BLOB TO VARIABLE($1;$i;$LOffset)
$fCommasQuotes := ($i=1)                ` 1 Značí Ano
BLOB TO VARIABLE($1;$sFieldDelimiter;$LOffset)
$sFieldDelimiter := Char(Num($sFieldDelimiter))     ` Převedení čísla řetězce na char
BLOB TO VARIABLE($1;$sRecordDelimiter;$LOffset)
$sRecordDelimiter:=Char(Num($sRecordDelimiter)) ` Převedení čísla řetězce na char
BLOB TO VARIABLE($1;Document;$LOffset)
SET BLOB SIZE($1;0)
If (Count parameters=2)
   $fBackground := $2
End if

$LNumberOfFields := Size of array($aiFieldNumbers)
ARRAY POINTER($pFields;$LNumberOfFields)
For ($i;1;$LNumberOfFields)
   $pFields{$i} := Field($LTable;$aiFieldNumbers{$i})   ` ukazatele k polím
```

```
End for

If ($fCRLF)
   $sRecordDelimiter := Char(Line feed)
End if
$pTable :=Table($LTable)                    `Ukazatel k tabulce pro import.

If ($fBackground )                          ` Import dat běží ve vlastním procesu

   MENU BAR (1)             ` Nastavit nabídku pro proces tak, že nabídky nebude vidět
                            ` když bude zobrazené okno importu.

   <>LBackground := <>LBackground + 1

End if

If ($LFormat # 2)                           ` Ne 4D vnitřní formát
   $hDocumentReference:= Open document (Document)
Else
   SET CHANNEL (10; Document)
End if

If (OK = 1)

   If ($fBackground )
      HIDE PROCESS (Current process)        ` Ukititi na pozadí
   End if

   $sTableName := Table name ($pTable)

   $LWindowID :=WIN_LNewWindow (200; 30; <>WIN_LCentered;
<>WIN_LStandardNoResize; "Importing")

   GOTO XY(6; 0)
   MESSAGE ("Import " + $sTableName + "…")

   Case of

      : (($LFormat=0) | ($LFormat=1))               ` Text nebo Merge formát

         ` Vytvořit array typůpolí
         ARRAY LONGINT (aLFieldTypes; $LNumberOfFields)

         For ($i; 1; $LNumberOfFields)
            GET FIELD PROPERTIES ($pFields{$i}; $LType)
            aLFieldTypes{$i} := $LType
         End for

         If (OK = 1)
```

```
      If ($LFormat = 1)                        ` Pokud je označeno políčko"Skip
Header Record"
         RECEIVE PACKET ($hDocumentReference; $tBuffer; $sRecordDelimiter)
                        `  Pouze import a ignorovat první záznam.
      End if

      CREATE EMPTY SET ($pTable->; "ImportSet")

      RECEIVE PACKET ($hDocumentReference; $tBuffer; $sRecordDelimiter)
      $i := 0
      While (OK = 1)
         $i := $i + 1
         CREATE RECORD ($pTable->)

         $LPosition := Length ($tBuffer)
         If ($fCRLF)
            $tBuffer•$LPosition• := $sFieldDelimiter      ` Jestliže CRLF nahradí CR s
$sFieldDelimiter
         Else
            $tBuffer := $tBuffer + $sFieldDelimiter
         End if

         For ($j; 1; $LNumberOfFields)
            $LPosition := Position ($sFieldDelimiter; $tBuffer)
            $tField := Substring ($tBuffer; 1; $LPosition - 1)
            $tBuffer := Substring ($tBuffer; $LPosition + 1)

            If (Substring ($tField; 1; 1) = <>sQU)
               $LPosition := Position ((<>sQU + $sFieldDelimiter); $tBuffer)

             If ($LPosition > 0)
               $tField := $tField + $sFieldDelimiter + Substring ($tBuffer; 1; $LPosition - 1)
                  $tField := Substring ($tField; 2; Length ($tField))          ` Get rid of the
quotation marks.
                  $tBuffer := Substring ($tBuffer; $LPosition + 1)
               End if
            End if

            Case of

               : (aLFieldTypes{$j} = Is Alpha Field) | (aLFieldTypes{$j} = Is Text) ` Alfa
nebo Text
                  $pFields{$j}-> := $tField

               : (aLFieldTypes{$j} = Is Date)                              ` Datum
                  $pFields{$j}-> := Date ($tField)

               : (aLFieldTypes{$j} = Is Time)                          ` Čas
                  $pFields{$j}-> := Time ($tField)

               : (aLFieldTypes{$j} = Is Boolean)                       ` Logické
```

```
                $pFields{$j}-> := ($tField="True") | ($tField="Yes")

            Else    ` Číslo
              $pFields{$j}-> := Num ($tField)

            End case

          End for

          SAVE RECORD ($pTable->)
          ADD TO SET ($pTable->; "ImportSet")

        If ($i % 25 = 0)
            GOTO XY (6; 1)
            MESSAGE ("Záznam " + String ($i))
          End if

          RECEIVE PACKET ($hDocumentReference; $tBuffer; $sRecordDelimiter)
        End while

        USE SET ("ImportSet")
        CLEAR SET ("ImportSet")

      End if

    : ($LFormat = 2)                          ` 4D Format
      CREATE EMPTY SET ($pTable->; "ImportSet")

      RECEIVE RECORD ($pTable->)
      $i := 0

      While (OK = 1)
        $i := $i + 1

        SAVE RECORD ($pTable->)
        ADD TO SET ($pTable->; "ImportSet")

        If ($i % 25 = 0)
          GOTO XY (6; 1)
          MESSAGE ("Record " + String ($i))
        End if

        RECEIVE RECORD ($pTable->)
      End while

      USE SET ("ImportSet")
      CLEAR SET ("ImportSet")

    : ($LFormat = 3)                          ` Použít ImportExport Formulář
        ` Note: Vývojář musí mít vytvořený formulář s názvem ImportExport pro tuto práci
      FldDelimit := Num ($sFieldDelimiter)
```

```
        RecDelimit := Num ($sRecordDelimiter)
        INPUT FORM ($pTable->; "ImportExport")
        IMPORT TEXT ($pTable->; "")
        INPUT FORM ($pTable->; "Input")

    End case

    FLUSH BUFFERS
    CLOSE WINDOW

    If ($fBackground)                          ` Vymazat čítač procesů na pozadí
      <>LBackground := <>LBackground - 1
    Else                                       ` Běží ve stejném okně procesu
      WIN_OutputWindowTitle
    End if

End if

If ($LFormat # 2)                              ` Ne vnitřní formát 4D
   CLOSE DOCUMENT ($hDocumentReference)
Else
   SET CHANNEL (11)
End if

   ` Konec metody
```

8. Nahraďte text metody M_IMPEXP_Export následujícím textem, nebo importujte pomocí textového editoru.

```
If (False)
    ` Metoda: M_IMPEXP_Export
    ` ACI Univerzita kurzy programování
    ` Generic ACI Shell Programming
    ` Vytvořeno: Jim Steinman
    ` Datum: 1/15/97

    ` Účel: Tato metoda připraví data pro export

  <>f_Version6x10 :=True
  <>f_Version6x20 :=True
  <>fGeneric :=True
  <>fJ_Steinman :=True

    ` Upraveno: 1/17/97 k ovládání vlastního exportního editoru
  <>fK_Wilbur :=True
End if

C_LONGINT($LNumberOfRecordsInSelection)

$LNumberOfRecordsInSelection:=Records in selection(pTable->)

If($LNumberOfRecordsInSelection>0)
  IMPEXP_ImportExportDialog ("Export")
Else
  ALERT("Nejsou záznamy k exportu")
End if
  ` Konec metody
```

9. Nahraďte text metody M_IMPEXP_Import následujícím textem, nebo importujte pomocí textového editoru.

```
If (False)
    ` Metoda: M_IMPEXP_Import
    ` ACI Univerzita kurzy programování
    ` Generic ACI Shell Programming
    ` Vytvořeno: Jim Steinman
    ` Datum: 1/15/97

    ` Účel: Tato metoda připraví data pro import

  <>f_Version6x10 :=True
  <>f_Version6x20 :=True
  <>fGeneric :=True
  <>fJ_Steinman :=True

    ` Modified: 1/17/97 To handle custom export editor
```

```
   <>fK_Wilbur :=True
End if
```

❖ IMPEXP_ImportExportDialog ("Import")

```
   ` Konec metody
```

10. Nahraďte text metody M_GEN_Quit následujícím textem, nebo importujte pomocí textového editoru.

```
If (False)
      ` Metoda: M_GEN_Quit
      ` ACI Univerzita kurzy programování
      ` Vytvořeno:  Jim Steinman
      ` Datum: 1/15/97

      ` Účel: This method quits the database

   <>fGeneric :=True
   <>f_Version6x10 :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

      ` Modified: 1/17/97
   <>fK_Wilbur :=True

End if

<>fQuit:=True              ` Nastaví značku vypnutí a nic npůjde nastartovat

If (<>LBackground > 0)

   If (<>LBackground = 1)
      CONFIRM ("Na pozadí běží proces. Vypnout až skončí?")
   Else
      CONFIRM ("Na pozadí běží procesy. Vypnout až skončí?")
   End if

   If (OK = 1)

      Repeat
         PROCESS _MyDelay (Current process; 240)
      Until (<>LBackground = 0)

      QUIT 4D
   Else
      <>fQuit:=False
   End if

Else
   QUIT 4D
End if

   ` Konec metody
```

11. Prověřte metodu objektu pro asImportExportFields.

```
If (False)
      `Metoda: asExportImportFields  [Dialogy];"IMPEXP_ImportExport"
      `ACI Univerzita kurzy programování
      `Vytvořeno: Kent Wilbur
      ` Datum: 16/1/97

      `Účel: Řídit změny v array exportovaných polí

   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fK_Wilbur :=True

End if

C_LONGINT($LFormEvent)
$LFormEvent := Form event

Case of
   : (asImportExportFields # 0) & ($LFormEvent = On Double Clicked)
      DELETE ELEMENT(asImportExportFields;asImportExportFields)
      DELETE ELEMENT(aiImportExport;asImportExportFields)

      Case of
         : (Size of array(asImportExportFields) = 0)
            asImportExportFields:=0

         : (asImportExportFields > Size of array(asImportExportFields))
            asImportExportFields := Size of array(asImportExportFields)
      End case

   : ($LFormEvent = On Drop)
      GEN_HandleArraysDragAndDrop (Self;->aiFields;->aiImportExport)
End case
      `Konec metody
```

12. Přověřte metodu objektu asFields.

```
If (False)
      `Metoda: asFields  [Dialogy];"IMPEXP_ImportExport"
      `ACI Univerzita kurzy programování
      `Vytvořeno: Kent Wilbur
      ` Datum: 1/16/97

      ` Účel: Přiřadit označené položky do array exportovaných polí
      ` Nebo umožnit uživateli řídit tento array
      ` Skrýt označené při přetažení ze špatné array
```

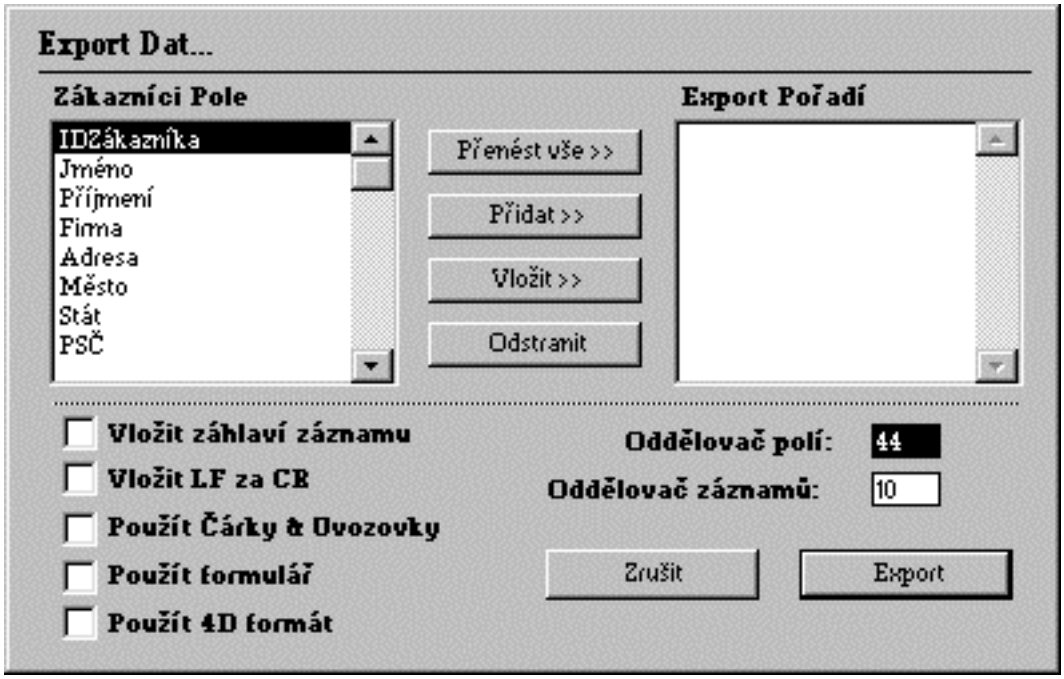
```
   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fK_Wilbur :=True

End if

C_LONGINT($LFormEvent)                ` Událost formuláře
C_LONGINT($LSourceElement)        ` Umístěné v array
C_LONGINT($LPid)                  ` Proces ID
C_POINTER($pSourceObject)         ` Ukazatel k zdrojovému objektu

$LFormEvent := Form event

Case of
   : ($LFormEvent = On Double Clicked)
      If ((asFields # 0) & (ckForm=0) & (ck4DFormat=0))
         $LSourceElement: = Size of array(asImportExportFields)+1

         INSERT ELEMENT(asImportExportFields;$LSourceElement;1)
         INSERT ELEMENT(aiImportExport;$LSourceElement;1)
         asImportExportFields{$LSourceElement} := asFields{asFields}
         aiImportExport{$LSourceElement} := aiFields{asFields}

         asImportExportFields := $LSourceElement

         If (asFields < Size of array(asFields))
            asFields := asFields+1
         Else
            asFields := 0                 ` Uživatel nevložil poslední pole dvakrát.
         End if
      End if

   : ($LFormEvent = On Drag Over)
      DRAG AND DROP PROPERTIES($pSourceObject;$LSourceElement;$LPid)
      If ($pSourceObject # Self)
         $0 := -1                         ` Tato operace není povolena, tak zrušit označení
      End if

   : ($LFormEvent = On Drop)
      GEN_HandleArraysDragAndDrop (Self;->aiFields;->aiFields)
End case
      `Konec metody
```

13. Otevřete formulář [zDialogy];"IMPEXP_ImportExport".
14. Otevřete vlastnosti objektu pro asImportExportFields.
15. Ujistěte se, že jsou na stránce Zobrazit zatržena políčka Táhnutelné a Vsaditelné.
16. Ujistěte se, že na stránce události jsou zatrženy události Při vsazení, Při odtažení a Při poklepnutí.
17. Otevřete vlastnosti objektu pro asFields.

18. Ujistěte se, že jsou na stránce Zobrazit zatržena políčka Táhnutelné a Vsaditelné.
19. Ujistěte se, že na stránce události jsou zatrženy události Při vsazení, Při odtažení a Při poklepnutí.
20. Proveďte metodu MyStartup a otestujte.

11.8. Jak udělat názvy polí a tabulek uživatelsky přívětivější

Někteří uživatelé by přivítali, kdyby v názvech polí a tabulek byly zobrazeny mezery mezi slovy. Pokud jsme dodrželi doporučení můžeme v názvech vyhledat velká písmena a vložit před ně mezery. Tímto způsobem budeme uživateli představovat přívětivější rozhraní a přitom stále používat dobré rozhraní pro vývojáře.



Jediný způsob jak to provést je procházet názvy písmeno po písmenu a určit, keré písmeno je velké. Můžeme tak provést otestováním ASCII hodnoty znaků. Pokud nepoužijeme české znaky stačí testovat zda ASCII není mezi 65 (velké A) a 90 (velké Z). Pokud použijete české znaky musíte testovat Uppercase znaku, pokud ASCII nezmění je písmeno velké, pokud se změní je písmeno malé. Samozřejmě předpokládá to striktní dodržování konvence názvů s velkými písmeny v počátcích slov.

11.8.1. Umístění mezer do názvů polí k zpřehlednění rozhraní.

1. Vytvořte novou metodu projektu GEN_sFriendlyName následujícím způsobem, nebo importujte pomocí textového editoru:

```
If (False)
      ` Metoda: GEN_sFriendlyName (Zdroj) -> Řetězec
      ` ACI Univerzita kurzy programování
      ` Generic ACI Shell Programming
      ` Vytvořeno: Jim Steinman
      ` Datum: 1/16/97

      ` Účel: Konvertuje malé/Velké bez mezery na  malé/ /Velké s mezerou
      ` Převede podtržítka na mezery

      ` Příklad: "SprávnýNázev" se převede na "Správný Název"

   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True
End if

      ` Declare parameters
C_STRING (255;$0)                     ` domončený řetězec
C_STRING (255;$1;$sSource)            ` řetězec k převedení

      ` Declare local variables
C_STRING (1; $sCharacter)          ` jednotlivý znak
C_LONGINT ($LCharacter)            ` čítač znaků
C_LONGINT ($LAscii )                  ` Ascii hodnota jestliže jednotlivý znak
C_LONGINT ($LPreviousAscii)        ` Ascii hodnota jestliže jednotlivý znak

      ` Přiřadí parametry proměnným
$sSource := $1
$0 := $sSource•1•
$LAscii := Ascii($0)
$LAsciiVel:=Ascii(Uppercase($0))

For ($LCharacter; 2; Length ($sSource))      ` Přeskočí první znak
   $sCharacter := $sSource•$LCharacter•
   $sCharVel:=Uppercase($sCharacter)
   $LPreviousAscii := $Lascii
   $LpreviousAsciiVel:= $LAsciiVel
   $LAscii := Ascii ($sCharacter)
   $LAsciiVel:= Ascii ($sCharVel)

   Case of
      :($LAscii =  95)                     ` Podtržítko
         $0 := $0 + <>sSP
      :($LAscii = $LAsciiVel)` Je velké
         If ($LPreviousAscii=$LpreviousAsciiVel)  `Předchozí znak je velké
```

```
            $0:=$0+$sCharacter
         Else
            $0 := $0 + <>sSP + $sCharacter
         End if
      Else
         $0 := $0 + $sCharacter
      End case
   End for
   ` Konec metody
```

2. Upravte metodu GEN_FieldsToArray následovně.

```
…
    If (($fIncludeInvisible) | (Not($fFieldInvisible)))` Vloží jestli není neviditelné nebo
      $LSizeOfArray:=$LSizeOfArray+1
      ARRAY STRING(31;$asFieldNames;$LSizeOfArray)
❖     $asFieldNames{$LSizeOfArray}:= GEN_sFriendlyName (Field name($pField))
      ARRAY POINTER($apFieldPointers;$LSizeOfArray)
      $apFieldPointers{$LSizeOfArray} := $pField
      ARRAY INTEGER($aiFieldNumbers;$LSizeOfArray)
      $aiFieldNumbers{$LSizeOfArray} := $i
    End if
…
```

3. Upravte metodu projektu P_IMPEXP_ExportData následovně.

```
…
    If ($LFormat = 1)                    ` Merge Format. Nejdříve exportovat názvy polí
      $sDelimiter := $sFieldDelimiter
      $tBuffer := ""

      For ($i; 1; $LNumberOfFields)

        If ($i = $LNumberOfFields) ` exportujeme poslední název pole
          $sDelimiter := $sRecordDelimiter
        End if

❖       $tBuffer := $tBuffer +$sQU + GEN_sFriendlyName (Field name ($pFields{$i}))
+
            $sQU+ $sDelimiter
      End for

      SEND PACKET ($tBuffer)
    End if

…
```

4. Vraťte se do prostředí Vlastní aplikace a testujte export .

12. Ovládání více oken procesů

Když je otevřeno mnoho oken nebo běží procesy na pozadí, zvláště je-li oevřeno několik oken z jedné tabulky, je běžným způsobem velice obtížné mít o těchto oknech přehled a ovládat je. Proto je potřeba vytvořit několik metod, které nám dají přehled o otevřených oknech a umožní je bude-li to nezbytné přenést na popředí. Můžeme kombinovat tuto úlohu s paletou tlačítek pro spouštění nových oken s procesy.

(možnost označená T níže)

Nebo protože můžeme rovněž použít paletu nástrojů samotné 4D, můžeme použít pro okna pouze malé plovoucí okno, obsahující pouze seznam otevřených oken. V tomto okamžiku nemáme žádnou metodu, kerou bychom takovéto funkce mohli zavést.

(možnost označená W níže)

Náš seznam oken potřebuje ovládat pouze procesy uživatele. Tyto procesy rovněž zahrnují procesy na pozadí, které si může uživatel chtít zkontrolovat. Přenese-li se však uschovaný proces na popředí zůstane stále uschován. Uschovaný proces je potřeba nejdříve zobrazit a pak teprve přenést na popředí. Také potřebujeme rozeznat, že daný proces je uschovaný proces. Můžeme to provést přiřazením záporných čísel procesů uschovaným procesům, pracujícím na pozadí.

12.1. Abs (Číslo) -> Číslo

Použití procesů se záporným ID procesu není možné. Takže před tím, než budeme s procesem zacházet musíme mít jistotu, že pracujeme s kladným číslem. Příkaz funkce

Abs(Číslo) převede toto číslo do kladných hodnot a zaručí nám, že budeme pracovat s kladným číslem. Funkce navrací kladnou hodnotu téže velikosti pro libovolné číslo předané do funkce Abs.

12.2. Frontmost process ({*}) -> Integer

Tato funkce navrátí ID procesu pro proces jehož okno je právě na popředí. Jestliže použijete volitelný (*) nebudou brány v úvahu procesy s plovoucími okny.

12.3. Více plovoucích palet potřebuje přepínací záhlaví

Nyní, když budeme mít dvě plovoucí paley musí uživatel vědět, kerá z nich je aktivní. Potřebujeme upravit existující patetu a přidat přepínací záhlaví.

12.4. Vytvoření palety tabulek

12.4.1. Vytvoření palety se seznamem, reprezentujícím dostupná okna

1. Vytvořte novou metodu projektu PROCESS_UpdateWindowArray následovně nebo importujte pomocí textového editoru.

```
If (False)
    ` Metoda: PROCESS_UpdateWindowArray (string; longint; boolean)
    ` ACI Univerzita kurzy programování
    ` Generic ACI Shell Programming
    ` Vytvořeno: Jim Steinman
    ` Datum: 1/16/97

    ` Účel: Adds and deletes window from the tables palette

  <>fGeneric :=True
  <>f_Version6x20 :=True
  <>fJ_Steinman :=True

End if

    ` Vytvoření parametrů
C_STRING (31; $1;$sProcessName)     ` Název procesu nebo okna k zobrazení
C_LONGINT ($2;$LCurrentProcess)     ` Proces ID (Volitelné)
C_BOOLEAN ($3;$fBackground)         ` Proces na pozadí (Volitelné)

    ` Vytvoření místních proměnných
C_LONGINT ($LLocation)                  ` Umístění v array řetězců
C_LONGINT ($LSizeOfArray)               ` Velikost array
C_LONGINT ($LState)                 ` Stav procesu
C_LONGINT ($LTime)                  ` Čas procesu

    ` Definice výchozích hodnot
If (Count parameters=3)
```

```
   $fBackground:= $3
Else
   $fBackground:=False
End if

If (Count parameters > 1)
   $LCurrentProcess := $2
   PROCESS PROPERTIES ($LCurrentProcess; $sProcessName; $LState; $LTime)

   If ($fBackground)                          ` Proces v pozadí má záporné číslo procesu
      $LCurrentProcess := $2 * -1
   End if

Else
   $sProcessName := $1
   $LCurrentProcess := Current process
End if

$LLocation := Find in array (<>PROCESS_asWindowName; $sProcessName)

While (Semaphore ("$ModifyWindowArrays"))
   PROCESS _MyDelay ($LCurrentProcess; 1)
End while

If ($LLocation = -1)
   $LSizeOfArray := Size of array (<>PROCESS_asWindowName)
   ARRAY STRING (35; <>PROCESS_asWindowName; $LSizeOfArray + 1)
   ARRAY INTEGER (<>PROCESS_aiWindows; $LSizeOfArray + 1)
   <>PROCESS_asWindowName{$LSizeOfArray + 1} := $sProcessName
   <>PROCESS_aiWindows{$LSizeOfArray + 1} := $LCurrentProcess
Else
   DELETE ELEMENT (<>PROCESS_asWindowName; $LLocation)
   DELETE ELEMENT (<>PROCESS_aiWindows; $LLocation)
End if

CLEAR SEMAPHORE ("$ModifyWindowArrays")
CALL PROCESS (-1)      ` Překreslí meziprocesní proměnné právě používané

   ` Konec metody
```

2.T. Do metody MyStartup přidejte následující kód.

```
❖          ` Následující je pro zobrazení tlačítek tabulky a ovladačů okna
•          ARRAY STRING (35; <>PROCESS_asWindowName; 0)
•          ARRAY INTEGER (<>PROCESS_aiWindows; 0)
•          <>PROCESS_LTablePalettePid:= PROCESS_LSpawnProcess
("P_PROCESS_TablesPalette"; 32;"$TablesPalette"; True; False)
•          PROCESS_UpdateWindowArray ("Úvodní obrazovka")
•          <>PROCESS_asWindowName:= 1
```

2.W. Do metody MyStartup přidejte následující kód .

❖ ` následující je pro zobrazení palety oken
- ARRAY STRING (35; <>PROCESS_asWindowName; 0)
- ARRAY INTEGER (<>PROCESS_aiWindows; 0)
- <>PROCESS_LTablePalettePid:= PROCESS_LspawnProcess ("P_PROCESS_WindowsPalette"; 32;"$TablesPalette"; True; False)
- PROCESS_UpdateWindowArray ("Úvodní obrazovka")
- <>PROCESS_asWindowName:= 1

3.T. Vytvořte novou metodu projektu P_PROCESS_TablesPalette následovně nebo importujte pomocí textového editoru.

```
If (False)
      ` Metoda: P_PROCESS_TablesPalette
      ` ACI Univerzita kurzy programování
      ` Vytvořeno: Jim Steinman
      ` Datum: 1/16/97

      ` Účel: Zobrazí paletu tabulek

   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

End if

C_LONGINT($LWindowID)

MENU BAR (1)

$LWindowID := WIN_LNewWindow (290; 60; <>WIN_LUpperRight;
<>WIN_LFloatingPalette + <>WIN_LFloatingTitleCoefficient; "Tabulky";
"PROCESS_HidePalette")
DIALOG ([zDialogy]; "PROCESS_TablesPalette")
<>PROCESS_LTablePalettePid:= 0

   ` Konec metody
```

3.W. Vytvořte novou metodu projektu P_PROCESS_WindowsPalette následovně nebo importujte pomocí textového editoru:

```
If (False)
      ` Metoda: P_PROCESS_WindowsPalette
      ` ACI Univerzita kurzy programování
      ` Vytvořeno: Jim Steinman
      ` Datum: 1/16/97

      ` Účel: Zobrazí paletu oken

   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

End if

C_LONGINT($LWindowID)

MENU BAR (1)
```

```
$LWindowID := WIN_LNewWindow (140; 30; <>WIN_LUpperRight;
<>WIN_LFloatingPalette + <>WIN_LFloatingTitleCoefficient; "Okna";
"PROCESS_HidePalette")
DIALOG ([zDialogy]; "PROCESS_WindowsPalette")
<>PROCESS_LTablePalettePid:= 0
   ` Konec metody
```

4.T. Vytvořte novou metodu projektu M_PROCESS_TablesPalette následovně nebo importujte pomocí textového editoru.

```
If (False)
      ` Metoda: M_PROCESS_TablesPalette
      ` ACI Univerzita kurzy programování
      ` Vytvořeno: Jim Steinman
      ` Datum: 1/16/97

      ` Účel: Tato metoda vytvoří nebo přenese na popředí proces Tables Palette

   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

End if

<>PROCESS_LTablePalettePid:= PROCESS_LSpawnProcess
("P_PROCESS_TablesPalette"; 32; "$TablesPalette"; True; False)

   ` Konec metody
```

4.W. Vytvořte novou metodu projektu M_PROCESS_WindowsPalette následovně nebo importujte pomocí textového editoru.

```
If (False)
      ` Metoda: M_PROCESS_WindowsPalette
      ` ACI Univerzita kurzy programování
      ` Vytvořeno: Jim Steinman
      ` Datum: 1/16/97

      ` Účel: Tato metoda vytvoří nebo přenese na popředí proces Windows Palette

   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

End if

<>PROCESS_LTablePalettePid:= PROCESS_LSpawnProcess
"P_PROCESS_WindowsPalette"; 32; "$TablesPalette"; True; False)
```

```
` Konec metody
```

5.T. Otevřete záhlaví nabídek #1 a přidejte k nabídce Soubor následující položky:

Menu Item	Method
Produkty	M_Products
-	
Pateta tabulek	M_PROCESS_TablesPalette
-	
Konec	M_GEN_Quit

5.W. Otevřete záhlaví nabídek #1 a přidejte k nabídce Soubor následující položky:

Menu Item	Method
Produkty	M_Products
-	
Paleta oken	M_PROCESS_WindowsPalette
-	
Konec	M_GEN_Quit

6. Vytvořte novou metodu projektu PROCESS_HidePalette následovně nebo importujte pomocí textového editoru.

```
If (False)
    ` Metoda: PROCESS_HidePalette
    ` ACI Univerzita kurzy programování
    ` Vytvořeno: Jim Steinman
    ` Datum: 1/16/97

    ` Účel: Skryje proces ovládací palty

  <>fGeneric :=True
  <>f_Version6x20 :=True
  <>fJ_Steinman :=True

End if

HIDE PROCESS(Current process)
PAUSE PROCESS(Current process)

  ` Konec metody
```

7. Nahraďte text v metodě GEN_ModifySelection následovně nebo importujte pomocí textového editoru:

```
If (False)
    ` Metoda: GEN_ModifySelection
    ` ACI Univerzita kurzy programování
    ` Generic ACI Shell Programming
    ` Vytvořeno: Jim Steinman
    ` Datum: 1/15/97

    ` Účel: Zobrazí výstupní formulář pro tabulku určenou ukazatelem pTable.

  <>fGeneric :=True
  <>f_Version6x10 :=True
  <>f_Version6x20 :=True
  <>fJ_Steinman :=True

    ` Upraveno: 1/17/97
```

```
   <>fK_Wilbur :=True

End if

C_LONGINT ($LWindowID)

MENU BAR (3)

INPUT FORM (pTable->; "Input";*)                ` Nastaví na navrženou velikost
OUTPUT FORM (pTable->; "Output")

$LWindowID :=WIN_LNewWindow (-1;  -1; <>WIN_DesignerSizeCascade;
<>WIN_LStandardNoResize)

ALL RECORDS (pTable->)
CREATE EMPTY SET(pTable->; "UserSet")

PROCESS_UpdateWindowArray (""; Current process)

WIN_OutputWindowTitle
MODIFY SELECTION (pTable->; *)

PROCESS_UpdateWindowArray (""; Current process)

CLOSE WINDOW
   ` Konec metody
```

8. Upravte metodu P_IMPEXP_ExportData následovně.

```
…
   $LWindowID :=WIN_LNewWindow (200; 30; <>WIN_LCentered;
<>WIN_LStandardNoResize; "Exporting")
❖  If ($fBackground )
❖     PROCESS_UpdateWindowArray (""; Current process;True)
❖  End if

.... Dále v kódu ....

   If ($fBackground )
❖     PROCESS_UpdateWindowArray (""; Current process;True)<>LBackground :=
<>LBackground -1
   End if
…
```

9. Upravte metodu projektu P_IMPEXP_ImportData následovně:

```
…
   $LWindowID :=WIN_LNewWindow (200; 30; <>WIN_LCentered;
<>WIN_LStandardNoResize; "Importing")
❖  If ($fBackground )
❖     PROCESS_UpdateWindowArray (""; Current process;True)
```

❖
```
End if

.... Dále v kódu....

CLOSE WINDOW
If ($fBackground )
❖   PROCESS_UpdateWindowArray (""; Current process;True)
   <>LBackground := <>LBackground -1
Else                                    ` Running in some process update the window
   WIN_OutputWindowTitle
End if
…
```

10. Restartujte databázi pro načtení nového Záhlaví #1 a testujte patety.

Náš seznam není synchronizován s oknem na popředí. V současnosti se seznam při otevření vrací do výchozí pozice Výchozí obrazovka. Mohli bychom nechat sezanam v okně se vracet do výchozí pozice při otevření, ale pokud uživatel bude přecházet z jednoho okna do druhého klepnutím na okno bude pozice v seznamu ponekud matoucí. Abychom tento problém vyřešili, použijeme meziprocesní komunikaci, abychom sdělili patetě, že bylo na popředí přeneseno jiné okno.

12.4.2. Synchronizování Palety tabulek s oknem na popředí

1. Vytvořte novou metodu projektu PROCESS_UpdateWindowsPalette následovně nebo importujte pomocí textového editoru.

```
If (False)
    ` Metoda: PROCESS_UpdateWindowsPalette
    ` ACI Univerzita kurzy programování
    ` Generic ACI Shell Programming
    ` Vytvořeno: Jim Steinman
    ` Datum: 1/16/97

    ` Účel: toto zajistí synchronizaci palety oken s oknem na popředí

  <>fGeneric :=True
  <>f_Version6x20 :=True
  <>fJ_Steinman :=True

End if

C_LONGINT ($LCurrentFrontProcess) ` platný proces na popředí
C_LONGINT ($LFormerFrontProcess) ` Starý proces popředí
C_LONGINT ($LLocation)  ` Umístění procesu v array čísel procesu

$LFormerFrontProcess := 0
Repeat
  $LCurrentFrontProcess := Frontmost process(*)

  If ($LCurrentFrontProcess # $LFormerFrontProcess)
    $LLocation := Find in array(<>PROCESS_aiWindows;$LCurrentFrontProcess)

    If ($LLocation>0)                    ` Tento proces je v paletě
      <>PROCESS_asWindowName := $LLocation
      $LFormerFrontProcess := $LCurrentFrontProcess
      CALL PROCESS(<>PROCESS_LTablePalettePid)      ` Obnovit rozevírací nabídku
    End if
  End if

  PROCESS_MyDelay (Current process;60)
Until (Process state(<>PROCESS_LTablePalettePid)= -1)
```

```
` Konec metody
```

2.T. Otevřete formulář [zDialogy]; "PROCESS_TablesPalette".

2.W Otevřete formulář [zDialogy]; "PROCESS_WindowsPalette".

3. Upravte tlačítko <>PROCESS_asWindowName následovně:

```
If (False)
    ` Object Metoda: <>PROCESS_asWindowName
    ` [Dialogy]; "PROCESS_TablesPalette"
    ` ACI Univerzita kurzy programování
    ` Vytvořeno: Jim Steinman
    ` Datum: 1/16/97

    ` Účel: Pokud je potřeba, přenést na popředí okno nebo skrytý proces

  <>fGeneric :=True
  <>f_Version6x20 :=True
  <>fJ_Steinman :=True

End if

Case of

  : (<>PROCESS_aiWindows{<>PROCESS_asWindowName} < 0)    ` Toto je proces na
pozadí

    If (Frontmost process (*) = Abs
(<>PROCESS_aiWindows{<>PROCESS_asWindowName}))
        ` Je přední okno proces pozadí (-num)
      HIDE PROCESS (-<>PROCESS_aiWindows{<>PROCESS_asWindowName})
    Else
      SHOW PROCESS (-<>PROCESS_aiWindows{<>PROCESS_asWindowName})
      BRING TO FRONT (-<>PROCESS_aiWindows{<>PROCESS_asWindowName})
    End if

  : (<>PROCESS_asWindowName> 0)
    SHOW PROCESS  (<>PROCESS_aiWindows{<>PROCESS_asWindowName})
    BRING TO FRONT (<>PROCESS_aiWindows{<>PROCESS_asWindowName})

End case
```

- ~~<>PROCESS_asWindowName:=1~~

```
  ` Konec metody
```

4. V metodě MyStartup upravte následovně:

```
  ` The following is for displaying the table buttons and window controls
ARRAY STRING (35; <>PROCESS_asWindowName; 0)
ARRAY INTEGER (<>PROCESS_aiWindows; 0)
```

- <>LWindowPaletteUpdater := 0

```
<>PROCESS_LTablePalettePid:= PROCESS_LSpawnProcess
("P_PROCESS_TablesPalette"; 32; "$TablesPalette"; True; False)
PROCESS_UpdateWindowArray ("Úvodní obrazovka")
<>PROCESS_asWindowName := 1
```

5.T. Upravte kód metody P_PROCESS_TablesPalette následovně:

```
If (False)
    ` Metoda: P_PROCESS_TablesPalette
    ` ACI Univerzita kurzy programování
    ` Vytvořeno: Jim Steinman
    ` Datum: 1/16/97

    ` Účel: Zobrazí paletu tabulek

  <>fGeneric :=True
  <>f_Version6x20 :=True
  <>fJ_Steinman :=True

End if

C_LONGINT($LWindowID)

MENU BAR (1)

• <>LWindowPaletteUpdater := PROCESS_LSpawnProcess
("PROCESS_UpdateWindowsPalette";16;"$UpdatePalette";True;False)

$LWindowID := WIN_LNewWindow (290; 60; <>WIN_LUpperRight;
<>WIN_LFloatingPalette + <>WIN_LFloatingTitleCoefficient; "Tabulky";
"PROCESS_HidePalette")
DIALOG ([zDialogy]; "PROCESS_TablesPalette")
<>PROCESS_LTablePalettePid:= 0
• <>LWindowPaletteUpdater :=  0
  ` Konec metody
```

5.W. Upravte kód metody P_PROCESS_WindowsPalette následovně:

```
If (False)
    ` Metoda: P_PROCESS_WindowsPalette
    ` ACI Univerzita kurzy programování
    ` Vytvořeno: Jim Steinman
    ` Datum: 1/16/97

    ` Účel: Zobrazí paletu oken

  <>fGeneric :=True
  <>f_Version6x20 :=True
  <>fJ_Steinman :=True
```

```
        End if

        C_LONGINT($LWindowID)

        MENU BAR (1)

•       <>LWindowPaletteUpdater := PROCESS_LSpawnProcess
("PROCESS_UpdateWindowsPalette";16;"$UpdatePalette";True;False)

        $LWindowID := WIN_LNewWindow (140; 30; <>WIN_LUpperRight;
        <>WIN_LFloatingPalette + <>WIN_LFloatingTitleCoefficient; "Okna";
        "PROCESS_HidePalette")
        DIALOG ([zDialogy]; "PROCESS_WindowsPalette")
        <>PROCESS_LTablePalettePid:= 0
•       <>LWindowPaletteUpdater :=  0

          ` Konec metody
```

6. Upravte kód na konci metody PROCESS_UpdateWindowArray následovně:

```
        …
        Else
          DELETE ELEMENT(<>PROCESS_asWindowName;$LLocation)
          DELETE ELEMENT(<>PROCESS_aiWindows;$LLocation)
        End if

        CLEAR SEMAPHORE ("$ModifyWindowArrays")
❖       CALL PROCESS(-1)
•       If (<>LWindowPaletteUpdater>0)
❖         DELAY PROCESS(<>LWindowPaletteUpdater;0)
❖       End if

          ` Konec metody
```

7. Proveďte metodu MyStartup a testujte patetu.

12.5. Uschování procesu a okna

Extra kredit Diskuze

Uživatel může občas otevřít příliš mnoho oken a tato okna jej matou. Můžeme do formulářů přidat tlačítko, keré dovolí uživateli dočasně si uschovat okno a pak použít nabídky oken v patetě k jeho opětovnému ukázání.

12.5.1. Provedení změny v tlačítkách výstupního formuláře

1. Nahraďte tlačítka ve výstupních formulářích za tlačítka z formuláře:

   [zDialogy]; "OutputButtons.rev"

14. Vše co potřebujete znát o zamykání záznamů

Pravidla pro zamykání záznamů z hlediska více uživatelů diskutované v této kapitole jsou rovněž platná pro více procesů jak v prostředí jednoho uživatele, tak i více uživatelů.

Na začátek, příprava databáze pro více procesů nebo víceuživatelů může být zrovna tak jednoduchá jako spuštění procesu nebo nastavení databáze pro server. Toto tvrzení však platí pouze při splnění následujících tří podmínek. Za prvé, pokud nebudete používat vložené formuláře, či budete používate v podformulářích pouze záznamy přímo svázené s rodičovským záznamem (podtabulky). Např. podtabulku pro více telefonních čísel jednoho zákazníka. Za druhé, nebudete v kódu adresovat záznamy z vztažených tabulek , totéž i ve formulářích. Za třetí, že budete k ovládání záznamů používat pouze příkazy, keré automaticky ošetřují zamykání záznamů a hlášení o uzamčení záznamu.Bohužel však velmi málo databází splňuje tyto tři podmínky, včetně naší databáze ACI Video. Proto v této kapitole vyvineme systém ošetřující kompletně uzamykání záznamů, který může vyhovovat pro většinu databází 4D. Tento systém je většinou generický a může být použit v libovolné databázi, kterou vytvoříte, tak že do ní tyto metody přenesete a použijete patřičná generická volání v potřebných místech.

14.1. Automatická upozornění na zamknuté objekty

Určité příkazy ošetřují hlídání uzamčenosti záznamů (či objektů obecně) a upozornění na ně automaticky. Tato automatická upozornění poskytují:

- MODIFY SELECTION
- Prostředí uživatele
- Prostředí návrháře

Když používáte tyto příkazy, není nutno abyste ošetřovali zamykání záznamů a patřičná hlášení, protože tak učiní 4D za vás.

Všimněte si prosím, že MODIFY SELECTION uzamyká a upozorňuje pouze na záznamy z hlavní tabulky, pro kerou byl příkaz použit. Nevztahuje se to na záznamy ze vztažených tabulek, použitých v podformuláři.

14.2. Definice týkající se záznamů

Předtím než budeme pokračovat musíte rozumět několika jednoduchým deefinicím ohledně záznamů, toho co se děje s pamětí a co může být potenciálně uzamčeno.

14.2.1. Platný záznam

Každá tabulka v každém procesu má jeden platný záznam určený ukazatelem na záznam. Ukazatel na záznam může :

- ukazovat na již existující záznam
- ukazovat na nový záznam (při přidávání záznamů)
- neukazovat na žádný záznam (jako když je platný výběr prázdný)

14.2.2. Zavedený záznam

Záznam je zaveden, když je zaveden do paměti. Zavedený záznam je vždy platný záznam. Platný záznam může být zavedený nebo nezavedený (vyvedený z) do paměti. Vyvedení záznamu z paměti nemění ukazatel na záznam.

14.2.3. Uzamčený záznam

Jestliže je záznam zaveden v módu číst/psát, je záznam uzamčen pro další uživatele a procesy. Uzamčený záznam je záznam, který lze v jiných procesech pouze číst, ale nelze do něj zapsat. Zázanm je uzamčen, když jiný uživatel nebo proces zavedl záznam v módu číst/psát. Pouze uživatel, který záznam pro jiné uzamkl může záznam vidět jako neuzamčený. Tento záznam je pro ostatní uživatele a procesy uzamčen. Záznam může být pro vás rovněž uzamčen v daném procesu, jestliže jste v módu tabulky pouze číst.

Jestliže jste ve více uživatelích, můžete a budete chtít vyvést záznam a odemknout jej tak pro ostatní, okamžitě, když skončíte práci se záznamem. Jestliže pracujete v databázi pouze s jedním procesem a uživatelem nemáte důvod toto provádět.

14.3. Příkazy s vlivem na stav (mód) tabulky

14.3.1. Příkaz READ ONLY

Tento příkaz přepne tabulku v procesu do stavu, kdy lze pouze číst její záznamy. Záznamy tabulky lze pak v daném procesu pouze zavést, ale nelze provádět změny v záznamech, které by bylo možno uložit na trvalo do tabulky. Záznam při zavedení není uzamčen, je již apriori uzamčen předtím než je zaveden. Funkce Locked vrátí vždy hodnotu True pro tabulku, kerá je ve stavu pouze číst.

Příkaz READ ONLY má vliv pouze na stav tabulky v daném procesu, kde byl použit. Když použijete příkaz READ ONLY nemá to vliv na stav tabulky v jíných procesech a na jiných pracovních stanicích.

Tento příkaz je obvykle používán k zabránění nechtěného uzamčení záznamu. Změní stav pro další zaváděný záznam. Jestliže je záznam již zaveden a uzamčen, zůstane

uzamčen i po použití příkazu READ ONLY. Pokud chcete záznam vyvést a odemknou musíte použít příkaz UNLOAD RECORD.

14.3.2. Příkaz READ WRITE

Tento příkaz přepne tabulky do stavu, který umožní v daném procesu číst a měnit záznamy. Záznamy pro tuto tabulku mohou být zaváděny a uzamykány pro další uživatele a procesy. Jestliže byl zaveden uzamčený záznam (byl zaváděn ve stavu pouze číst, nebo jej používá jiný uživatel či proces) použití tohoto příkazu jej neodemkne. Záznam lze uzamknout pro jiné pouze je-li tabulka ve stavu číst/psát. Záznam bude možno upravovat pouze tehdy je-li zaveden neuzamčený a tabulka je ve stavu číst/psát.

Když poprvé otevřete databázi, nebo proces jsou všechny tabulky ve stavu číst/psát a tento příkaz nemusí být použit. Tento příkaz READ WRITE je používán byla-li tabulka v předchzím běhu kódu převedena do stavu pouze číst příkazem READ ONLY.

Když je tabulka ve stavu číst/psát a do paměti je zaváděn záznam neuzamčený jiným uživatelem je záznam automaticky uzamčen pro jiné uživatele.

Tento příkaz mění stav pro záznamy, které budou zavedeny v dalším běhu kódu. Pokud byl již záznam zaveden jako uzamčený, zůstane ve stavu pouze pro čtení. Abychom mohli tento záznam zavést k úpravám je jej nutno nejdříve vyvést a pak opět zavést (případně počkat až jej jiný uživatel či proces uvolní). To znamená, že nelze jednoduše ke změně stavu záznamu přidat tlačítko s kódem READ WRITE do vstupního formuláře.

14.3.3. Příkazy, které se snaží zavést a uzamknout záznam

Všechny tyto příkazy mění ukazatel na záznam. Každý příkaz, kerý mění uklazatel na platný záznam se pokusí po změně ukazatele zavést platný záznam a uzamknout jej a provedou to pokud není již uzamčen jiným uživatelem/peocesem nebo v závislosti na stavu tabulky pouze číst. Protože tyto příkazy uzamykají za normální situace záznamy je to často dobrý důvod proč před jejich používáním převést tabulku do stavu pouze číst příkazem READ ONLY a převést ji do stavu READ WRITE až když chceme záznamy upravovat ato buď programem, nebo akcí uživatele přes formulář.

- ALL RECORDS
- APPLY TO SELECTION
- APPLY TO SUBSELECTION
- ARRAY TO SELECTION
- CREATE RECORD
- CREATE RELATED ONE
- DELETE SELECTION
- DISTINCT VALUES

- FIRST RECORD
- GOTO RECORD
- GOTO SELECTED RECORD
- LAST RECORD
- LOAD RECORD
- NEXT RECORD
- OLD RELATED MANY
- OLD RELATED ONE
- ONE RECORD SELECT
- ORDER BY
- ORDER BY FORMULA
- PREVIOUS RECORD
- POP RECORD
- PUSH RECORD
- QUERY
- QUERY BY EXAMPLE
- QUERY BY FORMULA
- QUERY SELECTION
- QUERY SELECTION BY FORMULA
- RELATE MANY
- RELATE MANY SELECTION
- RELATE ONE
- RELATE ONE SELECTION
- REDUCE SELECTION
- SCAN INDEX
- SEARCH BY INDEX
- SORT BY INDEX
- USE NAMED SELECTION
- USE SET

Příkaz ALL RECORDS je dobrý příklad na to přoč je dobré držet tabulku ve stavu pouze číst a přepnout ji do stavu číst/psát pouze pokud víte, že se budou upravovat záznamy. Při tomto příkaze se první záznam tabulky stává platným záznamem a je zaveden do paměti a uzamčen. Snaha by měla být vždy se vyhnout zbytnému zamykání záznamů.

14.3.4. Příkazy, které dočasně mění stav tabulky na pouze číst

Následující příkazy nastavují stav tabulky na stav pouze číst a po svém skončení převedou tabulku do původního stavu. 4D automaticky nastavuje stav tabulky na pouze číst pro ty příkazy, které rozhodně pro svůj účel nepotřebují tabulku ve stavu číst/psát a nepotřebují měnit záznamy. Před provedením těchto příkazů 4D uloží původí stav

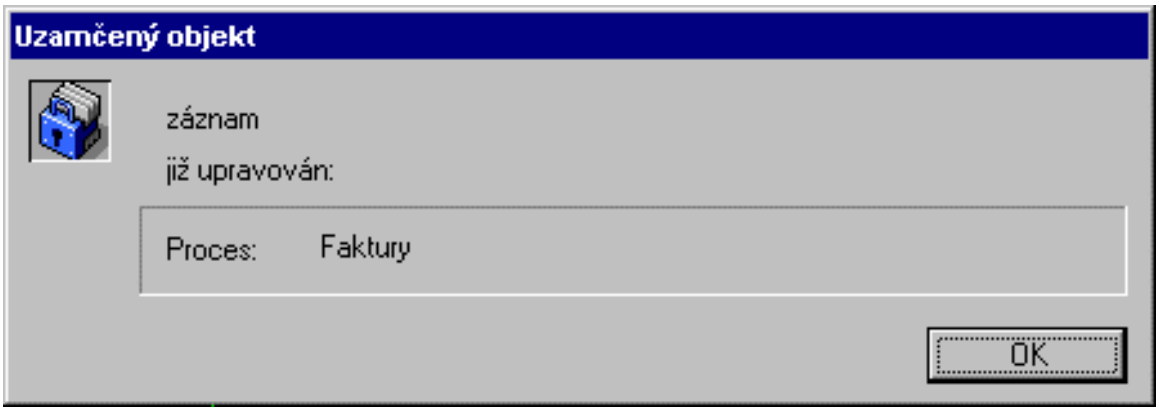
tabulek (číst/psát nebo pouze číst). Po provedení příkazů je stav tabulky navrácen k tomuto původnímu.

- DISPLAY SELECTION
- EXPORT DIF
- EXPORT SYLK
- EXPORT TEXT
- GRAPH TABLE
- PRINT LABEL
- PRINT SELECTION
- REPORT
- SELECTION TO ARRAY
- MERGE SELECTION
- MODIFY SELECTION

Situace u MODIFY SELECTION je složitější a potřebuje více vysvětlení. Tento příkaz nastavuje stav tabulky na pouze číst ve výstupním formuláři. Pokud uživatel poklepe na záznam k přechodu do vstupního formuláře bude tabulka ve stavu číst/psát (pokud tak měla být před použitím tohoto příkazu). Když uživatel přechází zpět do výstupního formuláře vrátí se tabulka do stavu pouze číst.

14.4. Zabránění nechtěnému zamykání záznamů a inicializace nového procesu

Jedna z nejběžnějších úloh ve 4D je zabránění nechtěnému uzamykání záznamů. Například, když vkládáte novou fakturu a přidáváte položku. Když napíšete Idproduktu, databáze vyhledá patřičný záznam v tabulce Produkty, aby použila jednotkové ceny a uzamkne přitom záznam. Tento záznam je nyní uzamčen pro všechny další uživatele. Při úloze, kdy pouze přebíráme jednotkovou cenu a nepotřebujeme nic zapisovat do záznamu je toto zamykání nadbytečné, přesto jste nechtěně tento záznam uzamkli. Nejlepší způsob jak tomu zabránit je převést tabulku Produkty do stavu pouze číst. Obecně je nejlepší při spuštění nového procesu a při spuštění celé databáze převést všechny tabulky do stavu pouze číst příkazem READ ONLY(*), to jedt pro všechny tabulky. Pak nastavit na číst/psát příkazem READ WRITE pouze ty tabulky do nichž budeme potřebovat zapisovat v tomto procesu.



Nyní jsme diskutovali několik věcí, které niní musíme převést do instrukcí programu pro každý proces. Je běžné vytvořit metodu, která se zmíněné požadavky ošetří. Nastavení všech tabulek READ ONLY, stanovení výchozí tabulky DEFAULT TABLE pro proces a stanovení všech potřebných výchozích hodnot proměnných. V této databázi zatím potřebujeme nastavit pouze jednu proměnnou, a to pTable. Jestliže předáme ukazatel na tabulku s kterou se chystáme pracovat zvýšíme účinnost našeho programování (zde na třech místech) a použijeme tentýž generický kód pro všechny procesy tabulek.

14.4.1. Nastavení všech tabulek READ ONLY, pouze číst

1. Přidejte následující kód do metody projektu MyStartup.

```
READ ONLY (*) `         Nastaví na pouze číst všechny tabulky
```

2. Vytvořte novou metodu projektu INITIALIZE_Process následovně:

```
If (False)
    ` Metoda: INITIALIZE_Process
    ` ACI Univerzita kurzy programování
    ` Generic ACI Shell Programming
    ` Vytvořeno: Jim Steinman
    ` Datum: 1/16/97
    ` Účel: Nastaví tabulky na Pouze číst a Číst Psát jak je potřeba
  <>fGeneric :=True
  <>f_Version6x20 :=True
  <>fJ_Steinman :=True

End if

  ` Vytvoření parametrů
C_POINTER($1)                       ` Ukazatel k tabulce

pTable := $1                ` Nastaví ukazatel k tabulkce jako procesní proměnnou
READ ONLY (*)                       ` Nastaví všechny tabulky na pouze číst
DEFAULT TABLE (pTable->)
```

```
   ` Konec metody
```

3. Vytvořte novou metodu projektu LOCK_Tables2ReadWrite následovně:

```
If (False)
   ` Metoda: LOCK_Tables2ReadWrite (pointer; {pointer}; ...)
   ` ACI Univerzita kurzy programování
   ` Generic ACI Shell Programming
   ` Vytvořeno: Jim Steinman
   ` Datum: 1/16/97
   ` Účel: Nastaví tabulky pro všechny ukazatele přiřazené jako parametry na Read-Write

  <>fGeneric :=True
  <>f_Version6x20 :=True
  <>fJ_Steinman :=True

End if

C_LONGINT($i)

For ($i; 1; Count parameters)
  READ WRITE (${$i}->)          ` Použije bezcílný parametr
End for
  ` Konec metody
```

4. Modify the project method called P_Customers as follows:

```
If (False)
   ` Metoda: P_Customers
   ` ACI Univerzita kurzy programování
   ` Vytvořeno: Jim Steinman
   ` Datum: 1/15/97

   ` Účel: Zobrazí výstupní formulář pro tabulku zadanou ukazatelem pTable.

  <>f_Version6x10 :=True
❖ <>f_Version6x20 :=True
  <>fJ_Steinman :=True

❖    ` Upraveno: 1/17/97
•  <>fK_Wilbur :=True
End if

•  INITIALIZE_Process (->[Zákazníci])
•  LOCK_Tables2ReadWrite (pTable)
❖  pTable:=->[Zákazníci]

   GEN_ModifySelection
```

```
    ` Konec metody
```

5. Modify the project method called P_Products as follows.

```
If (False)
    ` Metoda: P_Products
    ` ACI Univerzita kurzy programování
    ` Vytvořeno: Jim Steinman
    ` Datum: 1/15/97

    ` Účel: Zobrazí výstupní formulář pro tabulku zadanou ukazatelem pTable.

  <>f_Version6x10 :=True
❖ <>f_Version6x20 :=True
  <>fJ_Steinman :=True

❖   ` Upraveno: 1/17/97
•   <>fK_Wilbur :=True
End if

•  INITIALIZE_Process (->[Produkty])
•  LOCK_Tables2ReadWrite (pTable)
❖  ~~pTable:= ->[Produkty]~~

   GEN_ModifySelection

  ` Konec metody
```

6. Změňte následovně metodu projektu P_Invoices.

```
If (False)
    ` Metoda: P_Invoices
    ` ACI Univerzita kurzy programování
    ` Vytvořeno: Jim Steinman
    ` Datum: 1/15/97

    ` Účel: Zobrazí výstupní formulář pro tabulku zadanou ukazatelem pTable.

  <>f_Version6x10 :=True
❖ <>f_Version6x20 :=True
  <>fJ_Steinman :=True

❖   ` Upraveno: 1/17/97
•   <>fK_Wilbur :=True
End if

•  INITIALIZE_Process (->[Faktury])
•  LOCK_Tables2ReadWrite (pTable; ->[PoložkyFaktury])
❖  ~~pTable:= ->[Faktury]~~

   GEN_ModifySelection
```

```
` Konec metody
```

7. Změňte následovně metodu projektu P_DailySales

```
…
READ ONLY (*) ` Nastaví všechny tabulky na pouze číst
MENU BAR (4)

<>LNotifyPid := New process ("P_Notifier"; 16000; "$Notifier")
…
```

8. Restartujte databázi a v prostředí uživatele otestujte stav pouze číst.

14.5. Příkazy, které pomáhají při testování a zacházení s zamčenými záznamy

14.5.1. Locked ([Tabulka]) -> Logické

Tento příkaz testuje zda platný záznam v daném procesu je uzamčen. Vrací True v následujících případech:

- Záznam byl uzamčen jiným uživatelem nebo procesem, který je ve stavu číst/psát
- Tabulka je v tomto procesu ve stavu pouze číst
- Ukazatel na platný záznam neukazuje na žádný záznam
- Platný záznam byl vymazán jiným uživatelem nebo procesem

Funkce Locked vám neřekne proč je záznam uzamčen. Pokud chcete zjistit znovu stav záznamu a chcete jej pro zázpis musíte záznam znovu zavést.

14.5.2. LOAD RECORD ([Tabulka])

Jestliže je tabulka ve stavu číst/psát, tento příkaz zavede záznam a uzamkne jej pro další uživatele/procesy. Od tohoto okamžiku tento proces může měnit záznam a ostatní uživatelé nemohou. Jestlliže je tabulka ve stavu pouze číst, záznam je zaveden, ale není pro další uživatele uzamčen. Záznam nemůže být v tomto procesu měněn.

14.5.3. UNLOAD RECORD ([Tabulka])

Tento příkaz odstraní platný záznam z paměti (nepřemístí ale ukazatel na platný záznam). Jestliže byl záznam uzamčen pro jiné uživatele/procesy je odemčen. Měli by jste volat UNLOAD RECORD k ošetření toho, aby záznam nezůstal uzamčen pro ostatní uživatele/procesy.

14.5.4. Testování před změnou stavu, zda tabulka je nastavena na pouze číst

Funkce - Read only state – vám sdělí jestli je tabulka nastavena do stavu pouze číst. Jestliže prověříte tento stav před dočasnou změnou stavu, dozvíte se jestli po provededení změn musíte tabulku opět nastavit do stavu pouze číst. Tento přístup zabrání, aby jste tabulku náhodně nenastavili na pouze číst, když jiné následné metody budou tuto tabulku potřebovat ve stavu číst/psát.

14.6. Testování před prováděním změn, zda záznam nebyl zatím vymazán

Často je nezbytné naztavit tabulku do stavu číst/psán jenom na okamžik, kdy jsou provedeny změny. Běžně je to jednoduché, pokud však někdo jiný neprovádí změny nebo záznam nebyl mezitím vymazán. Začnete nastavením tabulky do stavu číst/psát READ WRITE, zavedete záznam a pak otestujete zde jiný uživatel neprovádí změny, nebo zda nebyl záznam vymazán. Jestliže byl záznam vymazán funkce Locked navrátí True. Předpokládejme, že testujete záznam ve smyčce a čekáte dokud záznam nebude odemčen. Pokud byl záznam vymazán funkce Locked bude vracet True stále, takže smyčku nikdy neopustíte.

Naštěstí můžete ještě použít volání LOCKED ATTRIBUTES ([Tabulka]; $process; $uživatel; $počítač; $názevprocesu) k určení, zda záznam nebyl právě vymazán. Rovněž vám tento příkaz sdělí kdo a kde záznam uzamkl. Tato informace je táž, kterou vidíte při automatickém hlášení uzamčenosti při použití MODIFY SELECTION. Při vymazání záznamu vrátí LOCKED ATTRIBUTES –1 jako IDprocesu).

Abychom se vyhnuli problémům se zamčenými záznamy z důvodů nedávného vymazání, jsou ještě jiná řešení. Nejjednodušší je nepovolit mazání. Jiné je nepovolit mazání, když jsou používány procesy tabulek. Toto řešení znamená pokud je třeba vymazat záznam, označit si nějaký přepínač a mazat později, když uživatelé opustí databázi. Při dalším použití a načtení nových výběrů s novými stavy záznamů je již vše v pořádku.

Zda je tento problém význačný, nebo ne záleží na druhu vaší databáze. Jestliže máte databázi s malým počtem záznamů a záznamy jsou téměř neustále přidávány a mazány je tento problém poměrně význačný. Jestliže máte velkou databázi a je mazání záznamů je poměrně řídké, není tento problém tak význačný. Samozřejmě záleží také na druhu a význačnosti informací, keré se v tabulce ukládají a proč se provádějí změny.

14.7. Zavedení metod k úpravám vztažených záznamů

Když potřebujete změnit záznam v tabulce, která není hlavní tabulkou a tabulka byla nastavena READ ONLY, musíte dočasně nastavit tabulku READ WRITE a pak znovu zavést záznam, který chcete upravit.

14.7.1. Dočasné nastavení tabulky READ WRITE

1. Vytvořte novou metodu projektu GEN_CustomConfirm následovně, nebo importujte z textového souboru:

```
If (False)
      ` Metoda: GEN_CustomConfirm (String)
      ` ACI Univerzita kurzy programování
      ` Generic ACI Shell Programming
      ` Vytvořeno: Jim Steinman
      ` Datum: 1/16/97

      `Účel: Zobrazí vlastní zprávu potvrzení

   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

End if

` Vytvoření parametrů
C_STRING(255;$1)                    ` Toto je zpráva k zobrazení

` Vytvoření místních proměnných
C_LONGINT($LWindowID)

$LWindowID :=WIN_LNewWindow (400;160;<>WIN_LUpper)
sMessage :=$1
DIALOG([zDialogy];"GEN_CustomConfirm")
CLOSE WINDOW
      `Konec metody
```

2. Vytvořte novou metodu projektu LOCK_LSet2ReadWrite následovně, nebo importujte z textového souboru:

```
If (False)
      ` Metoda: LOCK_LSet2ReadWrite (pointer; boolean)
      ` ACI Univerzita kurzy programování
      ` Generic ACI Shell Programming
      ` Vytvořeno: Jim Steinman
      ` Datum: 1/16/97

      ` Účel: Nastaví tabuky na číst psát a načte záznam

   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

End if
```

```
    ` Vytvoření parametrů a přeřazení místních proměnných
C_POINTER($1;$pTable)                  ` uakaztel k tbulce k nastavení číst psát.
C_BOOLEAN($2;$fAskUserIfLocked) ` nechat uživatele odejít pokud je to vyžadované.
C_LONGINT($0)          ` ($0-1=READ ONLY 0=READ WRITE -1=Záznam je vymazán
nebo uživatel zrušen

    ` Vytvoření místních proměnných
C_BOOLEAN ($fOriginallyReadOnly) ` Tabulka originálně nastavena na číst psát
C_LONGINT ($LProcess)                  ` ID Procesu který má uzamčený záznam nebo –1
pokud je vymazán
C_LONGINT ($OK)                        ` Možná OK flag
C_STRING (40; $sUserName)                  ` Jménu uživatele který má zamčený záznam
C_STRING (40; $sMachine)                   ` Název stroje který má zamčený záznam
C_STRING (40; $sProcessName)       ` Název procesu který má zamčený záznam
C_TIME($hBeginWait)                        ` Počátek čekacího času k dalšímu dotazu na
uživatele

    `Přiřazení hodnot k místním proměnným
$pTable := $1
$fAskUserIfLocked := $2
$hBeginWait := †00:00:00†
$fOriginallyReadOnly := Read only state ($pTable->)
$0 := Num ($fOriginallyReadOnly)

READ WRITE ($pTable->)
LOAD RECORD ($pTable->)
$OK := 1

While (Locked ($pTable->) & ($OK = 1))
  LOCKED ATTRIBUTES ($pTable->; $LProcess; $sUserName; $sMachine;
$sProcessName)

  If ($LProcess = -1)                  ` Záznam byl vymazán
    If ($fAskUserIfLocked)                     ` Pokud není tázán uživatel možná vše
zastavil ALERT
      ALERT ("Záznam byl mezitím vymazán.")
    End if
    $0 := -1
    $OK := 0

  Else

    If ($fAskUserIfLocked)                 ` Nechme uživatele odejít pokud chce
      If ($sUserName="")
        $sUserName := "vy"
      End if

      If ($hBeginWait=†00:00:00†)
        $sPeriod:="již "
      Else
```

```
            $sPeriod:=" stále "
         End if

         GEN_CustomConfirm ("Záznam je"+$sPeriod+"používán
uživatelem"+$sUserName+" v procesu  "+ $sProcessName+"."+Char(Carriage
return)+"Chcete čekat?")

          If (OK = 1)
            $fAskUserIfLocked:= False          `Nastaví flag že se nechceme ptát
            $hBeginWait:=Current time
         Else
            $0 := -1
            $OK := -1
         End if

      End if

      If (OK = 1)
         PROCESS _MyDelay (Current process; 120)          ` Čekat dvě sekundy
         LOAD RECORD ($pTable->)
      End if

      If ($fAskUserIfLocked#$2)          ` Uživatel vybral čekat, ale ne dlouho!
         If (($hBeginWait+(2*60))<(Current time+0))          ` 2 minuty
            $fAskUserIfLocked:=$2          ` Nastaví flag že budeme ještě čekat
         End if
      End if

   End if                                    ` $LProcess > 0

End while

If (($fOriginallyReadOnly) & ($0 = -1))          ` Obnovit tabulku na čtení pouze jestli je
problém
   READ ONLY ($pTable->)
End if
   ` Konec metody
```

3. Vytvořte novou metodu projektu LOCK_Set2ReadOnly následovně, nebo importujte z textového souboru:

```
If (False)
      ` Metoda: LOCK_Set2ReadOnly (pointer; longint)
      ` ACI Univerzita kurzy programování
      ` Generic ACI Shell Programming
      ` Vytvořeno: Jim Steinman
      ` Datum: 1/16/97

      ` Účel: Když je potřeba nastaví tabulku na pouze číst
```

```
   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

End if

      ` Vytvoření parametrů a přeřazení místních proměnných
C_POINTER($1;$pTable)                  ` uakazatel k tbulce k nastavení číst psát.
C_LONGINT($2;$LRestoreState)           ` hodnota ukazující originální stav tabulky.
                              ` ($0-1=READ ONLY 0=READ WRITE -1=záznam vymazán
nebo uživatel zrušen

      `přiřazení hodnot k místním proměnným
$pTable := $1
$LRestoreState := $2

If ($LRestoreState = 1)
   READ ONLY ($pTable->)
   LOAD RECORD ($pTable->)
End if

   ` Konec metody
```

4. Otevřete formulář [Faktury]; "Vstupní".
5. Nahraďte metodu objektu tlačítka bUpravit následujícím, nebo importem pomocí textového editoru:

```
If (False)
      ` Object Metoda: bModify [Invoices]; "Vstupní"
      ` ACI Univerzita kurzy programování
      ` Vytvořeno: Jim Steinman
      ` Datum: 1/15/97

      ` Účel: Umožní upravení záznamu v [Zákazníci]

   <>f_Version6x00 :=True
   <>f_Version6x10 :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

      ` Upraveno: 1/16/97
   <>fK_Wilbur :=True
      ` Upraveno: 1/17/97
   <>fK_Wilbur :=True

End if

C_LONGINT ($LTableState)
$LTableState := LOCK_LSet2ReadWrite (->[Zákazníci]; True)
```

```
If ($LTableState # -1)
   MODIFY RECORD ([Zákazníci];*)
   LOCK_Set2ReadOnly (->[Zákazníci]; $LTableState)
End if

WIN_InputWindowTitle

   ` Konec metody
```

6. Otevřete formulář [Zákazníci]; "Vstupní".
7. Nahraďte metodu objektu tlačítka bOpen nástedujícím: (Pozn.: Tlačítko je na stránce 2 formuláře)

```
If (False)
      ` Object Metoda: bOpen [Zákazníci]; "Vstupní"
      ` ACI Univerzita kurzy programování
      ` Vytvořeno: Jim Steinman
      ` Datum: 1/15/97

      ` Účel: Umožní upravení označené faktury

   <>f_Version6x00 :=True
   <>f_Version6x10 :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

      ` Upraveno: 1/16/97
   <>fK_Wilbur :=True
      ` Upraveno: 1/17/97
   <>fK_Wilbur :=True

End if

C_LONGINT ($LTableState1; $LTableState2)

$LTableState1 := LOCK_LSet2ReadWrite (->[Faktury]; True)
$LTableState2 := LOCK_LSet2ReadWrite (->[PoložkyFaktur]; True)

If (($LTableState1 # -1) & ($LTableState2 # -1))
   MODIFY RECORD ([Faktury];*)
   LOCK_Set2ReadOnly (->[Faktury]; $LTableState)
   LOCK_Set2ReadOnly (->[PoložkyFaktur]; $LTableState2)
End if

WIN_InputWindowTitle
      `Konec metody
```

8. Přejděte do prostředí Vlastí aplikace a oevřete několik procesů Faktur. Oevřete faktury od téhož zákazníka a pokuste se upravit záznam zákazníka ze vstupního formuláře faktury.

14.8. Někdy chybí varování

Tyto příkazy neupozorní automaticky uživatele jsou-li záznamy některé záznamy uzamčeny pro dané akce.

- Smyčky, které upravují záznamy
- APPLY TO SELECTION
- ARRAY TO SELECTION
- DELETE SELECTION
- DELETE RECORD

14.9. Texty automatických tlačítek

Protože 4D nevytváří v některých případech automatická upozornění musíte je tu vytvořit vy. Seznámili jste se s příkazy jako ALERT, CONFIRM a REQUEST, které zajišťují předání informace uživateli a případně i informaci od něj. Dialogy těchto příkazů mají automatická tlačítka OK a případně Storno. Máte možnost popisky těchto tlačítek změnit předáním dodatečných parametrů příkazu.

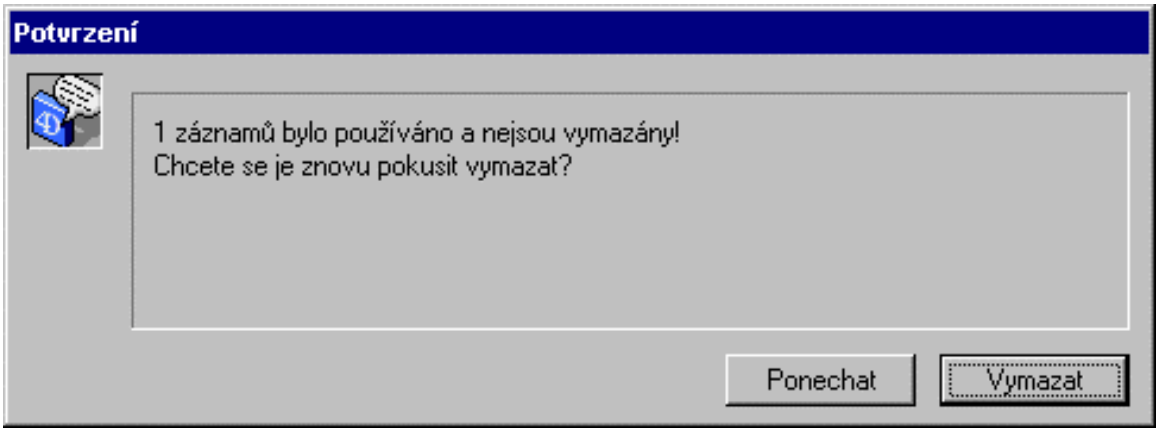
CONFIRM("TextUpozornění";{"Text OK tlačítka ";{"Text Storno tlačítka"}})

14.10. Sada LockedSet

Když použijete příkazy APPLY TO SELECTION, DELETE SELECTION nebo ARRAY TO SELECTION není automatické upozornění na uzamčenost záznamů. Místo toho se u těchto záznamů neprovedou změny a záznam samotný (odkaz) je odložen do sady nazvané LockedSet. Po užití těchto příkazů, by jste měli textovat zda je nějaký v sadě nějaký záznam a provést nějakou akci. Možné akce jsou:

- Informovat uživatele a nechat jej zrušit další proces a nechat zbylé záznamy nedotčeny.
- Informovat uživartele a nechat nezměněny nepovedené záznamy, převést je do platného výběru a zobrazit pro další rozhodnutí uživatele, co s nimi.
- Informovat uživatele a zeptat se jej zda chce čekat na odemčení zbývajících záznamů.
- Začít transakci před provedením příkazu, pokud jsou po provedení záznamy v sadě LockedSet zrušit všechny provedené změny zrušením transakce.

- Začít transakci před provedením příkazu, pokud jsou po provedení záznamy v sadě LockedSet zeptat se uživatele, zda chce čekat na odemčení záznamů, nebo chce uložit provedené změny, kromě uzamčených záznamů.



14.10.1. Kontrola LockedSet po provedení DELETE SELECTION

1. Nahraďte text metody projektu M_GEN_DeleteUserSet následujícím textem, případně jej importujte:

```
If (False)
      ` Metoda: M_GEN_DeleteUserSet
      ` ACI Univerzita kurzy programování
      ` Generic ACI Shell Programming
      ` Vytvořeno: Jim Steinman
      ` Datum: 1/15/97

      ` Účel: Stále vymazává označené záznamy

   <>fGeneric :=True
   <>f_Version6x10 :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

      ` Upraveno: 1/17/97
   <>fK_Wilbur :=True

End if

      `Define Local variables
C_LONGINT ($OK)                        ` Místní OK Flag
C_LONGINT ( $LNumberOfRecordsInSet)  ` Počet záznamů v Locked Set
C_LONGINT ($LProcessID)                  ` ID Procesu který uzamknul záznam(y)
C_LONGINT ($i)                         ` opakovací čítač
C_STRING (40; $sUserName)                ` Jméno uživatele ktrý uzamkl záznam
```

```
C_STRING (40; $sMachine)                    ` Název stroje který uzamkl záznam
C_STRING (40; $sProcessName)          ` Název procesu který uzamkl záznam

$LNumberOfRecordsInSet := Records in set ("UserSet")

If ($LNumberOfRecordsInSet > 0)
  CONFIRM ("Vymazat " + String ($LNumberOfRecordsInSet) + " označených záznamů?")

  If (OK = 1)
    CREATE SET (pTable->; "OriginalSelectionSet")          ` Vytvořit výběr
    USE SET ("UserSet")                                    ` Výběr se stane platným výběrem

    $OK := 1

    Repeat
      DELETE SELECTION (pTable->)

      If (Records in set ("LockedSet") > 0)
        $LNumberOfRecordsInSet := Records in set ("LockedSet")
        CREATE EMPTY SET (pTable->; "NonDeletedSet")
        USE SET ("LockedSet")

        For ($i; 1; $LNumberOfRecordsInSet)
          LOCKED ATTRIBUTES (pTable->; $LProcessID; $sUserName; $sMachine ;
                $sProcessName)

          If ($LProcessID # -1)                            ` Záznam zde stále je
            ADD TO SET (pTable->; "NonDeletedSet")
          End if

          NEXT RECORD (pTable->)
        End for

        $LNumberOfRecordsInSet := Records in set ("NonDeletedSet")

        If ($LNumberOfRecordsInSet > 0)
          CONFIRM (String ($LNumberOfRecordsInSet) + " bylo používáno a nejsou
vymazány!" + Char(Carriage return) + "Chcete se je znovu pokusit
vymazat?";"Vymazat";"Ponechat")

          If (OK = 0)
            ALERT ("nevymazané záznamy budou ukázány!")
            $OK := 0
          End if

          USE SET ("NonDeletedSet")
        End if

        CLEAR SET ("NonDeletedSet")

      Else
```

```
            USE SET ("OriginalSelectionSet")        ` Vrátit nevymazané záznamy do platného
výběru
            $OK := 1
          End if                                    ` Uzamčený výběr

       Until ($OK = 0)

     CLEAR SET ("OriginalSelectionSet")
     WIN_OutputWindowTitle
     End if                                         ` OK = 1

  Else
     ALERT ("Nejdříve musíte označit záznamy k vymazání!")
  End if                                            ` $LNumberOfRecordsInSet

     ` Konec metody
```

2. Přejděte do prostředí aplikace a otevřete dva procesy fakturace. Zaveďte nějakou fakturu poklepáním na ni a pak se ji v druhém procesu pokuste vymazat.

14.11. Vytvoření pořadového čísla ovládá zamykání záznamů

Jsou období, kdy musíte být schopni řídit úpravy záznamů. Předání dalšího pořadového čísla je jednou z těchto úloh. Pokud nemůžete změnit obsah záznamu s pořadovým číslem nelze vůbec pokračovat. Můžeme použít semafor k řízení přístupu k tabulce, ale to by byla ztráta času, protože ne všichni uživatelé přistupují k témuž záznamu pořadového čísla tabulky (Faktury, objednávky, zákazníci..) a zdržovali by se navzájem zbytzečně. Chceme dosáhnout pouze toho, že přistupují-li dva uživatelé k témuž pořadovému číslu (např. faktury) musí jeden počkat. K tomuto můžeme s výhodou použít vbestavěné uzamykání záznamů. Vytvoříme smyčku, v které bude uživatel čekat na odemčení záznamu, a to je běžná praxe.

14.11.1. Upravte metodu LNextSequence na uzamykání tabulek a zázmamů

1. Vytvořte novou metodu projektu LOCK_Tables2ReadOnly následovně:

```
If (False)
     ` Metoda: LOCK_Tables2ReadOnly (pointer; {pointer}; ...)
     ` ACI Univerzita kurzy programování
     ` Generic ACI Shell Programming
     ` Vytvořeno: Jim Steinman
     ` Datum: 1/16/97

     ` Účel: Nastaví všechny tabulky pro všechny ukazatele přiřazení jako parametry na Read-
Write
```

```
   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

End if

C_LONGINT($i)

For ($i; 1; Count parameters)
   READ ONLY (${$i}->)    ` Použije nesměrovaný ukazatel
End for

   ` Konec metody
```

2. Nahraďte obsah metody LNextSequence následujícím:

```
If (False)
      ` Metoda: LNextSequence (string; longint) -> longint
      ` ACI Univerzita kurzy programování
      ` Generic ACI Shell Programming
      ` Vytvořeno: Jim Steinman
      ` Datum: 1/15/97

      ` Účel: Udržuje tabulku čísel pro generování jedinečných řísel.

   <>fGeneric :=True
   <>f_Version6x10 :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

      ` Upraveno: 1/17/97
   <>fK_Wilbur :=True
End if

      ` Vytvoří parametry a přeřadí místní proměnné
C_STRING (31; $1;$sSequenceName)        ` Název sekvence pro
C_LONGINT ($2;$LReturnValue)            ` číslo k použití {Pouze pokud vrací číslo}
C_LONGINT ($0)          ` Další hodnota čísla pro navrácení do procesu

      ` Vytvoření místních proměnných
C_LONGINT  ($LParameters)

$LParameters := Count parameters

      `Přiřadí hodnoty k místním proměnným
$sSequenceName:= $1
If ($LParameters >1)
   $LReturnValue:= $2
End if
```

```
LOCK_Tables2ReadWrite (->[zSekvence]; ->[zNavrácenáČísla])

Case of

   : ($LParameters = 1)                                  ` Potřebujeme nové číslo
      QUERY ([zSekvence]; [zSekvence]SequenceName = $sSequenceName)

      If (Records in selection ([zSekvence]) = 1)

         While (Locked ([zSekvence]))     ` Další proces potřebuje číslo
            PROCESS _MyDelay (Current process; 10)
            LOAD RECORD ([zSekvence])
         End while

         RELATE MANY ([zSekvence]SequenceName)

         If (Records in selection ([zNavrácenáČísla]) > 0)    ` Testuje pro vracené číslo
            ORDER BY ([zNavrácenáČísla]Value)
            $0 := [zNavrácenáČísla]Value
            DELETE RECORD ([zNavrácenáČísla])
         Else
            $0 := [zSekvence]NextValue
            [zSequences]NextValue := [zSekvence]NextValue + 1
            SAVE RECORD ([zSekvence])
         End if

      Else                                                 ` Položka nikdy neexistovala.
         CREATE RECORD ([zSekvence])
         [zSekvence]SequenceName := $sSequenceName
         [zSekvence]NextValue := 2
         $0 := 1
         SAVE RECORD ([zSekvence])
      End if

   Else                                                    ` Vracíme číslo do databáze
      CREATE RECORD ([zNavrácenáČísla])
      [zNavrácenáČísla]SequenceName := $sSequenceName
      [zNavrácenáČísla]Value := $LReturnValue
      SAVE RECORD ([zNavrácenáČísla])
      UNLOAD RECORD ([zNavrácenáČísla])
      $0 := 0
End case

UNLOAD RECORD ([zSekvence])
LOCK_Tables2ReadOnly(->[zSekvence]; ->[zNavrácenáČísla])

   ` Konec metody
```

3. Vraťte se do prostředí Vlastní aplikace a otestujte.

14.12. Simulace více uživatelů procesy

Uzamykání záznamů může být testováno použitím více procesů na jednom počítači. Tento způsob ušetří čas a potřebu testovat na 4D Server více uživatelů a více počítačů v síti. Jsou však jiné důvody proč nakonec databáze musí být testována se 4D Server, jsou to především důvody chování a optimalizace rychlosti s více počítači a více uživateli.

15. 4th Dimension product line

15.1. Produkty pro jednoho uživatele

Řada produktů 4th Dimension zahrnuje následující jednouživatelské produkty:

- 4D (nástroj vývojáře, který se spouští pro jednoho uživatele)
- 4D Runtime (jednouživatelský stroj, který spouští aplikace pouze v prostředí Vlastní nabídky a pouze v interpretačním módu (nekompilované databázi))
- 4D Runtime Classic (jednouživatelský stroj, který spouští aplikace pouze v prostředí Vlastní nabídky a to pouze kompilované databáze)
- 4D First (zjednodušená verze jednouživatelské 4D)

15.2. Produkty pro více uživatelů

Rodina produktů 4th Dimension zahrnuje verzi klient/server, která obsahuje následující prvky:

- 4D Client (aplikace pro pracovní stanici)
- 4D Server (aplikaci pro databázový server)
- Network components – síťové komunikační protokoly (podpora Novell IPX/SPX, TCP/IP a AppleTalk ADSP)
- 4D Remote (vnitřní síťový protokol 4D pro komunikaci 4D Client a 4D Server přes telefonní linku)
- 4D Open (sada zásuvných příkazů/metod k připojení na 4D Server bez 4D Client)

15.3. Moduly připojení:

ACI vytvořila řadu zásuvných modulů, které dovolí 4D připojit se a komunikovat s různými servery a databázemi SQL:

- 4D SQL Server
- 4D for ORACLE
- 4D ODBC

Tyto moduly poskytují aplikacím 4D možnost fungovat jako front-end pro SQL Servery (Sybase, MS SQL, Oracle a databázové servery poskytující rozhraní ODBC).

15.4. Další zásuvné moduly (plug-ins)

- ACI Pack
- 4D Write

- 4D Calc
- 4D Draw
- DDE Tools (pouze pro Windows)
- DLL Tools (pouze pro Windows)
- OLE Tools (pouze pro Windows)

16. Technologie sdílení souborů

Podstata technologie klient/server bude mnohem jasnější jestliže ji porovnáme s její alternativou technologie sdílení souborů.

16.1. Víceuživatelské verze 4th Dimension 2.x

16.1.1. Data nemohou být vyrovnávána do paměti

Server souborů (file server) nemůže vyrovnávat data do paměti. Tento server je pasivní stroj: 4D je spuštěna pouze na pracovní stanici. Datový soubor zůstává na počítači serveru. Na počítači serveru neběží 4D takže nemůže rozumět tomu co je platný záznam, co je platný výběr atd. Protože server nerozumí struktuře dat 4D nemůže ani vyrovnávat data do paměti. Jediné co může počítač serveru dělat je obsluhovat a uzamykat určitý počet bytů pomocí systémového sdílení souborů dle požadavků 4D aplikace běžící na pracovní stanici. Systémové sdílení souborů později uloží data zpět do patřičných bytů a tyto byty odemkne takže k této části souborů dat bude mít přístup i jiný uživatel.

Pracovní stanice uživatele rovněž nemůže vyrovnávat data do paměti. Kdyby byla data vyrovnána do paměti na pracovní stanici 1 a uživatel na pracovní stanici 2 změnil záznamy odpovídající těmto datům nebyly by záznamy vyrovnané v paměti pracovní stanice 1 platné. Proto, vyrovnávání dat do paměti na místních pracovních stanicích není možné.

Protože nelze data vyrovnávat do paměti a je je nutno neustále číst z disku připojeného po síti. Je toto řešení pomalé. Je tomu tak proto, že přístup k disku a operace čtení zápis na disk jsou nejpomalejší operace na počítači.

16.1.2. Všechny operace s daty se musí vykonat na pracovn stanici

Protože je aplikace 4D spuštěna na pracovní stanici a nikoliv na serveru, nemůže server provádět operace 4D (např. dotazy) nad datovým souborem přímo na počítači serveru. Všechny operace nad daty se musí provést na uživatelské pracovní stanici. Tento způsob vyžaduje přenesení každého záznamu na pracovní stanici a výsledkem je samozřejmě značná degradace chování sítě. Proto je tato operace podstatně pomalejší než, kdyby byla jednoduše provedena na počítači serveru (kde jsou umístěna data). Řešení tohoto problému se na první pohled zdá jednoduché. Spustit 4D aplikaci na počítači serveru místo na každé uživatelské stanici a nechat každého uživatele používat 4D aplikaci ze stroje serveru. Představme si chování a odezvu, které bychom dostali od libovolné aplikace běžící pouze na serveru a ne na lokálním počítači; to je to co nyní požadujeme od 4D. Každý uživatel by neustále čekal na ukončení operace jiného uživatele a ukončení

jeho přístupu k datům. Stroj serveru by musel neustále interpretovat a provádět kód pro všechny operace všech uživatelů a musel by rovněž zajistit jejich přístup dat (čtení a zápis na disk) pro tyto operace všech uživatelů.

Příklad operací se sdílením souborů: Dotazy – děje se mnoho přístupů na disk na serveru a přenos dat přes síť. V jedné operaci je poslán přes síť pouze jeden záznam, tento záznam je zaveden do paměti pracovní stanice. Jestliže záznam vyhovuje kritériu dotazu je nastavena hodnota True a záznam je přidán do platného výběru. Jestliže záznam nevyhovuje kritériím dotazů je nastavena hodnota False a záznam není přidán do platného výběru. Záznam je pak vyveden z paměti a je požadován další záznam ze serveru.

Záznamy jsou posílány přes síť dvakrát pokud vyhoví kritériu dotazu, jednou při provádění dotazu a podruhé při vyhodnocení výsledku a zobrazení výběru na obrazovku. I při zobrazení výsledného výběru na obrazovku jsou záznamy do výstupního formuláře zasílány jeden po druhém.

17. 4D Server technologie klient/server

V prostředí klient/server jsou úlohy rozumně rozděleny mezi jednotlivé komponenty na počítačích klienta (pracovní stanici) a serveru. 4D Server není vyjímkou.

17.1. Úlohy prováděné 4D Server

17.1.1. Vyrovnávání dat do paměti (caching)

Narozdíl od serveru souborů, 4D Server rozumí co je záznam. Takže server může vyrovnávat data do paměti. 4D Server rovněž rozumí co je platný výběr. Takže server může jako odpověď na dotaz vyrovnat do paměti platný výběr. Čím více záznamů obsahuje výběr a čím více záznamů je vyrovnáno do paměti tím lepší je chování pro další operace, které přistupují k těmto záznamům.

17.1.2. Provádění operací nad daty

Protože platný výběr může být uložen v paměti na serveru, mohou být operace zacházející s platným výběrem provedeny na serveru. Dotazy se vykonají na serveru a pak je na pracovní stanici zaslán platný výběr v optimalizované verzi jeden obsah okna současně. Třídění se provádí na serveru. 4D Server používá více úloh sdílejících čas k současnému provádění více operací.

17.1.3. Provádění indexace

Protože jsou data uložena na serveru i úlohy indexování jsou prováděny přímo na serveru.

17.1.4. Provádění triggeru

Triggery jsou části 4D kódu přiřazené přímo tabulkám a prováděné po každém přístupu k záznamu. Nezáleží na tom kdo či co je zdrojem této akce.

Triggery se provádějí při následujících událostech databáze:

- Při uložení nového záznamu
- Při uložení záznamu
- Při mazání záznamu
- Při zavedení záznamu

Poznámka: Všechny triggery sdílejí pouze jednu sadu processních proměnných. Ve vašich triggerech by jste neměli používat proměnné procesu.

17.1.5. Provádění uložených procedur

Uložené procedury 4D Server jsou metody projektu 4D pro, které vývojář definuje, že se mají provést na počítači serveru a ne na počítači klienta.

4D Server, podobně jako 4th Dimension a 4D Client je schopen spustit najednou vedle sebe několik procesů díky konceptu sdílení času CPU.

17.2. 4D Client

4D Client provádí na pracovní stanici následující operace:

- Udržuje a ošetřuje rozhraní uživatele
- Zachycuje události způsobené uživatelem
- Překresluje obrazovku
- Vyrovnává platné záznamy do paměti
- Vyrovnává do paměti prvky struktury aplikace: záhlaví nabídek, formuláře, metody atd.
- Provádí požadovaný kód aplikace

Pokud vývojář neřekne jinak (uložené procedury) je všechen kód prováděn na klientu. Když 4D Client narazí na příkaz, který zachází s platným výběrem pošle zprávu na server k provedení této operace (plnění těchto úloh je standartní součástí 4D Server).

17.3. Provádění paralelního zpracování

Některé úlohy jsou rychlejší pod 4D Server než pod jednouživatelské 4D, protože v případě 4D Server pracujete s asynchronním zpracováním a komunikací mezi klientem a serverem.

18. Požadavky na hardware a úvahy o něm

18.1 Hardware 4D Server

18.1.1. Pevný disk serveru

Kupte si ten nejrychlejší disk jaký můžete dostat. Čím rychlejší ovladač tím rychlejší budou odpovědi serveru při přístupu k datům na disku. Zatím nejvýhodnější jsou disky s asynchronním přístupem SCSI II.

18.1.2. Paměť serveru

V paměti se ukládájí záznamy, výběry, indexy příp. formuláře, metody, nabídky.

Čím více RAM bude k dispozici tím rychlejší budou odezvy serveru na požadavky klienta

Příklad vyrovnávání dat

Jeden uživatel otevře databázi v Prostředí uživatele pak se přepne do jiné tabulky a následně zpět do první tabulky nebo otevře záznam a pak jej uzavře pro návrat do výstupního formuláře. Zobrazení záznamů je rychlejší, když se vrací do původní tabulky, protože záznamy byly již z disku v předešlé operaci přečteny a umístěny do paměti serveru. Při druhém přístupu byly již jen odeslány z paměti. Tento test může být proveden pouze pro prvního uživatele výběru záznamů, protože po jeho přístupu k výběru záznamů byl tento umístěn v paměti a ostatní uživatelé přistupují k témže datům v paměti.

Provádí triggery

Provádí uložené procedury

Úvahy pro koupi. Budete mít dost paměti RAM na stroji, který se chystáte používat? Počítače se systémem Windows 95 potřebují min. 16MB, počítače s Windows NT potřebují min. 24MB a počítače s MacOS potřebují min. 16MB. Pamatujte si, že toto jsou minimální nikoliv optimální hodnoty.

18.1.3. Rychlost CPU serveru

Rychlost CPU bude ovlivňovat operace, kterými jsou dotazy, třídění, vytváření array, ovládání a předávání výběru.

18.2. Hardware 4D Client

Počítač pro 4D Client je druhým nejdůležitějším faktorem pro chování databáze. Počítač klienta je zodpovědný za provádění kódu, dočasné ukládání formulářů, metod a dalších objektů struktury v paměti, udržuje platné záznamy pro každou tabulku, každý proces a překresluje formuláře. Čím rychlejší je stroj klienta, tím rychlejší budou tyto funkce.

18.2.1. Rychlost CPU klienta

CPU klienta provádí kód

Čím je CPU rychlejší tím rychleji bude aplikace prováděna, tím rychleji budou formuláře překresleny.

Rychlost CPU nemá žádný vliv na rychlost provádění dotazů a třídění, které se vykonávají na serveru.

18.2.2. Paměť klienta

RAM klienta ukládá formuláře, metody, nabídky, proměnné a platné záznamy.

Čím více RAM je k dispozici tím méně často bude potřeba znovu načítat formuláře a metody do paměti.

18.2.3. Pevný disk klienta

Pevný disk klienta ukládá některé zdroje aplikace přenesené ze 4D Server a to do souboru aplikace název aplikace.res pro zdroje typu textových řetězců, knihoven obrázků atd. a soubor název aplikace.rex pro zdroje typu formulářů a metod.

Rychlost místního pevného disku bude mít vliv na přístup k objektům struktury a zdrojům. Při dostatku paměti RAM není ovlivnění běhu aplikace rychlostí místního pevného disku příliš významné.

18.3. Úvahy o síti

Pro chování aplikace je rovněž důležitým faktorem síť. Požadavky na rychlost zavisí na množství dat, která budou přenášena po síti a používána síťovými protokoly. Který protokol je nejrychlejší závisí na vašem zcela určitém nastavení sítě. Doporučujeme vám vyzkoušet všechny dostupné protokoly při běžném zatížení aplikace a na testech určit, který protokol je nejlepší pro vaši konkrétní konfiguraci.

19. Zatížení sítě způsobené 4D Server

19.1. Při přihlášení uživatele

- Informace o struktuře a aplikaci jsou zaváděny ze serveru na klienta
- Je provedena metoda databáze Při spuštění

19.2. Když uživatel otevře formulář jsou zavedeny následující položky:

- Formulář
- Metoda formuláře
- Metody objektu jsou zavedeny v případě potřeby
- Pole a záznamy potřebné pro zobrazení jsou zavedeny podle druhu formuláře
  Výstupní formuláře:
  Jsou zavedena pouze pole na formuláři pro záznamy zobrazené v jednom okně.
  Vstupní formuláře:
  Jsou zavedena všechna pole záznamu ať jsou či nejsou na formuláři potřeba.

19.3. Když uživatel provádí dotaz

- Kritérium dotazu je předáno na server
- Je-li zobrazován formulář jsou zpět na klienta předána potřebné pole a to podle zásad uvedených výše.
- Jestliže není zobrazován formulář je předán pouze ukazatel na platný záznam a počet nalezených záznamů.

19.4. Když uživatel tiskne

- Formulář
- Metoda formuláře
- Metody objektů pro prováděné akce
- Celý záznam, který je tištěn

19.5. Když se uživatel odhlašuje

- Je uvolněna licence
- Záznamy jsou odemčeny
- Jsou vymazány semafory

19.6. Když je uživatel náhodně odpojen (problémy na síti, v počítači klienta)

- Při použití ADSP nebo IPX, 4D Server uvolní licenci, odemkne záznamy, zruší transakce a vymaže semaformy během dvou minut

- Při použití TCP/IP uživatel musí být na serveru odstraněn manuálně akcí administrátora

20. Transakce

Transakce je sada návazných změn dat, následujících po sobě. Úpravy provedené během transakce nejsou uloženy trvale na disk dokud není transakce potvrzena. Jestliže není transakce dokončena buď proto, že byla zrušena nebo z nějaké vnější příčiny nejsou úpravy uloženy.

20.1. Příkazy transakce

Příkaz START TRANSACTION způsobí, že 4D uloží všechny změněné záznamy do dočasného paměťového bufferu. Pokud se buffer naplní změněnými daty začnou se tyto změny zapisovat na serveru do dočasného souboru. Jestliže se rozhodnete trvale uložit vaše změny použijete příkaz VALIDATE TRANSACTIONS. Tento příkaz způsobí, že jsou všechny dočasné buffery vytvořené během transakce zapsány do databáze trvale a transakce se ukončí. Jestliže se rozhodnete neuložit všechny provedené změny, použijte příkaz CANCEL TRANSACTION po tomto příkazu se transakce ukončí, dočasné buffery jsou vymazány a do databáze není uložena žádná zmněna. Transakce je způsob jak provádět změny do databáze a teprve v průběhu těchto změn rozhodnout zda tyto změny budou uloženy natrvalo či ne. Degradace chování databáze zabráníte pokud budete provádět co nejkratší transakce a ukončíte je dříve než bude 4D muset zapisovat změněná data na disk.

Po potvrzení nebo zrušení transakce se výběry pro daný proces stanou prázdnými, protože transakce zachází s dočasnými adresami záznamů. Ze stejného důvodu musíte opatrně zacházet uvnitř transakce s pojmenovanými výběry. Po potvrzení či zrušení transakce mohou pojmenované výběry vytvořené během transkace obsahovat nesprávné adresy záznamů. Např. pojmenované výběry mohou obsahovat adresy vymazaných záznamů nebo dočasné adresy záznamů přidané během transakce.

Totéž platí o sadách, protože jsou založeny na bitové reprezentaci tabulek adres záznamů.

20.2. Transakce a zamykání záznamů

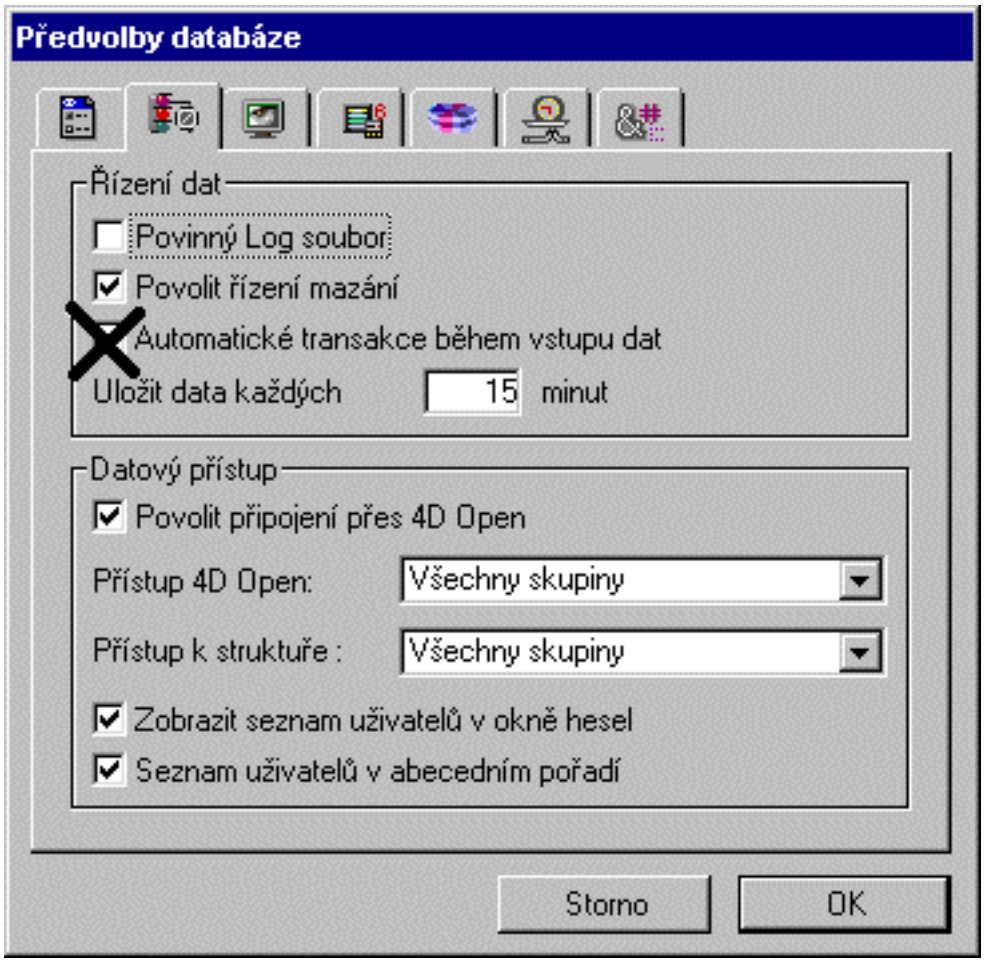
Všechny záznamy upravené během transakce jsou uzamčeny dokud transakce neskončí. Proto s ohledem na tento fakt je obzvlášť důležité používat co nejkratší transakce.

Typické použití pro transakci je, když výsledkem dané akce musí být, že budou buď dokončeny všechny položky nebo žádná položka. Vezměme si např. výdej ze skladu, kde musí být vydány buď všechny položky objednávky nebo žádná. Každá z položek musí být označena jako vydaná ze skladu a skutečně ze skladu odečtena. V tomto případě nelze vydat některé z položek a nevydat jiné, proto následuje přehled kroků nezbytných k vydání položek ze skladu. Skutečný kód zde není uváděn, protože jeho pročtení by zabralo příliš času a není pro náš účel podstatné.

- Označení pole ve faktuře určující, že zboží z faktury bylo vydáno.
- Označení pole ve faktuře určující, že faktura byla dodána.
- Zahájit smyčku přes všechny položky faktury, která odečte množství z faktury ze skladu (produktu) a uloží jestliže je zboží dostupné záznam produktů. Pokud zboží není na skladě, přeruší celou transakci.
- Uložení záznamu faktury při úspěšném výdeji všech položek.
- Uložení a potvrzení transakce a přechod na další fakturu.

20.3. Automatické transakce během vstupu dat

Od verze 3.0. zavádí 4D rys nazvaný automatické transakce během vstupu dat. Tento rys je automatickou akcí, která může sloužit v jednodušších databázích, ale není příliš výhodná z hlediska chování velké databáze a mnoha uživatelů. Transakce zatěžují paměť a při více uživatelích a mnoha procesech může dojít k přílišnému zatížení. Děje se tak proto, že při každém vstupu do vstupního formuláře je zahájena transakce. Každá transakce zapisuje všechny změny do odděleného dočasného souboru na serveru. Potom, kdy je transakce potvrzena čte všechny provedené změny z dočasného souboru a přidává tyto změny do databáze. To vytváří automaticky velké zatížení serveru. Server musí zapsat data dvakrát (jednou do dočasného souboru a podruhé do trvalého datového souboru). V době velkého zatížení může být výsledkem zatížení serveru větší až o 50% oproti stavu, kdy nejsou transakce používány. Automatické transakce během vstupu dat vytvářejí transakci pokaždé, když uživatel otevře vstupní formulář. Tento automatický rys může přespříliš zatěžovat server. Místo toho odškrtněte v předvolbách databáze políčko Automatické transakce během vstupu dat a použijte příkazy jazyka 4D k zavedení transakcí pouze do míst, kde jsou potřeba.



20.3.1. Vytvoření transakcí příkazy jazyka místo automaticky

1. Přejděte do Prostředí návrháře, zvolte Soubor ¬ Předvolby databáze… a odškrtněte políčko Automatické transakce během vstupu dat.
2. Otevřete metodu formuláře [Faktury]Vstupní.
3. Upravte metodu formuláře následujícím způsobem:

```
If(False)
    ` Form Metoda: Input
    ` ACI Univerzita kurzy programování
    ` Vytvořeno: Jim Steinman
    ` Datum: 15/1/97

    ` Účel: Metoda vstupního formuláře [Faktury]; "Vstupní"

  <>f_Version6x10 :=True
```

```
❖          <>f_Version6x20 :=True
           <>fJ_Steinman :=True

❖            ` Upraveno: 1/17/97
❖          <>fK_Wilbur :=True

        End if

        C_LONGINT($LFormEvent)

        $LFormEvent :=Form event

        Case of

          : ($LFormEvent = On Load)

            fNewInvoice := False
            If (pTable = (->[Customers]))` Přicházíme z tabulky Zákazníci pomocí tlačítka bOpen
              DISABLE BUTTON (bModify)  ` Nemůže provádět funkce na faktuře
              SET WINDOW TITLE ("Upravuje se faktura " + String ([Faktury]ČísloFaktury))
            Else

              If (Record number([Invoices]) = -3) `Nový záznam
                fNewInvoice := True
                [Invoices]InvoiceNo := LNextSequence ("InvoiceNo")
                GEN_TimeStamp (->[Invoices]DateCreated; ->[Invoices]TimeCreated;)
              End if
              WIN_InputWindowTitle
            End if

❖          START TRANSACTION

        End case

          ` Konec metody
```

4. Upravte metodu projektu E_FakturyStorno následovně:

```
If (False)
    ` Metoda: E_Faktury Storno
    ` ACI Univerzita kurzy programování
    ` Generic ACI Shell Programming
    ` Vytvořeno: Jim Steinman
    ` Datum: 15/1/97

    ` Účel: Zruší transakce a obnoví staré informace
                CANCEL

  <>f_Version6x10 :=True
❖ <>f_Version6x20 :=True
  <>fJ_Steinman :=True

❖   ` Upraveno: 1/17/97
❖ <>fK_Wilbur :=True

End if

C_LONGINT($x)

❖ CANCEL TRANSACTION

If (fNewInvoice)
  $x := LNextSequence ("ČísloFaktury"; [Faktury]ČísloFaktury)
End if

  ` Konec metody
```

5. Otevřete formulář [Faktury]Vstupní.
6. Vytvořte metodu objektu pro tlačítko bValidate následujícím způsobem nebo importujte pomocí textového editoru:

```
If (False)
    ` Object Metoda: bValidate  [Faktury];"Vstupní“
    ` ACI Univerzita kurzy programování
    ` Generic ACI Shell Programming
    ` Vytvořeno: Kent Wilbur
    ` Datum: 17/1/97

    ` Účel: Uloží a potvrdí transakci

  <>f_Version6x20 :=True
  <>fK_Wilbur :=True

End if
```

```
SAVE RECORD ([Faktury])
SAVE RECORD ([PoložkyFaktury])

VALIDATE TRANSACTION

If (fNewInvoice)
   ACCEPT
Else
   CANCEL
End if

   ` Konec metody
```

7. Vytvořte metodu objektu pro tlačítko bFirst následujícím způsobem:

```
If (False)
      ` Object Metoda: bFirst  [Invoices];"Input“
      ` ACI Univerzita kurzy programování
      ` Generic ACI Shell Programming
      ` Vytvořeno: Kent Wilbur
      ` Datum: 17/1/97

      ` Účel: Uloží a potvrdí transakci

   <>f_Version6x20 :=True
   <>fK_Wilbur :=True

End if

SAVE RECORD ([Faktury])
SAVE RECORD ([PoložkyFaktury])

VALIDATE TRANSACTION

FIRST RECORD ([PoložkyFaktury])

   ` Konec metody
```

8. Vyberte tlačítka bValidate; bFirst, bPrevious, bNext a bLast a pro všechna zvolte automatickou akci Žádná (zlepší chování 4D Server tak, že nebude vyžadovat uložení záznamu více než jednou).
9. Pokračujte v přidávání kódu podobném předchozímu pro bFirst a s úpravami pro tlačítka bPrevious, bNext a bLast buttons (jediným rozdílem v kódu bude řádek vyžadující patřičnou akci přechodu mezi záznamy výběru).
10. Vraťte se do prostředí Vlastní nabídky a proveďte testování.

20.4. Transakce a 4D Backup.

Protože transakce uzamyká všechny záznamy, které byly změněny, čeká 4D Backup na dokončení všech transakcí předtím než začne provádět zálohování. Pokud je 4D Backup spuštěn nemůže se nový uživatel do databáze připojit do doby dokončení operace. Tyto dva rysy v kombinaci mohou potenciálně uzamknout všechny nepřipojené uživatele vně databáze. Příklad: jeden uživatel opustí svůj stroj, když je ve vstupním formuláři, ve kterém je spuštěna transakce. Podle časového harmonogramu je vyvolán 4D Backup a snaží se začít zálohování, musí však čekat dokud tento uživatel nedokončí transakci, což se však nestane, protože uživatel opustil svůj počítač. Z tohoto důvodu je dobré používat transakce uvážlivě a vyškolit vaše uživatele, aby neopouštěli počítače dokud nedokončí své úlohy a neuzavřou vstupní formuláře nebo nejlépe všechna otevřená okna.

20.5. Tipy a rady pro transakce

20.5.1. Nepoužívejte UNLOAD RECORD v transakci

Příkaz UNLOAD RECORD se pokusí vyvést a odemknout záznam. Jestliže byl záznam změněn, transakce chce udržet záznam uzamčený, protože byl změněn v transakci. Tento příkaz ve svém důsledku neprovede nic vyjma toho, že server musí pracovat o něco více.

20.5.2. Příkaz DISTINCT VALUES se chová speciálním způsobem

Protože změněné záznamy a nové záznamy jsou uloženy ve speciálním odděleném bufferu, příkaz DISTINCT VALUES pracuje pouze na základě původních indexů a ne hodnot upravených během transakce. Z tohoto důvodu používejte příkaz DISTINCT VALUES uvnitř transakce po zralé úvaze.

20.6. Záměrné uzamčení záznamů transakcí

Příležitostně může být výhodné uzamknout určitou sadu záznamů pro všechny ostatní uživatele. Příklad: všechny vybrané záznamy musí být obnoveny společně bez možnosti narušení. Dále je uveden koncept provádějící tento záměr, započetí transakce, úprava neindexovaného pole zpět na svou vlastní hodnotu s použitím příkazu APPLY TO SELECTION, kontrola zda nějaký záznam není uzamčen, jestliže ne pokračování, jestliže ano přerušení operace.

```
START TRANSACTION
APPLY TO SELECTION([Tabulka];[Tabulka]NeindexovanéPole :=
[Tabulka]NeindexovanéPole )

If(Records in set("LockedSet")=0)

   ` Zde proveďte požadovanou akci
```

```
   VALIDATE TRANSACTION

Else

   CANCEL TRANSACTION
End if
```

Poznámka: Výběr neindexovaného pole proveďte v následujícím pořadí: Integer; LongInteger; Krátké Alfa; Logické.

21. Komunikace mezi pracovními stanicemi

Obecně není do 4D zabudován žádný prostředek automatické komunikace mezi pracovními stanicemi. Jeden ze způsobů může být vytvoření speciální tabulky pro vkládání zpráv jedné pracovní stanice pro druhou. Toto řešení je funkční v některých případech. Ale je potřeba přidat kroky k přinucení pracovní stanice neustále kontrolovat tuto tabulku. Je potřeba ošetřit případy, kdy stanice nepracuje a neodpovídá na zprávu a způsob jak a kdo vymazává zprávy, které již nejsou potřeba.

21.1.Použití globálních semaforů

Již dříve jsme zmínili funkci semaforu a použili lokální semafory, které jsou vhodné jako přepínač a ukazatel na určité věci, na které je potřeba upozorňovat jestliže jsou použity dva nebo více procesů na témže počítači a tyto procesy se snaží provádět stejnou věc ve stejnou dobu. Tentýž základní koncept existuje mezi pracovními stanicemi a nazývá se globální semafor. Globální semafor je příznak či přepínač používaný ke komunikaci mezi pracovními stanicemi.

Proměnné existují v paměti na klientu. Proto jejich hodnoty nemohou být viděny jinými klienty. Pouze hodnoty v polích nebo globální semafory mohou být viděny a zpřístupněny všem klientům. Globální semafor je držen v paměti na počítači serveru. Globální semafor je příznak celé databáze a jeho existence je buď True nebo False pro všechny uživatele databáze.

21.2. Příklad použití globálního semaforu

Řízení provádění kódu mezi uživateli

Globální semafor může být použit pro zajištění toho, že v daném čase bude určitou akci či metodu provádět pouze jeden uživatel. První uživatel vytvoří semafor a nikdo jiný po dobu existence tohoto semaforu nemůže provést danou akci (např. výdej ze skladu pro určitou objednávku nebo potvrzení faktury).

```
If (Semaphore ("Posting"))
   ALERT ("Nyní nemůžete potvrzovat, zkuste později.")
Else
   POSTING `Spuštění metody potvrzení
   CLEAR SEMAPHORE ("Posting")
End if
```

Udržení odpojených uživatelů mimo databázi a nepovolení přístupu

Vynucení spuštění metody

Jestliže určitý semafor existuje bude spuštěna nějaká metoda. Např. při synchronizovaném zpracování tiskových úloh pro uživatele se zakázaným přístupem k lokální síti bude na pracovní stanici lokální sítě vyvolán tisk či export.

21.3. Použití záznamů v databázi ke komunikaci

Jiný způsob k přenosu informací mezi pracovními stanicemi je použití skutečných dat v databázi, zkuste následující kroky:

1. Vložte data do záznamů v databázi.
2. Nastavte semafor nebo jen opusťte data a pokračujte.
3. Spusťte obslužný počítač, který kontroluje existenci semaforu a pak kontroluje určitá data nebo pouze periodicky kontroluje určitou tabulku dat na určité záznamy.
4. Po vymazání dat, původní stroj musí vymazat semafor.

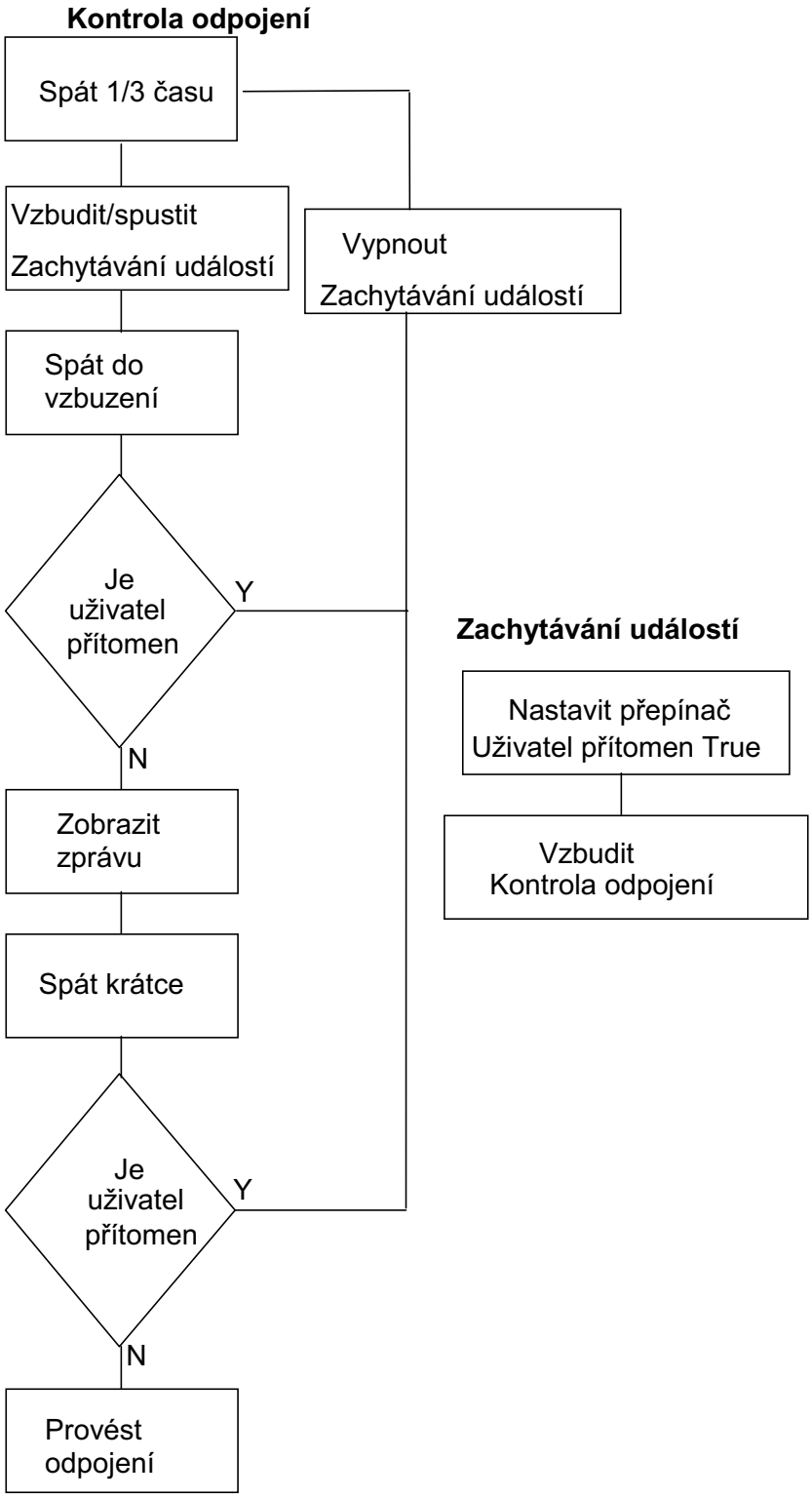
22. Otázky licencí a 4D Server

4D Server je licenzován na počet současně (právě) připojených uživatelů. To není hůře než v jiných schématech licenzování. Když ale uživatel používá méně licencí než je počítačů, keré se skutečně mohou do databáze připojit vzniká často otázka, jak rozeznat a případně odpojit neaktivní uživatele, aby se mohl připojit další, který to nutně potřebuje. V průběhu let je používáno několik řešení této úlohy.

Jedno je neustálá kontrola aktivity uživatelů a pokud uživatel neprovede akci vstup/výstup a neobrátí se na databázi po určitou dobu, je odpojen. To samozřejmě způsobuje občas problém, pokud uživatel kontroluje něco z obrazovky, případně i píše delší text, neobrátí na databázi po několik minut a přesto nechce být odpojen.

Jiné řešení je zapnout Event manager v určitých místech a zde kontrolovat každý vstup z klávesnice a každé klepnutí myši na lokální stanici. To samozřejmě zatěžuje procesor, protože než znak přejde do okna na popředí musí nejdřív projít specielním procesem.

Zádné z těchto řešení není perfektní. Musíte vždy vyřešit frekvenci kontroly s úvahou na zpomalení počítače. Následující kód je kompromis. Neodpojí uživarlele, kerý byl nějak aktivní a používal počítač během doby kontroly. Předpokládejme, že chceme odpojit uživatele po 60 minutách nečinnosti. Následující příklad kontroluje uživatele každých 20 minut (po 1/3 požadovaného času) a pak spustí čítač. Pokud uživatel během spuštění čítače provede vstup z klávesnice je čítač vypnut a kontrola bude za dalších 20 minut. Jestliže se vstup z klávesnice nedostaví ani po 60 minutách je uživatel odpojen.


Kontrola odpojení
Spát 1/3 času
Vzbudit/spustit
Zachytávání událostí
Vypnout
Zachytávání událostí
Spát do
vzbuzení
Je
uživatel
přítomen
Y
N
Zobrazit
zprávu
Spát krátce
Je
uživatel
přítomen
Y
N
Provést
odpojení
Zachytávání událostí
Nastavit přepínač
Uživatel přítomen True
Vzbudit
Kontrola odpojení

22.1. Ošetřování událostí

ON EVENT CALL(MetodaUdálosti;{NázevProcesu})

Tento příkaz instaluje proceduru s názvem předaným v parametru MetodaUdálosti jako proceduru pro ovládání událostí jako oddělený proces nazvaný parametrem NázevProcesu. Je nejlepší spustit tento proces jako lokální proces s názvem začínajícím znakem $ jako prvním znakem názvu. Po spuštění tohoto procesu budou všechny klepnutí myší a na klávesnici zachytávány tímto procesem a pak předávány do procesu s oknem na popředí.

Pro určité konfigurace počítače (CPU, paměť) může tímto procesem dojít k podstatnému zpomalení počítače, tak že se aplikace nedá téměř použít.

Proto je nutno tento proces vypnout jak to nejdříve jde pomocí příkazu ON EVENT CALL ("").

Upozornění: Neumisťujte příkaz TRACE do metody zachytávání událostí. Tímto způsobem by jste vytvořili nekonečnou smyčku, kde každá událost vyvolá opět okno ladění, jež z tohoto důvodu pak nelze ukončit.

22.1.1. Odpojení neaktivních uživatelů

1. Vytvořte novou metodu procesu OnCallQuit následovně, nebo importujte text:

```
If (False)
      ` Metoda: OnCallQuit
      ` ACI Univerzita kurzy programování
      ` Generic ACI Shell Programming
      ` Vytvořeno: Kent Wilbur
      ` Datum: 1/17/97

      ` Účel: Kontroluje přítomné uživatele a nastaví flag, že je

   <>fGeneric := False
   <>f_Version6x20 :=True
   <>fK_Wilbur :=True

End if

<>fUserPresent := True  `uživatel pracuje
DELAY PROCESS (<>LLogOffPid; 0)  `resum proces

   ` Konec metody
```

2. Vytvořte novou metodu procesu P_LogoffChecker následovně, nebo importujte text:

```
If (False)
      ` Metoda: P_LogoffChecker
      ` ACI Univerzita kurzy programování
      ` Generic ACI Shell Programming
      ` Vytvořeno: Kent Wilbur
      ` Datum: 17/1/97

      ` Účel: Ovládá proces odpojení uživatele

   <>fGeneric := False
   <>f_Version6x20 :=True
   <>fK_Wilbur :=True

End if

      `Vytvoření místních proměnnýách
C_LONGINT($LCheckTime;)              ` čas k testování uživatele
C_LONGINT($LQuitDelay)               ` čas do dalšího testu
C_LONGINT($LWindowID)                ` ID okna
C_BOOLEAN($fUserLogoff)              ` Flat k vypnutí uživatele ze systému
C_STRING(255;$sMessage)              ` řetězec zprávy

MENU BAR (1)

<>LLogOffPid := Current process
<>fUserPresent := False
$fUserLogoff := False
$LCheckTime := 60 * 60 * 60  `Minutes * Seconds * Ticks
$LQuitDelay := $LCheckTime/3

Repeat

   PROCESS _MyDelay(<>LLogOffPid; $LQuitDelay)  ` pozdržet proces

   If (<>fUserPresent)                  ` Existuje uživatel udávající časovač vypnutí a reset
      ON EVENT CALL ("")
      <>fUserPresent := False
      $fUserLogoff := False
      $LQuitDelay := $LCheckTime/3`Divide time by three then check

   Else

      If ($fUserLogoff)                 ` uživatel stále přihlášen po vnitřní časové době
         BEEP
         BEEP
         BEEP
         BRING TO FRONT (Current process)  `Upozornit uživatele, že se bude vypínat
```

```
$LWindowID:=WIN_LNewWindow (300; 150;;<>WIN_LCentered;<>WIN_LModal)
GOTO XY (0; 0)
$sMsg := "POZOR : POZOR : POZOR" + (Char(Carriage return) * 2) +
         "Budete odpojen za 1 minutu!" + (Char(Carriage return) * 2) +
         "Pokud neklepnete do tohot okna."
MESSAGE ($sMsg)
PROCESS _MyDelay (Current process; 3600)  ` Dát uživateli šanci že je stále zde
CLOSE WINDOW

If (<>fUserPresent)                        ` Uživatel se vrátil
   $LQuitDelay := 0                        ` nastavit zdržení na nula
Else
   ON EVENT CALL ("")                      ` vypnou zachytávání událostí
   M_GEN_Quit
End if

Else

   $LQuitDelay := $LCheckTime
   $fUserLogoff := True                    ` Else, $fUserLogoff is set to True
   ON EVENT CALL ("OnCallQuit")  ` vypnout časovač

End if `$fUserLogoff

End if `<>fUserPresent

Until (False)
   ` Konec metody
```

3. Upravte metodu nazvanou P_LogoffChecker následovně.

```
$LCheckTime := 15 * 60  `Minut * Sekund * Tiků
```

4. Do metody projektu MyStartup přidejte následující kód.

```
      ` Tato část řídí odpojení uživatele, který již nepracuje
$Lpid := PROCESS_LSpawnProcess ("P_LogoffChecker"; 32; "$Logoff"; False; False)
```

5. Upravte metodu projektu P_IMPEXP_ExportData následovně:

```
…
C_LONGINT ($Lpid)` ID procesu Logoff testu
…
   $LWindowID :=WIN_LNewWindow (200; 30; <>WIN_LCentered;
<>WIN_LStandardNoResize; "Exporting")
   If ($fBackground )
      PROCESS_UpdateWindowArray (""; Current process;True)
   End if
   $Lpid := Process number ("$Logoff")       ` je zde test logoff
```

```
    If ($Lpid > 0)
      ON EVENT CALL ("") `vypnout zachytávání událostí
      PAUSE PROCESS($Lpid )
    End if

    .... Dále v kódu ....

    If ($fBackground )
        PROCESS_UpdateWindowArray (""; Current process;True)
      <>LBackground := <>LBackground -1
    End if
    If (($Lpid > 0) & (<>LBackground=0))
      <>fUserPresent := True                    ` nastavit pokud je uživatel zde
      RESUME PROCESS ($Lpid )
    End if
    …
```

6. Upravte metodu projektu P_IMPEXP_ImportDatanásledovně:

```
    …
    C_LONGINT ($Lpid)` ID procesu Logoff testu
    …
      $LWindowID :=WIN_LNewWindow (200; 30; <>WIN_LCentered;
    <>WIN_LStandardNoResize; "Importing")
      If ($fBackground )
        PROCESS_UpdateWindowArray (""; Current process;True)
      End if
      $Lpid := Process number ("$Logoff")       ` je zde test logoff
      If ($Lpid > 0)
        ON EVENT CALL ("") ` vypnout zachytávání událostí
        PAUSE PROCESS($Lpid )
      End if

    .... Dále v kódu ....

    If ($fBackground )
        PROCESS_UpdateWindowArray (""; Current process;True)
      <>LBackground := <>LBackground -1
    Else                                        ` běží v nějakém proceu obnoví okno
      WIN_OutputWindowTitle
    End if
    If (($Lpid > 0) & (<>LBackground=0))
      <>fUserPresent := True                    ` nastavit pokud je uživatel zde
      RESUME PROCESS ($Lpid )
    End if
    …
```

7. Restartujte databázi a otestujte svůj kód pro odpojení.

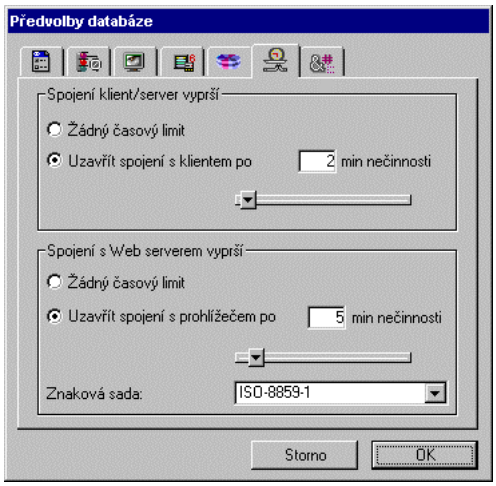
8. Obnovte metodu projektu P_LogoffChecker následovně:

```
$LCheckTime := 60 * 60 * 60  `Minutes * Seconds * Ticks
```

9. Ponechte databázi zapnutou dokud se sama nevypne a pak ji znovu spusťte.

22.2. Ztracení uživatelé

V případě, že se počítač uživatele odpojí ze sítě dříve než se odpojí regulérně ze 4D Server je pro 4D Server nezbytné vymazat paměť zabranou tímto uživatelem, odemknout záznamy, vymazat semafory, ukončit transakce atd. Jak bude dlouho server čekat na regulérní odpojení uživatele je řízeno parametrem nastaveným v předvolbách databáze. Poznámka: tento mechanismus pracuje pouze pro klienty připojené přes ADSP a IPX. Protokol TCP/IP je protokol založený na transakcích a není zde udržováno neustálé spojení. Ztráta klienta připojeného přes TCP/IP vyžaduje, aby administrátor fyzicky přišel k počítači serveru a manuálně odstranil klienta ze seznamu uživatelů (fyzická přítomnost administrátora není potřeba pokud se použijí sofwary na vzdálené řízení počítače jako např. Timbuktu).



23. Předvedení 4D Server

23.1. Demonstrace použití vyrovnávací paměti aplikace

Počkejte dokud váš instruktor nespustí na stroji serveru aplikaci ACI Video a mezitím zkontrolujte svůj počítač

1. V ovládácím panelu Síť zkontrolujte, že je váš počítač pojmenován.
2. Zkontrolujte, že nemáte zapnuto mnoho dalších aplikací a případně je ukončete.
3. Ve složce systému (Windows) poklepejte na složku ACI. Zkontrolujte, že neobsahuej žádné soubory s příponou .rex a .res a složky s názvem databáze.
4. Spusťte 4D Client a připojte se k dataábzi ACI Video. Změřte čas, který zabere spuštění aplikace, ale nic nedělejte. Uvidíte upozornění pro přenos zdrojů na váš počítač.
5. Počkejte dokud se obrazovka neustálí v jednom stavu a nebudou obnoveny údaje o dnešních prodejích. Změřte čas pro následující operace
6. Spusťte proces Zákazníci, změřte čas prvního spuštění a zaznamenejte jej.
7. Uzavřete okno Zákazníci.
8. Spusťte proces Zákazníci a změřte tento čas spuštění. Porovnejte jej s časem pro první spuštění.

23.2. Předvedení serveru pro více uživatelů

1. Pracujte podle pokynů instruktora.

23.3. Předvedení automatických hlášení v Prostředí návrháře

1. Pracujte podle pokynů instruktora.

23.4. Kontrola uzamčenosti formulářů v Prostředí návrháře

1. Pracujte podle pokynů instruktora.

23.5. Úprava formuláře zatímco jiní uživatelé tento formulář používají pro zobrazení dat

1. Pracujte podle pokynů instruktora.

24. Techniky optimalizace

24.1. Kompilace vaší databáze

Nejúčinnějším způsobem pro zlepšení chování databáze je její zkompilování pomocí kompilátoru. Některé úlohy jsou v interpretačním módu tak pomalé, že je téměř nelze používat. Po kompilaci jsou však velice rychlé.

Jestliže jste při navrhování vašeho kódu důsledně používali příkazy kompilátoru, bude kompilace databáze poměrně jednoduchá. V našem kódu jsme v konkrétních metodách netypovali proměnné procesu a meziprocessní proměnné. Typování proměnných můžete provést v libovolné metodě, kde jsou tyto proměnné použity a nebo pro urychlení kompilátoru je lze napsat do metody kompilátoru. Metoda kompilátoru je speciální metoda, která začíná názvem COMPILER a má speciální úlohu při kontrole typů proměnných a přiřazování těchto typů při běhu kompilátoru.

K vytvoření této metody si můžete pomoci samotným kompilátorem. Spusťte kompilátor a nechte jej vytvořit soubor typů. Pak otevřete tento soubor a podívejte se do jeho obsahu, které proměnné potřebují změnit typ z výchozích typů zvolených kompilátorem. Nápověda: soustřeďte se na proměnné typu TEXT a REAL.

Jestliže chcete použít tyto metody k zkrácení dalších průchodů kompilátoru musíte rovněž speciálně deklarovat všechny parametry v metodách COMPILER. Tyto deklarace jsou však odlišné od jiných deklarací kompilátorů a z důvodů syntaxe nesmí být nikdy provedeny v interpretačním módu. Ujistěte se, že jste tyto části zahrnuli do výrazu If (False) takže nebudou nikdy v interpretačním módu volány.

Protože tento proces může zabrat poměrně dlouhý čas je dále ukázán pouze výsledek tohoto procesu. Tyto metody kompilátoru můžete volat i v interpretačním módu k natypování vašich proměnných procesu a meziprocessních proměnných.

ZVLÁŠTĚ NEDĚLEJTE: Netypujte žádné vyhrazené proměnné používané samotnou 4D. Např. Document; FldDelimit; RecDelimit; C1 … Cx. nebo jiné další proměnné uvedené ve zdrojích STR# 128 a STR# 129. Ujistěte se, že jste do vašich příkazů kompilátorů nezahrnuli žádnou proměnnou používanou v externích zásuvných modulech. Tyto proměnné mohou být nalezeny ve zdrojích VAR# nebo VAR<> uvnitř modulu.

Poznámka: Nastavení kompilátoru Optimized nemá již takový význam jako dříve pokud neprovádíte složité matematické výpočty na strojích s procesorem 680x0. Při zapnutí této možnosti dostaneme poměrně malé výhody, ale zabere delší čas pro kompilaci a při běhu databáze může někdy vyvolat neočekávaná ukončení běhu databáze.

Nastavení kompilátoru: Správným nastavením můžete zlepšit chování kompilátoru a výsledné aplikace. Ujistěte se, že tlačítka jsou typována jako LongInteger. Nastavte výchozí numerické typy na LongInteger. Tuto změnu však proveďte po úvaze, protože může způsobit problém.

Příklad problému:

```
C_REAL(r1;r2; r3)
r1 := 1.5
r2 := 2
r3 := MyMethod(r1; r2)


` MyMethod
$1 := $0 + 1
$0 := $0 * $2
```

Najdete dva problémy obsažené v tomto kódu?

24.1.1. Vytvoření metody COMPILER

1. Vytvořte novou metodu projektu COMPILER a importujte text pomocí textového editoru:

```
If (False)
    ` Metoda: COMPILER
    ` ACI Univerzita kurzy programování
    ` Generic ACI Shell Programming
    ` Vytvořeno: Jim Steinman
    ` Datum: 16/1/97

    ` Účel: Místo pro psaní proměnných pro kompilátor.

  <>fGeneric :=True
  <>f_Version6x20 :=True
  <>fJ_Steinman :=True

End if

ARRAY INTEGER(<>PROCESS_aiViewNumber;0)
ARRAY INTEGER(<>PROCESS_aiWindows;0)
ARRAY INTEGER(<>aiImportExport;0)
ARRAY STRING(31;<>PROCESS_asViewName;0)
ARRAY STRING(35;<>PROCESS_asWindowName;0)

C_BOOLEAN(<>fGeneric;<>fJ_Steinman;<>fK_Wilbur)
C_BOOLEAN(<>fQuit;<>fUserPresent)
C_BOOLEAN(<>f_Version6X00;<>f_Version6X10;<>f_Version6X20)
C_BOOLEAN(<>WIN_fIsWindows)

C_LONGINT(<>LBackground)
C_LONGINT(<>LLogOffPid)
C_LONGINT(<>LMainProcess)
```

```
C_LONGINT(<>LNotifyPid)
C_LONGINT(<>LSalesPid)
C_LONGINT(<>LWindowPaletteUpdater)
C_LONGINT(<>PROCESS_LTablePalettePid)
C_LONGINT(<>WIN_DesignerSize)
C_LONGINT(<>WIN_DesignerSizeCascade)
C_LONGINT(<>WIN_L1Pixel)
C_LONGINT(<>WIN_LCascade)
C_LONGINT(<>WIN_LCentered)
C_LONGINT(<>WIN_LDeskAccessory)
C_LONGINT(<>WIN_LFillScreen)
C_LONGINT(<>WIN_LFloatingPalette)
C_LONGINT(<>WIN_LFloatingScrollCoefficient)
C_LONGINT(<>WIN_LFloatingTitleCoefficient)
C_LONGINT(<>WIN_LFloatingToggelableCoef)
C_LONGINT(<>WIN_LFloatingZoomCoefficient)
C_LONGINT(<>WIN_LLower)
C_LONGINT(<>WIN_LLowerLeft)
C_LONGINT(<>WIN_LLowerRight)
C_LONGINT(<>WIN_LMenuBarHeight)
C_LONGINT(<>WIN_LModal)
C_LONGINT(<>WIN_LMovableModal)
C_LONGINT(<>WIN_LOffScreen)
C_LONGINT(<>WIN_LShadow)
C_LONGINT(<>WIN_LStandardNoResize)
C_LONGINT(<>WIN_LStandardResizable)
C_LONGINT(<>WIN_LStandardZoom)
C_LONGINT(<>WIN_LToolBarHeight)
C_LONGINT(<>WIN_LUpper)
C_LONGINT(<>WIN_LUpperLeft)
C_LONGINT(<>WIN_LUpperRight)
C_LONGINT(<>WIN_LWindowHorizonOpen)
C_LONGINT(<>WIN_LWindowVerticalOpen)

C_REAL(<>WIN_rResize)
C_STRING(2;<>sComma;<>sQu;<>sSP)
C_TIME(<>hTimeDifference)

ARRAY INTEGER(aiFields;0)
ARRAY INTEGER(aiImportExport;0)
ARRAY LONGINT(aLFieldTypes;0)
ARRAY POINTER(apBox;0)
ARRAY REAL(arAmount;0)
ARRAY REAL(arTotalSales;0)
ARRAY STRING(15;asCategory;0)
ARRAY STRING(25;asCompany;0)
ARRAY STRING(31;asFields;0)
ARRAY STRING(31;asImportExportFields;0)
ARRAY STRING(31;hlCustomerTabs;0)
ARRAY STRING(40;asReports;0)
```

```
C_BOOLEAN(fIncludeInvisible)
C_BOOLEAN(fNewInvoice)
C_BOOLEAN(fOnlyIndexed)
C_BOOLEAN(fProcessAvailable)

C_DATE(dFirstOfMonth;dFrom;dLastOfMonth;dTo)

C_LONGINT(bAdd;bAppend;bCalc;bCalcPtr;bCalculate;bCancel;bChangePrices)
C_LONGINT(bChart;bClear;bCopyAll;bCustomers)
C_LONGINT(bDailySales;bDelete;bDeleteItem;bDeleteSelected;bDone;bDownArrow)
C_LONGINT(bEscape;bExport;bFirst;bHide;bInsert;bInvoices;bLabel;bLast;bLocal)
C_LONGINT(bModify;bMyGrid;bNewRecord;bNext;bNextPage)
C_LONGINT(bOK;bOmit;bOpen;bOrderBy)
C_LONGINT(bPrevious;bPreviousPage;bPrint;bProducts)
C_LONGINT(bQuery;bQueryByForm;bQueryCategory)
C_LONGINT(bRemove;bReport;bReturn;bShowAll;bShowSubset;bUpArrow;bValidate)
C_LONGINT(bzCancel;bziCancel;bziDelete;bziEscape;bzEscape;bzReturn)
C_LONGINT(ck4DFormat;ckCapitalizeOff;ckCommasQuote;ckCRLF;ckForm)
C_LONGINT(ckInclude;ckOrderSelection)
C_LONGINT(LBox10;LBox11;LBox12;LBox13;LBox14;LBox15;LBox16;LBox17)
C_LONGINT(LBox18;LBox19;LBox20;LBox21;LBox22;LBox23;LBox24;LBox25)
C_LONGINT(LBox1;LBox2;LBox3;LBox4;LBox5;LBox6;LBox7;LBox8;LBox9)
C_LONGINT(LBox26;LBox27;LBox28;LBox29;LBox30;LBox31;LBox32;LBox33)
C_LONGINT(LBox34;LBox35;LBox36;LBox37)
C_LONGINT(LDaysInMonth;LEnd;LStart)
C_LONGINT(rb1;rb2;rb3;rb4;sb1;sb2;sb3;sb4)
C_LONGINT(WIN_LCentered)
C_LONGINT(WIN_LHorizonOffset)
C_LONGINT(WIN_LVerticalOffset)

C_POINTER(pTable)

C_REAL(rAmount;rSalesToday;rTotal)

C_STRING(2;sFieldDelimiter;sRecordDelimiter)
C_STRING(11;sDate)
C_STRING(15;sPage)
C_STRING(50;vRecNum)
C_STRING(60;sDlogText1;sDlogText2;sDlogText3)
C_STRING(255;sMessage)

If (False)  ` Ukazatele

   C_POINTER(GEN_Capitalize ;$1)

   C_STRING(GEN_CustomConfirm ;255;$1)

   C_DATE(GEN_dFirstofMonth ;$0)
   C_DATE(GEN_dFirstofMonth ;$1)

   C_DATE(GEN_dLastofMonth ;$0)
```

```
C_DATE(GEN_dLastofMonth ;$1)

C_POINTER(GEN_FieldsToArray ;$1)
C_POINTER(GEN_FieldsToArray ;$2)
C_POINTER(GEN_FieldsToArray ;$3)
C_BOOLEAN(GEN_FieldsToArray ;$4)
C_BOOLEAN(GEN_FieldsToArray ;$5)

C_BOOLEAN(GEN_HandleArraysDragAndDrop ;$0)
C_POINTER(GEN_HandleArraysDragAndDrop ;$1)
C_POINTER(GEN_HandleArraysDragAndDrop ;$2)
C_POINTER(GEN_HandleArraysDragAndDrop ;$3)

C_STRING(GEN_sFriendlyName ;255;$0)
C_STRING(GEN_sFriendlyName ;255;$1)

C_LONGINT(GEN_ShowTimingResults ;$1)
C_LONGINT(GEN_ShowTimingResults ;$2)

C_POINTER(GEN_TimeStamp ;$1)
C_POINTER(GEN_TimeStamp ;$2)

C_STRING(IMPEXP_ImportExportDialog ;6;$1)

C_BLOB(IMPEXP_ExportData ;$1)
C_BOOLEAN(IMPEXP_ExportData ;$2)

C_BLOB(IMPEXP_ExportDataOptimized ;$1)
C_BOOLEAN(IMPEXP_ExportDataOptimized ;$2)

C_POINTER(INITIALIZE_Process ;$1)

C_LONGINT(LNextSequence ;$0)
C_STRING(LNextSequence ;31;$1)
C_LONGINT(LNextSequence ;$2)

C_LONGINT(LOCK_LSet2ReadWrite ;$0)
C_POINTER(LOCK_LSet2ReadWrite ;$1)
C_BOOLEAN(LOCK_LSet2ReadWrite ;$2)

C_POINTER(LOCK_Set2ReadOnly ;$1)
C_LONGINT(LOCK_Set2ReadOnly ;$2)

C_POINTER(LOCK_Tables2ReadOnly ;$1)
C_POINTER(LOCK_Tables2ReadOnly ;$2)

C_POINTER(LOCK_Tables2ReadWrite ;$1)
C_POINTER(LOCK_Tables2ReadWrite ;$2)

C_BOOLEAN(PROCESS_fCheckProcessAvailable ;$0)
C_LONGINT(PROCESS_fCheckProcessAvailable ;$1)
```

```
   C_LONGINT(PROCESS_LNextView ;$0)
   C_STRING(PROCESS_LNextView ;15;$1)

   C_LONGINT(PROCESS_LSpawnProcess ;$0)
   C_STRING(PROCESS_LSpawnProcess ;31;$1)
   C_LONGINT(PROCESS_LSpawnProcess ;$2)
   C_STRING(PROCESS_LSpawnProcess ;31;$3)
   C_BOOLEAN(PROCESS_LSpawnProcess ;$4)
   C_BOOLEAN(PROCESS_LSpawnProcess ;$5)
   C_BOOLEAN(PROCESS_LSpawnProcess ;$6)

   C_LONGINT(PROCESS_MyDelay ;$1)
   C_LONGINT(PROCESS_MyDelay ;$2)

   C_STRING(PROCESS_UpdateWindowArray ;31;$1)
   C_LONGINT(PROCESS_UpdateWindowArray ;$2)
   C_BOOLEAN(PROCESS_UpdateWindowArray ;$3)

   C_BLOB(P_IMPEXP_ExportData ;$1)
   C_BOOLEAN(P_IMPEXP_ExportData ;$2)

   C_BLOB(P_IMPEXP_ImportData ;$1)
   C_BOOLEAN(P_IMPEXP_ImportData ;$2)

   C_BOOLEAN(SPE_ApplyToSelection ;$1)
   C_REAL(SPE_ApplyToSelection ;$2)

   C_BOOLEAN(SPE_ArrayToSelection ;$1)
   C_REAL(SPE_ArrayToSelection ;$2)

   C_LONGINT(WIN_LNewWindow ;$0)
   C_LONGINT(WIN_LNewWindow ;$1)
   C_LONGINT(WIN_LNewWindow ;$2)
   C_LONGINT(WIN_LNewWindow ;$3)
   C_LONGINT(WIN_LNewWindow ;$4)
   C_STRING(WIN_LNewWindow ;255;$5)
   C_STRING(WIN_LNewWindow ;31;$6)
   C_BOOLEAN(WIN_LNewWindow ;$7)

   C_LONGINT(WIN_WindowTypeOffsets ;$1)
   C_POINTER(WIN_WindowTypeOffsets ;$2)
   C_POINTER(WIN_WindowTypeOffsets ;$3)

End if
   `Konec metody
```

24.2. Pište kód, který je optimalizován pro 4D Server, kdekoliv je to možné

Když máte možnost vždy použijte příkazy optimalizované pro 4D Server, které rozdělí zpracování mezi 4D Server a 4D Client.

Následující příkazy provádějí zpracování dat na serveru. Kdekoliv je to možné použijte tyto příkazy, spíše než příkazy uváděné v sekci „Příkazy, které se provádějí na klientu“

ALL RECORDS
ARRAY TO LIST
ARRAY TO SELECTION
Average
CLEAR NAMED SELECTION
CLEAR SET
COPY NAMED SELECTION
CREATE EMPTY SET
CREATE SET
CUT NAMED SELECTION
DELETE SELECTION
DIFFERENCE
DISPLAY RECORD
DISPLAY SELECTION
DISTINCT VALUES
INTERSECTION
RELATE ONE SELECTION
LIST TO ARRAY
Max
Min
MODIFY SELECTION
RELATE MANY SELECTION
REDUCE SELECTION
SCAN INDEX
QUERY
QUERY BY EXAMPLE
QUERY SELECTION
SELECTION TO ARRAY
ORDER BY
Std deviation
Sum
Sum squares
UNION
USE NAMED SELECTION
Variance

24.3. Vyhněte se příkazům, které se provádějí pouze na klientu

Příkazy pro následující akce se neprovádějí na serveru takže všechny záznamy musí projít sítí z klienta na server, aby mohly být zpracovány na klientu.

Export

Export a import se nevykonává na počítači serveru. Představte si jaký by musel být výkon serveru, kdyby musel přistupovat k disku při velkých exportech či importech a prováděl více exportů a importů pro několik klientů. Tyto příkazy přenesou záznamy sítí jeden záznam po druhém.

EXPORT DIF, TEXT, SYLK
IMPORT DIF, TEXT, SYLK
SEND RECORD
RECEIVE RECORD

Tisky

Tisky a s nimi případně svázaný export se nevykonávají na počítači serveru, ale na počítači klienta. Jsou to následující příkazy:

PRINT LABEL
PRINT FORM
PRINT SELECTION
REPORT

Vytváření diagramů vyžaduje předběžné zpracování. Toto zpracování se vykonává na počítači klienta. Proto i následující příkaz přenáší záznamy přes síť jeden záznam po druhém.

GRAPH TABLE

Operace založené na výrazech

Protože výraz je jeden či více řádků kódu jsou operace založené na výrazech či používající výrazy prováděny na počítači klienta a přenášejí ke zpracování záznamy přes síť. Tyto operace nevyužívají výhod architektury klient/server. V době zpracování záznamu přenášejí přes síť celý záznam a operace se vykoná na klientu. Pro uložení putuje záznam zpět na server.

APPLY TO SELECTION
QUERY BY FORMULA
QUERY SELECTION BY FORMULA
ORDER BY FORMULA

Použití příkazu APPLY FORMULA, QUERY BY FORMULA na klientu připojeném k serveru je velice pomalé. Při neopatrném použití a u velkých databází může vyplnění tohoto příkazu zabrat třeba i celou noc. Tyto příkazy vyvolávají veliký pohyb dat po síti a nezpomalí proto pouze provádění úlohy a počítač jednoho uživatele, ale rovněž všechny ostatní uživatele databáze a případně sítě.

Jestliže je často potřebný výraz jehož výsledkem je konstanta bude mnohem lepší databáze v denormalizovaném tvaru, která tuto hodnotu uloží přímo v databázi kdykoliv je potřeba.

Jiným návrhem na řešení může být nahradit příkaz QUERY BY FORMULA řadou příkazů query ve vztažených tabulkách a pak použít příkaz RELATE MANY SELECTION nebo RELATE ONE SELECTION k nalezení patřičných záznamů.

Pokud nelze použití výrazů na velké databáze vyloučit je dalším způsobem řešení použít části kódu s výrazy v samostatných metodách, které budou spouštěny jako uložené procedury na serveru a klient využije výsledek.

24.4. Nahrazení příkazů, které se provádějí pouze na klientu jinými příkazy

Import a export záznamů

K přenosu dat ze serveru či na server použijte raději příkazy SELECTION TO ARRAY a ARRAY TO SELECTION místo příkazů přistupujících pouze k jednomu záznamu.

Místo APPLY TO SELECTION použijte SELECTION TO ARRAY, pak smyčku For a potom ARRAY TO SELECTION (je to 14x rychlejší). Jeden z účastníků tohoto kurzu zrychlil svůj kód s použitím této techniky v jedné operaci z 90 minut na 20 sekund.

24.4.1. Náhrada APPLY TO SELECTION

1. Vytvořte novou metodu projektu ApplyToSelection a importujte z textového editoru:

```
If (False)
   ` Metoda: ApplyToSelection
   ` ACI Univerzita kurzy programování
   ` Vytvořeno: Jim Steinman
   ` Datum: 16/1/97

   ` Účel: Demonstruje princip použití array místo
   ` APPLY TO SELECTION prostředí více uživatelů na serveru.
```

```
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

End if

      ` Vytvoření místních proměnných
C_LONGINT ($i)                          ` opakovací čítač
C_LONGINT ($LWindowID )                 ` ID okna
C_BOOLEAN ($fOriginallyReadOnly) ` Originální stav z tabulky

If (Records in selection ([Produkty])=0)
   ALERT ("Musíte mít záznamy ve vašem platném výběru")
Else

   $LWindowID :=WIN_LNewWindow (200; 160; <>WIN_LCentered)
   DIALOG ([zDialogy]; "ApplyToSelection")
   CLOSE WINDOW

   If (OK=1)
      $LWindowID :=WIN_LNewWindow (220; 90; <>WIN_LCentered)
      GOTO XY(10; 3)
      MESSAGE ("Načítám ceny")

      SELECTION TO ARRAY ([Produkty]Cena;arAmount)    ` Vzít všechny ceny najednou

      $fOriginallyReadOnly := Read only state ([Produkty]) ` Ujistěte se, že je tabulka
odemčená
      LOCK_Tables2ReadWrite (->[Produkty])

      For ($i; 1; Size of array (arAmount))                  ` Opakovat přes změnu cen
         If (rb1=1)
            arAmount{$i} := arAmount{$i} + rAmount
         Else
            arAmount{$i} := Round (arAmount{$i} + arAmount{$i} * rAmount/100; 2)
         End if
      End for

      GOTO XY (10; 3)
      MESSAGE ("Ukládám změny")

      START TRANSACTION                                  ` Všechny změny nebo žádné

      ARRAY TO SELECTION (arAmount; [Produkty]Cena) ` Uložit všechny změny
najednou

      CLOSE WINDOW

      If (Records in set ("LockedSet")=0)
         VALIDATE TRANSACTION
      Else
```

```
      CANCEL TRANSACTION
      ALERT ("Některé záznamy byly používány, změny se neuložily!")
    End if


    UNLOAD RECORD ([Produkty])
    If ($fOriginallyReadOnly)                    ` Ujistěte se, že pokud je třeba tabulka je
opět zamčená
      LOCK_Tables2ReadOnly(->[Produkty])
    End if

  End if
End if

  ` Konec metody
```

2. Otestujte váš kód na několika záznamech v Prostředí uživatele pomocí dialogu Provést metodu.

3. Nahraďte obsah metody PostFaktury následujícím textem:

```
If (False)
    ` Metoda: PostFaktury
    ` ACI Univerzita kurzy programování
    ` Vytvořeno: Jim Steinman
    ` Datum: 16/1/97

    ` Účel: Používá metodu SumFaktury na všechny
    ` záznamy [Zákazníci] v platném výběru

  <>f_Version6x10:=True
  <>f_Version6x20:=True
  <>fJ_Steinman:=True

    ` Upraveno: 17/1/97 Nahrazeno  APPLY TO SELECTION
    ` k omezení zatížení sítě přenosem všech záznamů zákazníků
    ` přes síť
  <>fK_Wilbur:=True

End if

 ` Vytvoření místních proměnných
C_LONGINT($i)  ` opakovací čítač
C_LONGINT($LWriteStatEe)  ` Read/Write stav tabulk yZákazníci
C_LONGINT($LProcessID)  ` ID Procesu
C_LONGINT($LNumberOfRecordsInSet)  ` Počet záznamů v zamčenémvýběru
C_STRING(40;$sUserName)  ` Název uživatele zamčeného záznamu
C_STRING(40;$sMachine)  ` Název stroje zamčeného záznamu
C_STRING(40;$sProcessName)  ` Název procesu zamčeného záznamu
```

```
CONFIRM("Sečíst faktury pro každého zákazníka? (Nelze zpět!)")

If (OK=1)
   COPY NAMED SELECTION([Zákazníci];"OriginalSelection")

$LWriteState:=LOCK_LSet2ReadWrite (->[Zákazníci];False)

   Repeat
      ARRAY REAL (arTotSales;Records in selection([Zákazníci]))
      ARRAY STRING(25;asCompany;0)

      SELECTION TO ARRAY ([Zákazníci]IDZákazníka;asCompany)  ` ponese přes síť
pouze název zákazníka

      For ($i;1;Size of array(arTotSales))
         QUERY([Faktury];[Faktury]IDZákazníka=asCompany{$i})` # záznamů a první
záznam půjdou přes síť
         arTotSales{$i}:=Sum([Faktury]FakturaCelkem)      ` Pouze součet půjde pře síť
      End for

      ARRAY TO SELECTION (arTotSales;[Zákazníci]ProdejCelkem)

      $LNumberOfRecordsInSet:=Records in set("LockedSet")


      If ($LNumberOfRecordsInSet>0)
         USE SET("LockedSet")
         CREATE EMPTY SET(pTable->;"NonDeletedSet")
         For ($i;1;$LNumberOfRecordsInSet)
            LOCKED ATTRIBUTES(pTable-
>;$LProcessID;$sUserName;$sMachine;$sProcessName)
            If ($LProcessID#-1)                    ` Záznam je tam stále
               ADD TO SET (pTable->;"NonDeletedSet")
            End if
            NEXT RECORD(pTable->)
         End for
         USE SET("NonDeletedSet")
         $LNumberOfRecordsInSet:=Records in set("NonDeletedSet")
         CLEAR SET("NonDeletedSet")
      End if

   Until ($LNumberOfRecordsInSet=0)

   LOCK_Set2ReadOnly (->[Zákazníci];$LWriteState)

   USE NAMED SELECTION("OriginalSelection")
   CLEAR NAMED SELECTION("OriginalSelection")

End if
      `Konec metody
```

4. Otestujte kód v Prostředí uživatele pomocí dialogu provést metodu.

24.5. Nezatěžujte zbytečně server

Server je stroj, který i za běžné situace vykonává mnoho rozličných úloh. Aby jste server zbytečně nepřetěžovali, vyhněte se psaní metod následujícím způsobem:

```
QUERY([MyFile])
FIRST RECORD([MyFile])
LOAD RECORD([MyFile])
```

První dva řádky byly obzvlášť populární ve starších programech 4D pro určité nekonzistentnosti v chování některých příkazů ve starších verzích 4D. Od verze 3 se však všechny příkazy chovají stejně. To znamená příkaz QUERY provede dotaz, přejde na první nalezený záznam a zavede jej do paměti. Příkaz FIRST RECORD přejde na první záznam výběru a zavede jej do paměti. Samozřejmě příkaz LOAD RECORD zavede opět záznam do paměti. Výsledkem jsou dvě vyvedení a tři zavedení téhož záznamu předtím než počítač klienta provedl cokoliv se záznamem. Kromě toho tím vzniká významné zatížení sítě.

Jiný běžný způsob psaní kódu je:

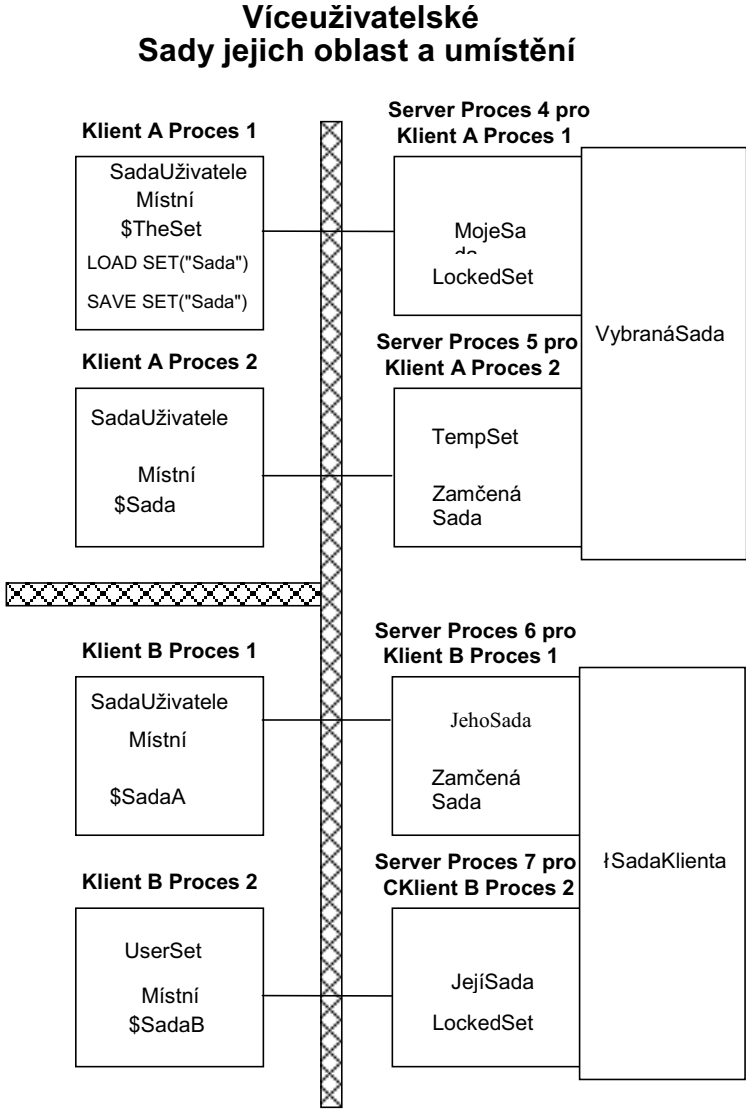
```
While(Locked([MyFile]))
   UNLOAD RECORD([MyFile])
   DELAY PROCESS(Current process;20)
   LOAD RECORD([MyFile])
End while
```

Příkaz LOAD RECORD vyvede a znovu zavede libovolný platný záznam. Příkaz UNLOAD RECORD není potřeba. Kromě toho jestliže je kód prováděn v Prostředí uživatele aplikace nemá příkaz DELAY PROCESS žádný efekt. Místo toho je potřeba použít metodu obdobnou PROCESS_MyDelay uváděnou v tomto kurzu.

Viz také kapitolu Příkazy a metody, které vytvářejí speciální funkce pro 4D Server

24.6. Používejte správně sady

Od v6 jsou sady ukládány na server. Sady mohou někdy být dosti velké a zabírat podstatnou část paměti. Jestliže se chystáte použít několik velkých sad pro příkazy INTERSECTION, UNION či DIFFERENCE pro účely tisku dlouhých zpráv a tyto sady budou používány v době vytváření zprávy a užívány stále znovu a znovu v různých kombinacích. Může tento způsob použití vyvolat značný pohyb po síti a hlavně zatížit server. V tomto případě bude lepší vytvořit tyto sady jako místní „"$Sada“ a tímto přemístit tyto sady na počítač klienta, kde budou zabírat paměť klienta a zatěžovat pouze tohoto jednoho klienta. Musíte vždy uvažovat tak co by se stalo pokud bude daná úloha prováděna pro více klientů a jak v tuto dobu zatíží paměť serveru.



24.7. Nepřetěžujte provádění kompilovaného kódu

Provádění kompilovaného kódu se řídí velmi přísnými pravidly. Samozřejmě všechny proměnné musí mít předem známý typ. Určité techniky programování však způsobují značné problémy při provádění kódu. Odkazy mimo rámec definovaných položek mohou způsobit zhroucení kompilovaného kódu. Změny počtu typu parametrů způsobují značnou zátěž při provádění kompilovaného kódu. Lokální proměnné žijí v paměti stack. Jestliže váš kód znovu přiřadí hodnotu do lokální proměnné musí vytvořit proměnnou za běhu. S některými úlohami si provádění kompilovaného kódu neporadí. Proto je nutno se přepnout do interpretačního módu pro ošetření těchto proměnných. Totéž se může stát,

když používate Get pointer ("MyVar"+String($i)) a proměnná MyVarx neexistuje v deklaracích kompilátoru. Oba tyto způsoby pracují, ale oba značně sníží rychlost vaší kompilované databáze.

24.8. Řízení dotazování

- Omezte sekvenční dotazy a třídění
- Omezte třídění podle více než jednoho pole
- Omezte dotazy typu Obsahuje nebo Končí na
- Vytvořte váš vlastní dialog pro sestavování dotazů pro řízení způsobu dotazování uživatele
- Jako vývojář můžete vybrat nejoptimálnější způsob dotazování (zabránít sekvenčním dotazům, dotazům přes relace)
- Ve vašem vlastním dialogu dotazů můžete použít tlačítko, které otevře Editor dotazů 4D.
- Tato druhá úroveň dotazů umožní pokročilým uživatelům vytvářet složitější dotazy, které jste již nenavrhli.
- Nedotazujte se přes relace
- Dotazy na základě vztažených tabulek provádějte přímo ve vztažené tabulce a použijte RELATE ONE SELECTION nebo RELATE MANY SELECTION. Tyto příkazy jsou sice sekvenční, ale jsou prováděny na serveru. Kombinace hledání ve vztažené tabulce a těchto příkazů podstatně urychlí výsledek hledání.

24.9. Řízení třídění

- Vytvořte vlastní dialogové okno pro třídění, ve kterém se vyhnete možnosti sekvenčního třídění a třídění založeném na výrazech.

24.10. Prázdná první tabulka

Když se 4D spouští jednou z prvních věcí, kterou vykoná je zavedení tabulek adres první tabulky do paměti. Jestliže se do paměti zavádí tabulka adres tabulky, která neobsahuje žádný záznam, tato operace se vykoná téměř okamžitě. Rozdíl ve využití paměti u prázdné tabulky a tabulkou o jednom záznamu je 32K. Jestliže se zavádí do paměti tabulka adres o tisíci záznamech zabere to určitý čas. První tabulka bez záznamů může být použita k ukládání speciálních formulářů, dialogů a pro dočasné vytváření záznamů, které nejsou nikdy uloženy.

Struktura tabulek adres byla u 4D v6 značně změněna. Minimální velikost je však stále 32K. Na druhou stranu je třeba vědět, že velikost přeskočí z 32K na 96K mezi 4096 a 4097 uloženými záznamy. Velikost tabulky adres se zvyšuje o 32K pro každý dodatečný blok 4096 záznamů nad prvními 8192 záznamy.

24.11. Použití automatických relací a výstupní formuláře

Výstupní formuláře jsou optimalizovány pro 4D Server

> 4D Server posílá pouze pole, která jsou ve formuláři umístěna a rovněž pole používaná v metodách objektů a posílá pouze ta data, která se vejdou do patřičné obrazovky.

Ujistěte se, že do formuláře umisťujete pouze ta pole, která potřebuje uživatel nezbytně vidět na výstupním formuláři. Formulář se 4 poli se zobrazí rychleji než formulář s 20 poli. Čím méně polí na formuláři tím menší provoz na síti a tím rychlejší překreslování obrazovky. Pole, která jsou ve výstupním formuláři, ale jsou dostupná pohybem posuvníku jsou přenášena i když nejsou vidět a zatěžují tak síť.

Velikost okna je důležitým faktorem

> Okno, které zobrazuje 5 záznamů bude vyvolávat menší pohyb na síti a zobrazovat se rychleji než totéž okno s formuláře, který zobrazuje 20 záznamů.

24.12. Vstupní formuláře

Čím více dat formulář používá (grafika, barvy, text) tím více paměti je potřeba k uložení formuláře a tím delší dobu trvá zavedení formuláře.

Počet objektů ve formuláři je rovněž důležitým faktorem. Čím více objektů na formuláři tím delší čas trvá vykreslení formuláře, protože je zde více objektů, které je potřeba vykreslit.

24.12.1. Umístěte všechny společné objekty vícestránkového formuláře na stránku 0

Od v6 je možno vytvořit hlavní stránku pro každý formulář. Libovolný objekt umístěný na této stránce formuláře se objeví a je funkční na všech stránkách daného formuláře. Protože vstupní formulář se zavádí do paměti jako celek se všemi svými objekty sníží tento způsob vytváření počet objektů a tím sníží čas k vykreslení formuláře. Zabere rovněž méně místa v paměti takže bude více místa pro další položky a pamět se bude méně obnovovat.

24.12.2. Snižte co nejvíce počet objektů na stránce formuláře

Pro dosažení dobrého výsledku při zobrazení formulářů bylo ve starších verzích 4D potřeba použít více objektů. Ve v6 je několik nových objektů, které mohou při stejném vizuálním efektu nahradit grafických objektů použitých dříve. Používejte tedy radši tyto nové objekty a snižte tak počet objektů ve formuláři.

24.12.3. Překonvertujte všechny statické objekty ve formuláři do jednoho obrázku

4D spotřebuje při zobrazení formuláře čas k výpočtu vektorů pro každý zobrazený objekt na formuláři. Jestliže máte velký počet čar, obdelníků, obrázků atd. je mnohem rychlejší překonvertovat je do jednoho obrázku. Tento jeden objekt i když je veliký se bude na obrazovce vykreslovat mnohem rychleji než mnoho oddělených objektů.

24.12.4. Rozdělte velké vícestránkové formuláře do oddělených formulářů

Všechny stránky formuláře jsou zaváděny najednou. Důležitým faktorem při úvahách je množství dat a celkový počet objektů pro všechny stránky formuláře. Čím méně dat a objektů formulář obsahuje tím rychleji se vykreslí.

Jestliže uživatel používá po většinu času pouze jednu stránku a pouze příležitostně přechází na další stránky, může být výhodnější vytvořit pro tyto další stránky oddělený formulář a zavést jej pouze v případě potřeby. Místo použití vícestránkového formuláře pak použijete pro otevření nového okna příkaz OPEN WINDOW. Navrhněte všechny formuláře tak, že se vejdou do okna téže velikosti. Použijte příkaz DIALOG k zobrazení formuláře a místo tlačítek automatických akcí přechodu na další stránky bude obsahovat tlačítko, které přijme (uzavře) tento formulář a otevře opět dialog a zobrazí další stránku. Výsledné chování bude stejné jako v případě vícestránkového formuláře pokud nebudete volat příkaz CLOSE WINDOW. Pro tento příkaz existuje jedno omezení, jestliže budete používat tlačítko ACCEPT k přechodu mezi stránkami formuláře, který zobrazuje záznam, budou data vždy uložena a aby jste se vrátili zpět případným tlačítkem Storno musíte zahájit transakci před prvním otevřením okna a podle požadované akce uživatele transakci potvrdit nebo zrušit. Nebo jestliže nechcete používat transakce musíte veškerou práci uživatele ukládat v proměnných a podle akce uživatele je na konci uložit do skutečného záznamu nebo neuložit.

24.13. Nastavte tabulky READ ONLY

Abyjste zabránili nechtěnému uzamykání záznamů a případnému čekání na odemčení záznamu, převáděte tabulky kde to lze do stavu Pouze číst.

24.14 Řízení procesu

Kdekoliv je to možné používejte lokální procesy. Čím méně procesů bude spuštěných na serveru tím méně procesů se bude dělit o čas CPU.

Nejčastěji používané procesy raději uschovávejte než, aby jste je ukončovali. Spouštění procesů zvláště globálních procesů zabírá určitý čas, ukončování procesu trvá ještě déle. Pozastavení a uschování procesu a jeho opětovné vyvolání trvá podstatně kratší dobu.

24.14.1. Úpravy LSpawnProcess k vyvolání pozastavených procesů

1. Vytvořte novou metodu projektu PROCESS_fCheckProcessAvailable následovně:

```
If (False)
      ` Metoda: PROCESS_fCheckProcessAvailable (longing) -> boolean
      ` ACI Univerzita kurzy programování
      ` Generic ACI Shell Programming
      ` Vytvořeno: Jim Steinman
      ` Datum: 16/1/97

      ` Účel: Kontrola zda není  požadovaný proces pouze pozastaven.

   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

End if

      ` Definice parametrů
C_LONGINT($1;$LProcessID)            ` ID procesu k testování
C_BOOLEAN($0;$fProcessAvailable)  ` Dostupný proces

      ` Definice místních proměnných
C_LONGINT($LState)                        ` stav procesu

   ` Přiřazení parametrů k místním proměnným
$LProcessID := $1

$LState := Process state($LProcessID)
If ($LState=Paused)
   GET PROCESS VARIABLE($LProcessID;fProcessAvailable;$fProcessAvailable)
   $0 := $fProcessAvailable
End if
 `Konec metody
```

2. Nahraďte text metody PROCESS_LSpawnProcess následujícím textem nebo importujte z textového editoru:

```
If (False)
      ` Metoda: PROCESS_LSpawnProcess (string; integer; string; bool; bool;bool) -> longint
      ` ACI Univerzita kurzy programování
      ` Generic ACI Shell Programming
      ` Vytvořeno: Jim Steinman
      ` Datum: 16/1/97

      ` Returns -> longint - process id, nula pokud nebyl proces vytvořen
      ` ---------------------------------------------------

      ` Účel: Je místo pouhého volání funkce New Process.
      ` Zabrání vytvoření procesu jestliže přepínač <>fQuit existuje.
      ` Tato metoda vytvoří nový proces řízeným způsobem.
      ` Možnost upozornit uživatele, když proces nemusí být vytvořen
      ` z důvodu nedostatku paměti. Vývojář řídí zda předaná
      ` metoda má být provedena ve více než jednom procesu (více oken).

      ` umožňuje vyhledat pozastavené procesy a vzbudit je.

   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

End if

      ` Definice prametrů a přeřazení místních proměnných
C_LONGINT($0;$Lpid)                    ` ID procesu, nula pokud nebyl vytvořen.
C_STRING(31;$1;$sMethod)               ` Metoda ke spuštění v novém procesu.
C_LONGINT($2;$LStack)                  ` Stack velikost (paměť pro procesové proměnné).
C_STRING(31;$3;$sProcessName)          ` Název vytvořeného procesu.
C_BOOLEAN($4;$fInformUser)             ` Informuje uživatele, že není dost paměti ke spuštěnbí
procesu.
C_BOOLEAN($5;$fAllowMultipleViews)     ` (Volitelné) Umožní více spuštění.
C_BOOLEAN($6;$fReclaimPausedProcess)` (Volitelné) Žádá navrácení přerušného procesu

C_LONGINT($LState)                         ` Stav procesu
C_LONGINT($i)                          ` opakovací čítač
C_LONGINT($LLocation)                      ` Pozice názvu procesu v array
C_STRING(255;$sMessage)                    ` Zpráva uživateli o nedostatku paměti.

If (Not(<>fQuit)) ` Nezačít proces pokud se vypíná databáze
   $sMethod := $1
   $LStack := $2 * 1024
   $sProcessName   := $3
   $fInformUser :=$4
   $fReclaimPausedProcess := False
   If (Count parameters>4)
      $fAllowMultipleViews := $5
      If (Count parameters>5)
         $fReclaimPausedProcess := $6
      End if
```

```
End if

$Lpid := Process number ($sProcessName)

If ($Lpid= 0) ` Již neběží žádný proces

   $Lpid := New process ($sMethod; $LStack; $sProcessName)

Else
   $0 := 0
   If ($fReclaimPausedProcess )
      If (PROCESS_fCheckProcessAvailable ($Lpid))          ` Restartovat dostupný proces
         $0 := $Lpid
      End if

      If (($fAllowMultipleViews) & ($0=0))
         $LLocation := Find in array(<>PROCESS_asViewName;$sProcessName)

         If ($LLocation>0)                              ` Máme dodatečné procesy
            $i := 1
            While (($0=0) & ($i<=<>PROCESS_aiViewNumber{$LLocation}))
               $Lpid := Process number($sProcessName+String($i))
               If (PROCESS_fCheckProcessAvailable ($Lpid)) ` Restartovat dostupný proces
                  $0 := $Lpid
               End if
               $i := $i+1
            End while
         End if                                 ` Dodatečné procesy
      End if                                    ` Více spuštění
   End if                                       ` Navrácení procesů

   If (($fAllowMultipleViews) & ($0=0)) ` Pokud je nastavena flaf více spuštění
      $Lpid := New process ($sMethod; $LStack; $sProcessName + String
(PROCESS_LNextView ($sProcessName)))
   Else
      RESUME PROCESS ($Lpid)
      SHOW PROCESS ($Lpid)
      BRING TO FRONT ($Lpid)
   End if

End if

If ($Lpid=0) ` Něco je špatně
   If ($fInformUser )
      $sMessage := "Nedostatek paměti k splnění akce. "
      $sMessage := $sMessage + "Zkustě uzavřít nějaká okna k uvolnění paměti."
      BEEP
      ALERT ($sMessage)
   End if

End if
```

```
    $0 := $Lpid

  Else
    ALERT ("Databáze se vypíná. Nelze začít nový proces.")
    $0 := 0
  End if
    ` Konec metody
```

3. Upravte metodu projektu MyStartup následovně:

```
…

INITIALIZE_Characters
<>LBackground := 0
fProcessAvailable := False

…
```

4. Přidejte následující řádek kódu na konec metody projektu INITIALIZE_Process :

```
fProcessAvailable := False
```

5. Upravte metodu projektu M_Customers následovně:

```
If (False)
    ` Metoda: M_Customers
    ` ACI Univerzita kurzy programování
    ` Vytvořeno: Jim Steinman
    ` Datum: 16/1/97

    ` Účel: Tato metoda buď vytvoří nebo přenese na popředí proces Zákazníci

  <>f_Version6x20 :=True
  <>fJ_Steinman :=True
End if

C_LONGINT ($Lpid)

$Lpid := PROCESS_LSpawnProcess ("P_Customers"; 32; "Customers"; True; True; True)

  ` Konec metody
```

6. Upravte metody projektu M_Invoices a M_Products podle metody M_Customers výše.

7. Vytvořte novou metodu projektu GEN_ClearSelection následovně:

```
If (False)
    ` Metoda: GEN_ClearSelection (pointer; {pointer{; …}}
    ` ACI Univerzita kurzy programování
```

```
      ` Generic ACI Shell Programming
      ` Vytvořeno: Jim Steinman
      ` Datum: 16/1/97
      ` Účel: Vymaže platný výběr pro určenou tabulku nebo všechny tabulky

   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

End if

C_LONGINT($i)
C_POINTER($pTable)

If(Count parameters > 0)
   For ($i; 1; Count parameters)
      REDUCE SELECTION (${$i} ->;0)      ` Použije odkaz na parametr
   End for
Else
   For ($i; 2; Count tables)
      $pTable := Table ($i)
      REDUCE SELECTION ($pTable ->;0)
   End for
End if
   ` Konec metody
```

8. Upravte metodu projektu P_Customers následovně:

```
If (False)
      ` Metoda: P_Customers
      ` ACI Univerzita kurzy programování
      ` Vytvořeno: Jim Steinman
      ` Datum: 15/1/97

      ` Účel: Zobrazí výstupní formulář se seznamem pro tabulku určenou pTable.

   <>f_Version6x10 :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

      ` Modified: 1/17/97
   <>fK_Wilbur :=True
End if
```

❖

```
Repeat

   INITIALIZE_Process (->[Zákazníci])
   LOCK_Tables2ReadWrite (pTable)

   GEN_ModifySelection
```

```
❖           GEN_ClearSelection

❖           fProcessAvailable := True
❖           PAUSE PROCESS(Current process)
❖         Until (<>fQuit)
            ` Konec metody
```

9. Upravte metodu projektu P_Products následovně:

```
If (False)
    ` Metoda: P_Products
    ` ACI Univerzita kurzy programování
    ` Vytvořeno: Jim Steinman
    ` Datum: 15/1/97

    ` Účel: Zobrazí výstupní formulář se seznamem pro tabulku určenou pTable.

  <>f_Version6x10 :=True
  <>f_Version6x20 :=True
  <>fJ_Steinman :=True

    ` Upraveno: 1/17/97
  <>fK_Wilbur :=True
End if

❖   Repeat

    INITIALIZE_Process (->[Produkty])
    LOCK_Tables2ReadWrite (pTable)

    GEN_ModifySelection

❖       GEN_ClearSelection

❖       fProcessAvailable := True
❖       PAUSE PROCESS(Current process)
❖     Until (<>fQuit)
        ` Konec metody
```

10. Upravte metodu projektu P_Invoices následovně:

```
If (False)
    ` Metoda: P_Invoices
    ` ACI Univerzita kurzy programování
    ` Vytvořeno: Jim Steinman
    ` Datum: 15/1/97

    ` Účel: Zobrazí výstupní formulář se seznamem pro tabulku určenou pTable.

  <>f_Version6x10 :=True
  <>f_Version6x20 :=True
  <>fJ_Steinman :=True

    ` Modified: 17/1/97
  <>fK_Wilbur :=True
End if

Repeat

  INITIALIZE_Process (->[Faktury ])
  LOCK_Tables2ReadWrite (pTable; ->[PoložkyFaktury)

  GEN_ModifySelection

  GEN_ClearSelection

  fProcessAvailable := True
  PAUSE PROCESS(Current process)
Until (<>fQuit)
  ` Konec metody
```

11. Restartujte databázi a otestujte pozastavování procesu.

24.14.2. Prověrka okna Seznam procesů na lokální a globální procesy

1. Přejděte do Prostředí návrháře.
2. Zvolte Návrh->Seznam procesů (průzkumník aplikace).
3. Zkontrolujte jaké číslo ID má přidělen proces Uživatel/Aplikace.
4. Podívejte se do seznamu uživatelů na serveru, jaké číslo má na serveru.
5. Všimněte, že jsou skutečně dva oddělené procesy, jeden běžící na klientu a jiný běžící na serveru (s jiným číslem) pro každý globální proces na počítači klienta.
6. Zkontrolujte jaké číslo ID má proces $TablesPalette.
7. Zkontrolujte jaké číslo ID má proces $Notifier.
8. Podívejte se do okna procesů na serveru zda tyto procesy naleznete .
9. Všimněte si, že na serveru neexistují odpovídající procesy, protože to jsou lokální procesy $TablesPalette.
10. Přejděte zpět do prostředí Vlastní nabídky a klepněte na Zákazníci nebo Faktury.

11. Přepněte se mezi okny a sledujte co se stane na paletě tlačítek.
12. Sledujte co se stane s procesem $Notifier a kontrolujte stav procesu.
13. Uschovejte okno DenníProdeje a sledujte změnu stavu procesu DailySalesDialog na serveru.

24.14.3. Nepoužívejte proces Uživatel/Aplikace

Proces Uživatel/Aplikace je nejpomalejším procesem pro provádění vašich akcí. Je to proto, že tento proces musí sdílet svůj čas se samotnou 4D. Některé z nezbytných akcí jsou vykonávány 4D v tomto procesu na pozadí. Jestliže přidělování času procesům 4D ovládá 4 procesy a všechny jsou v běhu a vykonávají tutéž úlohu každý z nich dostane stejné množství času. Protože však 4D samotná potřebuje určitý čas v procesu Uživatel/Aplikace bude tento proces nejpomalejší.

24.14.4. Pozastavení procesu Uživatel/Aplikace

Skutečně pozastavit tento proces nelze, ale existuje trik, který tomuto procesu přidělí minimální množství času.

1. Vytvořte formulář dialogu pro tabulku [zDialogy], který bude obsahovat pouze jedno tlačítko Storno.
2. Vytvořte následující metodu formuláře pro tento formulář:

```
If (False)
      ` Form Metoda: [zDialogy];"MůjDialog"
      ` ACI Univerzita kurzy programování
      ` Vytvořeno: Jim Steinman
      ` Datum: 15/1/97

      ` Účel: Zobrazit výstupní formulář protabulku zadanou ukazatelem pTable.

   <>f_Version6x10 :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

      ` Modified: 1/17/97
   <>fK_Wilbur :=True
End if

If(Form event = On Outside Call)
   CANCEL
End if
      `Konec metody
```

3. Vytvořte metodu, která provádí následující:

```
HIDE PROCESS(Current process)
```

```
Dialog([zDialogy];"MůjDialog")
SHOW PROCESS(Current process)
```

Způsob jak tento trik funguje je následující. Otevřením dialogu převedeme proces do stavu, kdy čeká na událost a procesy v tomto stavu dostávají velice málo času pro zpracování.

24.15. Ovládání výběrů

Správné ovládání výběrů vám dovolí ovládat paměť na 4D Server.

- Výběry jsou udržovány v paměti na serveru
- Výběry požadují 4 byty na záznam ve výběru, na tabulku, na proces.
- Vymazávajte pojmenované výběry z paměti, když je již nepotřebujete pomocí příkazu CLEAR NAMED SELECTION
- Omezujte počet záznamů v platném výběru, když již není více potřeba pomocí příkazu REDUCE SELECTION

24.16. Používejte array ukazatelů místo častého volání Get pointer

Opakované volání rutin 4D zpomaluje systém. Volání 4D je libovolný příkaz nebo dereferencování ukazatele. Příkaz Get pointer je nejpomalejší příkaz ze všech. Jestliže můžete omezit volání do systému můžete zvýšit rychlost svého kódu. Jestliže používáte časté volání ke získání ukazatele na skupinu objektů je mnohem rychlejší vytvořit array ukazatelů a pak jednoduše referencovat prvek array místo neustálého volání funkce get pointer.

24.16.1. Funkce zacházející s datumy

Ve 4D existuje několik funkcí, které jsou užitečné při zacházení s datumy.

- Date (datumový řetězec) -> Datum

  Tato funkce převede řetězec ve tvaru !00/00/00! do datumu.

- Day number (datum) -> Číslo

  Tato funkce navrátí číslo reprezentující den v týdnu pro všechny předané datumy.

Den	Číslo
Neděle	1
Pondělí	2
Úterý	3
Středa	4
Čtvrtek	5

Pátek	6
Sobota	7

- Month of (datum) -> Číslo

  Tato funkce navrátí číslo reprezentující měsíc v předaném datumu.

- Year of (datum) -> Číslo

  Tato funkce navrátí číslo reprezentující rok v předaném datumu.

24.16.2. Demonstrace použití array ukazatelů

1. Vytvořte novou metodu projektu GEN_dFirstofMonth následovně nebo importujte z textového editoru:

```
If (False)
      ` Metoda: GEN_dFirstofMonth (Date) -> Date
      ` ACI Univerzita kurzy programování
      ` Generic ACI Shell Programming
      ` Vytvořeno: Jim Steinman
      ` Datum: 16/1/97

      ` Legenda
      ` $1 – Datum pro které nalezneme první den v měsíci

      ` Účel: Nalezne a vrátí první den v měsíci pro předaný datum.

   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

End if

C_DATE ($0; $1)

$0 := Date ("1/" + String (Month of ($1)) +"/" + String (Year of ($1)))
      `  Konec metody
```

2. Vytvořte novou metodu projektu GEN_dLastofMonth následovně:

```
If (False)
      ` Metoda: GEN_dLastofMonth (Date) -> Date
      ` ACI Univerzita kurzy programování
      ` Generic ACI Shell Programming
      ` Vytvořeno: Jim Steinman
      ` Datum: 1/16/97

      ` Legenda
      ` $1 – Datum k nalezení posledního dne v měsíci

      ` Účel: Nalezne a vrátí poslední den v měsíci z předaného datumu.

   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

End if

C_DATE ($0; $1)
```

```
C_LONGINT ($LMonth)

$LMonth := Month of ($1)
If ($LMonth # 12)
   $0 := Date ("/1/" + String ($LMonth + 1) + "/" + String (Year of ($1)))-1
Else
   $0 := Date ("31/12/" + String (Year of ($1)))
End if
      ` Konec metody
```

3. Vytvořte novou metodu projektu M_Calendar následovně:

```
If (False)
      ` Metoda: M_Calendar
      ` ACI Univerzita kurzy programování
      ` Vytvořeno: Jim Steinman
      ` Datum: 16/1/97

      ` Účel: Zobrazí okno kalendáře

   <>fGeneric :=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

End if

C_LONGINT (LBox1; LBox2; LBox3; LBox4; LBox5; LBox6; LBox7; LBox8; LBox9;
                        LBox10; LBox11; LBox12)
C_LONGINT (LBox13; LBox14; LBox15; LBox16; LBox17; LBox18; LBox19; LBox20;
                        LBox21; LBox22; LBox23)
C_LONGINT (LBox24; LBox25; LBox26; LBox27; LBox28; LBox29; LBox30; LBox31;
                        LBox32; LBox33; LBox34)
C_LONGINT (LBox35; LBox36; LBox37)
C_LONGINT ($LWindowID)

INPUT FORM ([zDialogy]; "Calendar";*)

$LWindowID := WIN_LNewWindow (-1;-1;<>WIN_LCentered;<>WIN_LModal;
"Kalendář")        ` Otevře nové okno

DIALOG ([zDialogy]; "Calendar")        ` Otevře "Calendar Dialog" formulář jako dialog

CLOSE WINDOW
      ` Konec metody
```

4. Otevřete nabídku #1 a upravte nabídku Soubor následujícím způsobem:

```
Produkty          M_Products
-
```

Paleta tabulek	M_PROCESS_TablesPalette
Kalendář…	M_Calendar
-	
Konec	M_GEN_Quit

5. Podívejte se na tento kód pomocí 4D Server, následujte pokynů svého instruktora a zhodnoťte efektivitu tohoto příkladu.

24.17. Vyhněte se častým voláním operačního systému

Import a export dat mohou opakovaně volat systémové diskové operace. Operační systém je nejpomalejší část databázového prostředí. Pro import je ve skutečnosti rychlejší přečíst velký blok importovaného souboru a pak jej v paměti postupně rozdělit na jednotlivé části dat a převést jej do sloupců v paměti.

Při exportu slučujte data v paměti a zapisujte do souboru v několika málo krocích a velkých datových blocích. Použijte techniku $VelkýBlok:= $VelkýBlok + $tMaláProměnná

Tato technika je obvykle lepší, když ji kombinujete s kompilovanou databází a v interpretovaném módu může být pomalejší než jiné techniky.

Na počítači PowerBook 540c byly provedeny určité informační testy s našim existujícím exportem na pozadí proti optimalizovanému exporut napozadí. Průměrné čas jsou uvedeny v následující tabulce:

	Interpretovaná	Kompilovaná
Existující kód	00:16:00	00:00:42
Optimalizovaný kód	00:29:00	00:00:36

Tento příklad jasně ukazuje, že kompilovaný kód výrazně zlepší chování. Kromě toho z něho můžeme odhadnout, že určité části kódu je rozumně možné provádět pouze v kompilované databázi. Tento příklad neukazuje velké vylepšení mezi optimalizovaným a existujícím kódem v kompilované databázi, byl však použit opuze malý datový soubor s pouhými 1024 záznamy a 48K exportovaných dat. Proto za celou dobu provádění byl operační systém volán pouze dvakrát. Podstatně větší zlepšení bychom viděli při výrazném zvětšení datového souboru.

24.17.1. Optimalizace dotazů pro kompilovaný kód

1. Změňte P_IMPEXP_ExportData na IMPEXP_ExportData
2. Vytvořte novou metodu projektu IMPEXP_ExportDataOptimized následovně a nebo importujte pomocí textového editoru.

```
If (False)
    ` Metoda: IMPEXP_ExportDataOptimized (BLOB; boolean)
    ` ACI Univerzita kurzy programování
    ` Generic ACI Shell Programming
    ` Vytvořeno: Kent Wilbur
    ` Datum: 16/1/97

  <>fGeneric :=True
  <>f_Version6x20 :=True
  <>fK_Wilbur :=True

End if

C_BLOB($1)                          ` BLOB parametrů exportu
C_BOOLEAN($2;$fBackground)          ` proces na pozadí (volitelné; True = pozadí)

    ` vytvoření místních proměnných
C_POINTER($pTable)                      ` Ukazatel k tabulce
C_LONGINT($LFieldType)                  ` čítač parametrů
C_LONGINT($LFormat)                     ` Umístění okna
C_BOOLEAN($fCRLF)                       ` vložit řádek po return
C_BOOLEAN($fCommasQuotes)           ` použít čárky a uvozovky
C_STRING(2;$sFieldDelimiter)        ` Oddělovač polí
C_STRING(2;$sRecordDelimiter)       ` Oddělovač záznamů
C_STRING(2;$sDelimiter)                 ` dočasný oddělovač
C_STRING(2;$sQU)                    ` Uvedení znaků pro tečky & uvozovky Export
C_LONGINT($LNumberOfFields)         ` počet polí k exportu
C_LONGINT($LNumberOfRecords)        ` Numbepočet záznamů k exportu
C_LONGINT($LPosition)               ` pozice znaků
C_LONGINT($LWindowID)                   ` ID okna pro zprávu
C_LONGINT($LTable)                  ` číslo tabulky
C_LONGINT($LOffset)                 ` Offset pro rozluštění BLOB
C_LONGINT($i;$j;$k)                     ` opakovací čítač
C_LONGINT($LDataLength)                 ` délka dat k exportu
C_LONGINT($LLastPosition)               ` pozice v paměťovém bloku která vrací poslední
znak
C_TIME($hDocumentReference)         ` odkaz k dokumentu
C_TEXT($tBuffer)                    ` buffer dat
C_TEXT($tField)                         ` dočasná data polí
ARRAY INTEGER($aiFieldNumbers;0)

    ` definice výchozích hodnot
$fBackground:=False
```

```
      ` Přiřazení BLOB parametrů k proměnným
$LOffset := 0
BLOB TO VARIABLE($1;$LTable;$LOffset)
BLOB TO VARIABLE($1;$aiFieldNumbers;$LOffset)
BLOB TO VARIABLE($1;$LFormat;$LOffset)
BLOB TO VARIABLE($1;$i;$LOffset)
$fCRLF := ($i=1)                              ` 1 značí ano
BLOB TO VARIABLE($1;$i;$LOffset)
$fCommasQuotes := ($i=1)                      ` 1 značí ano
BLOB TO VARIABLE($1;$sFieldDelimiter;$LOffset)
$sFieldDelimiter := Char(Num($sFieldDelimiter))     ` převede řetězec čísel na char
BLOB TO VARIABLE($1;$sRecordDelimiter;$LOffset)
$sRecordDelimiter:=Char(Num($sRecordDelimiter)) ` převede řetězec čísel na char
BLOB TO VARIABLE($1;Document;$LOffset)
SET BLOB SIZE($1;0)
If (Count parameters=2)
   $fBackground := $2
End if

$LNumberOfFields := Size of array($aiFieldNumbers)
ARRAY POINTER($pFields;$LNumberOfFields)
For ($i;1;$LNumberOfFields)
   $pFields{$i} := Field($LTable;$aiFieldNumbers{$i})   ` ukazatel k poli
End for

If ($fCRLF)
   $sRecordDelimiter := $sRecordDelimiter  + Char(Line feed)
End if
$pTable :=Table($LTable)                      `ukazatel k exportované tabulce.

If ($fBackground )                            ` export běží ve vlastním okně

   MENU BAR (1)  `nastaví nabídku pro proce tak, že nabídka bude prázdná
      `když bude zobrazené okno exportu.

   <>LBackground := <>LBackground + 1
   USE NAMED SELECTION ("<>ExportSelection")
   CLEAR NAMED SELECTION ("<>ExportSelection")
   SET CHANNEL (10; Document)

Else` export dat byl volán procesem který otevřel soubor

SET CHANNEL (12; "")

End if

If (OK = 1)

   If ($fBackground )
      HIDE PROCESS (Current process)        ` přesunout na pozadí
   End if
```

```
   FIRST RECORD ($pTable->)
   $LNumberOfRecords := Records in selection ($pTable->)

   $sTableName := Table name ($pTable)

   $LWindowID :=WIN_LNewWindow (200; 30; <>WIN_LCentered;
<>WIN_LStandardNoResize; "Export")
   If ($fBackground )
      PROCESS_UpdateWindowArray    (""; Current process;True)
   End if
   $Lpid := Process number ("$Logoff") ` je zde test logoff
   If ($Lpid > 0)
      ON EVENT CALL ("")                ` vypnout zachytávání událostí
      PAUSE PROCESS($Lpid )
   End if

   GOTO XY (6; 0)
   MESSAGE ("Export " + $sTableName + "…")

   Case of

      : (($LFormat = 0) | ($LFormat = 1))` Text or Merge formát

         ` vytvořit array typů polí
         ARRAY LONGINT (aLFieldTypes; $LNumberOfFields)

         For ($i; 1; $LNumberOfFields)
            GET FIELD PROPERTIES ($pFields{$i}; $LType)
            aLFieldTypes{$i} := $LType
         End for

         If ($fCommasQuotes)              ` děláme tečky a uvozovky
            $sQU :=<>sQU
         Else
            $sQU :=""
         End if

         If ($LFormat = 1)                ` Merge Formát. nejdřív exportovat názvy polí
            $sDelimiter := $sFieldDelimiter
            $tBuffer := ""

            For ($i; 1; $LNumberOfFields)

               If ($i = $LNumberOfFields) ` exportujeme poslední název pole
                  $sDelimiter := $sRecordDelimiter
               End if

               $tBuffer := $tBuffer +$sQU + GEN_sFriendlyName (Field name ($pFields{$i}))
+$sQU+ $sDelimiter
            End for
```

```
      SEND PACKET ($tBuffer)
   End if

      ` now export the records
   $tBuffer := " " * 32000                  ` blok 32000 místa
   $LLastPosition := 0

   For ($i; 1; $LNumberOfRecords)
      $sDelimiter := $sFieldDelimiter

      For ($j; 1; $LNumberOfFields)

         If ($j = $LNumberOfFields)          ` exportujeme poslední pole záznamu
            $sDelimiter := $sRecordDelimiter
         End if

         Case of
            : ((aLFieldTypes{$j} # Is Alpha Field) &  (aLFieldTypes{$j} # Is Text) &
(aLFieldTypes{$j} # Is Boolean))      ` převést na řetězec

               If ((aLFieldTypes{$j}=Is Real) | (aLFieldTypes{$j}=Is Integer) |
(aLFieldTypes{$j}=Is Longint))         ` If it's a number
                  $tData :=(String(($pFields{$j})->)+$sDelimiter)
               Else
                  $tData :=$sQU+(String(($pFields{$j})->)+$sQU+$sDelimiter)
               End if

            : (aLFieldTypes{$j} = Is Boolean)  `   je to Logické

               If ($pFields{$j}->)
                  $tData := $sQU+"True" +$sQU+ $sDelimiter
               Else
                  $tData := $sQU+"False" +$sQU+ $sDelimiter
               End if
Delimit
         Else

            $tData := $sQU+ (($pFields{$j}->) +$sQU+ $sDelimiter)
         End case

         $LDataLength := Length ($tData)
         If ($LLastPosition + $LDataLength > 30000)
            SEND PACKET (Substring($tBuffer; 1;$LLastPosition))
            $tBuffer := " "*32000
            $LLastPosition :=0
         End if

         For ($k;1;$LDataLength)
            $tBuffer[[$LLastPosition + $k]] := $tData[[$k]]
         End for
```

```
      $LLastPosition :=$LLastPosition +$LDataLengths
    End for

    NEXT RECORD ($pTable->)

    If ($i % 25 = 0)                    ` nehrnout dopředu věci pouze k zobrazení
      GOTO XY (6; 1)
      MESSAGE ("Záznam " + String ($i) + " z " + String ($LNumberOfRecords))
    End if

  End for

  If ($LLastPosition  > 0)
    SEND PACKET (Substring($tBuffer; 1;$LLastPosition))
  End if

: ($LFormat = 2)                        ` 4D Formát

  For ($i; 1; $LNumberOfRecords)

    SEND RECORD ($pTable->)
    NEXT RECORD ($pTable->)

    If ($i % 25 = 0)                    ` nehrnout dopředu věci pouze k zobrazení
      GOTO XY (6; 1)
      MESSAGE ("Record " + String ($i) + " of " + String ($LNumberOfRecords))
    End if

  End for

: ($LFormat = 3)                        ` Použít formulář ImportExport
    ` Účel: Umožnit exportu záznamů do platnétabulky použít formulář
    ` Note: Vývojář musí mít vytvořený formulář s názve ImportExport pro tuto práci
  FldDelimit := Num ($sFieldDelimiter)
  RecDelimit := Num ($sRecordDelimiter)
  OUTPUT FORM ($pTable->; "ImportExport")
  EXPORT TEXT ($pTable->; "")
  OUTPUT FORM ($pTable->; "Výstupní")

End case

CLOSE WINDOW
SET CHANNEL (11) ` Zavřít dokument

If ($fBackground )
  PROCESS_UpdateWindowArray   (""; Current process;True)
  <>LBackground := <>LBackground -1
End if

If (($Lpid > 0) & (<>LBackground =0))
  <>fUserPresent := True                ` nastavit pokud je uživatel zde
```

```
    RESUME PROCESS($Lpid )
  End if

End if                          ` OK = 1

  ` Konec metody
```

3. Vytvořte novou metodu projektu P_IMPEXP_ExportData následovně:

```
If (False)
    ` Metoda: P_IMPEXP_ExportData (BLOB; boolean)
    ` ACI Univerzita kurzy programování
    ` Generic ACI Shell Programming
    ` Vytvořeno: Kent Wilbur
    ` Datum: 1/16/97

  <>fGeneric :=True
  <>f_Version6x20 :=True
  <>fK_Wilbur :=True

End if

    ` Vytvoření parametrů
C_BLOB($1)                              ` BLOB parametrů exportu
C_BOOLEAN($2;$fBackground)` proces na pozadí (volitelné; True = pozadí)

    ` vytvoření místních proměnných
C_LONGINT($LParms)      ` počet přicházejících parametrů

$LParms:=Count parameters
If ($LParms=2)
  $fBackground := $2
Else
  $fBackground := False
End if

If (Compiled application)
  IMPEXP_ExportDataOptimized ($1;$fBackground )
Else
  IMPEXP_ExportData ($1;$fBackground )
End if
 `Konec metody
```

24.18. Používejte RELATE ONE místo LOAD RECORD

Kde je to možné a vhodné používejte RELATE ONE místo LOAD RECORD. RELATE ONE má optimalizované chování a jestliže je záznam již zaveden ve správném stavu a pokud se cizí klíč nezměnil, záznam se nebude znovu zavádět.

24.19. Vyhněte se podtabulkám

Když zavádíte záznam (kromě výstupního formuláře s použitím automatických relací) jsou zaváděna všechna pole záznamu současně. Podtabulka je typ pole, pole o více hodnota tzn. že všechny podzáznamy rodičovského záznamu jsou zavedeny do paměti jako součást rodičovského záznamu. To je samozřejmě velký rozdíl proti datům, která jsou ukládána v oddělených tabulkách se vztahem. Když je zaváděn záznam jedinců je měněn pouze platný výběr tabulky skupin (jestliže je vztah automatický) a pouze jeden záznam, platný záznam tabulky skupin je zaveden do paměti.

Kromě toho žádný z příkazů uváděných výše optimalizovaný pro server nepracuje s podtabulkami.

Podtabulky nejsou vždy špatným řešením. V některých případech mohou dost podstatně vylepšit chování vaší databáze. Znamená to však, že musí vše dobře zvážit a programovat pečlivě a podrobně vědět kdy je můžete použít a kdy ne. Pokud si nejste jisti je lepší se použití podtabulek vyhnout.

24.20. Zdroje (resource)

Zdroje jsou ukládány na serveru ve speciálních složkách. MAC4DX nebo WIN4DX podle toho pro kterou platformu jsou napsány. Tyto zdroje jsou převáděny na lokální disk klienta v době prvního spuštění databáze. Přístup ke zdrojům je rychlý, protože od této doby jsou uloženy lokálně ve složce ACI v systémové složce v souborech, které končí příponou .res. Kromě toho jsou zdroje typu externích modulů ukládány v téže systémové složce v podsložce s názvem databáze. Jsou umístěny na lokálním disku, aby mohli být v případě potřeby rychle vyvolány. Protože se zdroje nemění často obnovují se pouze při prvním spuštění či změnách velikosti souboru. První přenesení zdrojů při prvním spuštění může zabrat určitý čas, každý z možných procesorů X86, Pentium 68XXX nebo PPC potřebuje zdroje přesně vyhovující této CPU. Server rozesílá tyto zdroje inteligentně a zasílá na klienta pouze část kódu určenou pro toto CPU. Pokud se vám zdá čas prvního kopírování zdrojů příliš dlouhý můžete tyto zdroje kopírovat manuálně ze strojů se stejnou CPU. Při tomto způsobu buďte opatrní a ujistěte se vždy, že kopírujete zdroje mezi stroji se stejným CPU. Platí to především pro moduly a externí zásuvné metody, písma, zvuky a vnitřní zdroje dostupné Customizer Plus.

24.21. Jestliže mažete větší množství záznamů proveďte kompaktaci

Zkompaktování dat pomocí 4D Tools zmenší velikost datového souboru a sníží čas potřebný pro přístup k záznamům. Rovněž tabulky indexů a adres se při kompaktaci obnoví a zmenší a sníží se čas potřebný k jejich zavedení do paměti, samozřejmě se tím i sníží množství paměti použité na serveru.

24.22. Vyhněte se použití grafiky v oblasti obsahu výstupních formulářů

Obrázky, zaškrtávací políčka, voliče atd. se vykreslují pomaleji než text. Na novějších rychlých strojí není zpomalení tak výrazné.

26. Příkazy a metody, které přidávají speciální funkce pro 4D Server

26.1. Count users -> Integer

Tento příkaz vrátí počet uživatelů, kteří jsou připojen k serveru.

Tato informace může být použita k ovládání událostí, které se mají vykonat, když je připojen a pouze a právě jeden uživatel. Např. vymazání označených záznamů, potvrzování procesu atd.

26.2. Sequence number -> Longint

S použitím server se toto číslo zvyšuje dokonce i tehdy jestliže uživatel zruší nový záznam. Nepoužívejte jej.

26.3. Current date (*)

26.4. Current time (*)

U těchto příkazů jestliže nepoužijete žádný parametr vrátí hodnoty podle systémových hodin počítače klienta. To může občas znamenat problém a to při více uživatelích, kdy mohou být na jednotlivých počítačích klientů nastaveny různé časy nebo různá časová pásma. Jestliže v těchto příkazech použijete volitelnou hvězdičku obdržíte datum a čas podle hodin serveru. Použití tohoto příkazu však vyvolává určitý provoz na síti. Lepším řešením je použít tento příkaz pro získání datumu a času ze serveru při spuštění, porovnat jej s lokálním datumem a časem a odložit rozdíl do meziprocessní proměnné. Pak kdykoliv potřebujete datum a čas přidejte tento rozdíl k místnímu datumu a času, tím se vyhnete zbytečnému zatížení sítě.

26.5. On Server Startup

4D Server může provádět kód mezi jinými je na 4D Server automaticky prováděn kód triggerů. Protože v naší databázi jsou triggery použity k obnově datumu a času úprav záznamů potřebujeme se ujistit, že máme na serveru nastaveny určité proměnné, které budeme používat. Server má své speciální metody databáze a jedna z nich se provádí pokaždé, když je server spouštěn. Vytvoříme a použijeme tuto metodu k inicializaci některých proměnných potřebných pro triggery.

26.5.1. Použití Current date a time účinněji pro síť

1. Přejděte do Prostředí návrháře.

2. Upravte metodu projektu MyStartup následovně:

```
•	C_LONGINT ($LDateDifference)
•	C_DATE ($dServerDate; $dMyDate;)
•	C_TIME ($hServerTime; $hMyTime; <>hTimeDifference)

•	$dServerDate := Current date (*)		` získání datumu ze serveru
•	$dMyDate := Current date			` Získání vlastního datumu

•	$LDateDifference := $dServerDate-$dMyDate	` nastavení rozdílu

•	$hServerTime := Current time (*)		` získání času ze serveru
•	$hMyTime := Current time `Get My time
•	<>hTimeDifference := $hServerTime-$hMyTime ` nastavení rozdílu
•	<>hTimeDifference := <>hTimeDifference +$LDateDifference * †24:00:00† ` nastavení
celkového rozdílu času
```

3. Vytvořte metodu databáze PřiSpuštěníServeru

```
.  ` Metoda: On Server Startup
   ` ACI Univerzita kurzy programování
   ` Vytvořeno: Jim Steinman
   ` Datum: 1/16/97
   ` Účel: nastaví prostředí na uživatelské vlastní hodnoty

<>hTimeDifference := †00:00:00†
<>LMainProcess := Current process
   `Konec metody
```

4. Nahraďte metodu projektu GEN_TimeStamp následovně nebo importujte pomocí textového editoru:

```
If (False)
      ` Metoda: GEN_TmeStamp (Date; Time)
      ` ACI Univerzita kurzy programování
      ` Generic ACI Shell Programming
      ` Vytvořeno: Jim Steinman
      ` Datum: 15/1/97

      ` Účel: Tato metoda naplňuje datumy a čas vytvoření a úprav.

   <>fGeneric := True
   <>f_Version6x10 := True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

         ` Modified: 17/1/97 k užití ofsetu proměnných k serveru
```

```
   <>fK_Wilbur:=True

      `$1 – uklazatel na obnovované datum
      `$2 -   ukazatel na obnovovaný čas

End if

      ` Define the parameters
C_POINTER($1)  ` ukazatel na datum
C_POINTER($2)  ` ukazatzel na čas

      ` definuje lokální proměnné
C_DATE($dDate)
C_TIME($hTime)

$dDate := Current date
$hTime := Current time+<>hTimeDifference

Case of
   : ($hTime > †24:00:00†)
      While ($hTime > †24:00:00†)
         $dDate := $dDate+1
         $hTime := $hTime -†24:00:00†
      End while
   : ($hTime<†00:00:00†)
      While ($hTime < †24:00:00†)
         $dDate := $dDate -1
         $hTime := $hTime +†24:00:00†
      End while
End case

` návrat datumu a času
$1-> := $dDate
$2-> := $hTime
      ` Konec metody
```

4. Otestujte kód s užitím serveru a při nastavení různého datumu a času na klientu a serveru.

27. Uložené procedury

27.1. Stručný přehled

Uložené procedury pro 4D Server jsou normální metody projektu pro které vývojář určí, že se mají provést na serveru.

4D Server podobně jako 4th Dimension a 4D Client je schopen spustit a vykonávat vedle sebe několik nezávislých procesů pomocí sdílení času procesoru.

Uložené procedury jsou zařazeny do tohoto kontextu a vykonávají se na serveru spolu s jinými procesy:

- Procesy jádra (Kernel) vykonávají různé operace databázového jádra
- Uživatelské procesy vykonávají úlohy pro 4D Client nebo 4D Open
- Procesy uložených procedur vykonávají metody projektu určené vývojářem k vykonání na počítači serveru

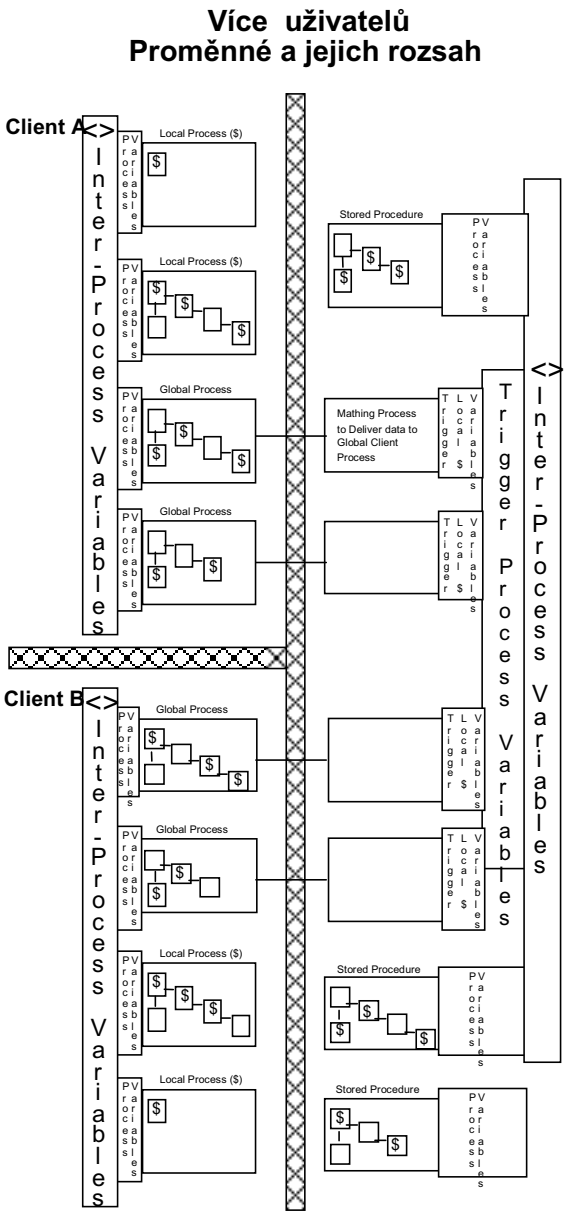
Procesy uložených procedur sdílejí následující prostředí:

- Do procedury databáze PřiSpuštění serveru mohou být zahrnuty návrhářem databáze
- Příkaz New process může pomocí parametrů vytvořit nový proces na 4D Server: další volání New process z uložené procedury vytvoří další uloženou proceduru (proces na serveru).
- Příkazy pro ovládání rozhraní jako BRING TO FRONT nemají žádný význam na počítači serveru, protože procesy uložených procedur nemohou obsahovat rozhraní uživatele. Je možno otevřít okno se zobrazením informací na serveru je to však neooperativní.
- Všechny procesy uložených procedur sdílejí stejnou tabulku meziprocessních proměnných. Tato tabulka je stejná i pro procesy všech 4D Client procesů spuštěných na serveru. Je však nutné si uvědomit, že meziprocessní proměnné na počítači klienta a meziprocessní proměnné procesu klienta na počítači serveru jsou zcela jiné proměnné.
- Procesy uložených procedur na 4D Server sdílejí stejné meziprocessní sady a pojmenované výběry. Platnost meziprocessních sad a pojmenovaných výběrů je pouze na počítači serveru.

- Lokální semafory používané v uložených procedurách jsou lokální pro počítač 4D Serveru.
- Mechanisky CALL PROCESS a Outside call nemají žádný význam na počítači serveru, protože procesy uložených procedur nemají rozhraní uživatele.

27.2. Proměnné a jejich rozsah na serveru

Rozsah proměnných a jejich platnost na serveru je odlišná od klienta. Server má tabulku meziprocessních proměnných, která je sdílena mezi všemi procesy na serveru. Každá uložená procedura má rovněž své vlastní proměnné lokální a procesu tak jako klient. Rozdíl je platnost proměnných používaných triggery. Ačkoliv triggery nejsou typické uložené procedury musíte být opatrní, protože triggery sdílejí tabulku proměnných procesu mezi všemi triggery všech klientů. Proto by jste neměli používat proměnné procesu v triggerech.



27.3. Výměna dat mezi procesy klientů a serveru

Do této doby jsme se zabývali komunikací mezi procesy pouze v jedné aplikaci 4D a to na stroji buď s jednouživatelskou 4D nebo na stroji ve víceuživatelském prostředí se 4D Client.

Meziprocessní komunikace může být vytvořena pomocí následujících objektů a příkazů prostředí 4D:

- Příkazy ovládání procesů jako New process
- Příkazy ovládání rozhraní procesů jako BRING TO FRONT
- Záznamy (*)
- Meziprocessní proměnné, sady a pojmenované výběry
- Lokální semafory
- Globální semafory (*)
- CALL PROCESS a On Outside Call
- Pomocí metody databáze PřiSpuštění, používané pro nastavení výchozích hodnot proměnných a arrray pro meziprocessní komunikaci.

(*) Pomocí těchto objektů je možno rovněž komunikovat mezi procesy na libovolném stroji v systému 4D Client/Server.

Se zavedením uložených procedur na 4D Server lze přidat do architektury 4D Client/Server dva další hlavní rysy:

- Meziprocessní komunikaci (stejnou jako na 4D Client) lze přidat (s určitými omezeními) na počítač 4D Server.
- Lze přidat nové mechanismy pro umožnění komunikace mezi procesy běžícími na 4D Client s procesy běžícími na 4D Server.

Tyto nové mechanismy jsou:

- Odesílání a přijímání proměnných pomocí příkazu SET PROCESS VARIABLE, GET PROCESS VARIABLE a VARIABLE to VARIABLE.
- Odesílání a přijímání nových dokumentů na či z počítače serveru pomocí příkazů SET DOCUMENT a GET DOCUMENT.
- Výměna sad a pojmenovaných výběrů.
- Předávání parametrů do procesů v době jejich spuštění.

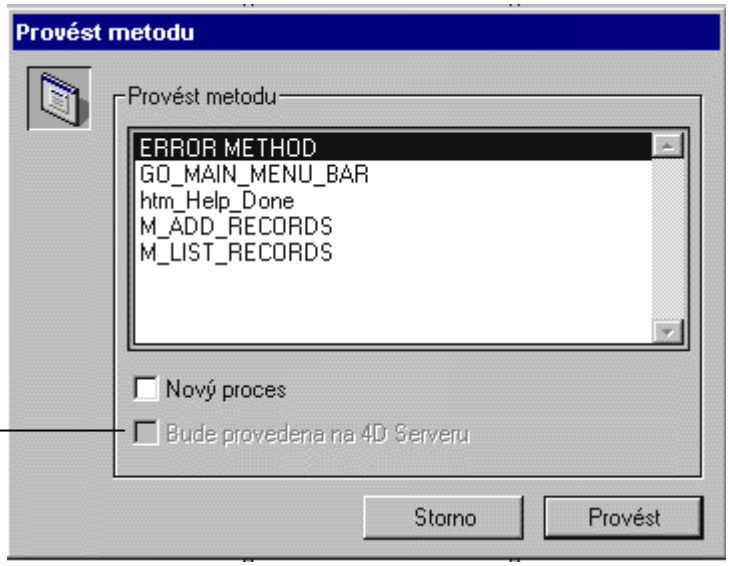
27.3. Jak provést uloženou proceduru?

Uložená procedura může být provedena dvěma způsoby:

27.3.1. Provedením v Prostředí uživatele

1. V Prostředí uživatele zvolte Zvláštní – Provést metodu

2. V dialogovém okně Provést metodu zaškrtněte políčko Bude provedena na Serveru



Nedostupné v jednouživatelské verzi

27.3.2. Execute on server (Procedura ; VelikostStack { ; NázevProcesu { ; Parametr1 { ; Parametr2 ;... } } } { ; * }) -> ČísloIDProcesu

Příkaz Execute on server spustí nový proces na počítači serveru s názvem *NázevProcesu* a vrátí číslo odkazu na tento proces.

Procedura je název procedury, kterou chceme provést v novém procesu na serveru.

VelikostStack je množství paměti přidělné paměti stack tohoto procesu. Stack je místo v paměti používané k uložení objektu procesu jako volání procedur, lokální proměnné, parametry metod vynuceně uložené záznamy do stack a rekursivní volání. Velikost 16K do32K je vyhovující pro většinu procesů. Minimální velikost stack je 16K.

Volitelný paramter *NázevProcesu* je název procesu tento název se objeví v hlavní okně 4D Server v sekci *Stored Procedures*.

Důležitá poznámka: Jestliže předáte parametry uložené procedury musíte nutně předat *NázevProcesu*.

Parametry *Parametr1*, *Parametr2*,...,*ParametrN* jsou parametry předávány uložené proceduře. Jestliže předáte uložené proceduře parametry jsou obdrženy jako ” $1, $2,...,$N a takto se na ně uvnitř procedury odkazujete.

Volitelný parametr * dovoluje určit jestli uložená procedura bude jedinečná nebo ne. Test na jedinečnost se provádí podle názvu procesu. Jestliže je uložená procedura již spuštěna příkaz Execute on server vrátí pouze *ČísloIDProcesu* pro tuto uloženou proceduru.

Poznámka: Aby se rozlišili čísla procesů klientů od čísel procesů uložených procedur příkazy Execute on server a rovněž New process. (volaný na počítači serveru) navracejí **záporná** čísla procesu.

27.4. Meziprocessní komunikace

27.4.1. SET PROCESS VARIABLE (IDProcesu ; CílováProm1 ; Výraz1 { ; CílováProm2 ; Výraz2; ... })

Příkaz SET PROCESS VARIABLE změní hodnotu proměnných v určeném procesu na hodnoty určené ve VýrazX. Když nastavujeme proměnné v procesu serveru z počítače a procesu klienta musí být IDProcesu záporné číslo. IDProcesu musí být kladné číslo, když nastavujeme proměnné v jiném procesu na stejném počítači. CílováPromX musí být proměnnou procesu nebo meziprocessní proměnou.

Poznámka: Uložené procedury (procesy na Serveru) nemohou nastavovat proměnné v procesech klienta. Tento způsob komunikace je jednosměrnný.

27.4.2. GET PROCESS VARIABLE (IDProcesu ; ZdrojováProm1 ; CílováProm1 { ; ZdrojováProm2 ; CílováProm2 ; ... })

Příkaz GET PROCESS VARIABLE čte proměnné z určeného procesu a nastavuje hodnoty v proměnných CílováPromX ve volajícím procesu. Když nastavujeme proměnné z procesu serveru na klientu musí být IDProcesu záporné číslo. IDProcesu musí být kladné číslo, když nastavujeme proměnné z jiného procesu na stejném stroji. ZdrojováPromX musí být proměnná procesu nebo meziprocessní.

Poznámka: Uložené procedury (procesy na serveru) nemohou obdržet proměnné z procesu klienta, tento proces komunikace je jednosměrný.

27.4.3. VARIABLE TO VARIABLE (IDProcesu; CílováProm1; ZdrojováProm1 {; CílováProm2; ZdrojováProm2; ...; CílováPromN; ZdrojováPromN})

Příkaz VARIABLE TO VARIABLE předá do CílovéPromX v určeném procesu hodnotu proměnné ZdrojováVarX z volajícího procesu.

VARIABLE TO VARIABLE provádí totéž jako SET PROCESS VARIABLE s následujícími rozdíly:

- Zatímco SET PROCESS VARIABLE předává výraz (a proto nemůže předat array jako celek) VARIABLE TO VARIABLE předává zdrojovou proměnnou (a proto můžete předat array jako celek).
- V SET PROCESS VARIABLE může být každá cílová proměnná proměnnou nebo prvkem array, ale nemůže být array jako celek. V příkazu VARIABLE TO VARIABLE může být každá cílová proměnná buď prvek array nebo array jako celek.

  Pro každý pár CílováPromX, ZdrojováPromX (či výraz) musí být typ zdroje kompatibilní s cílovou proměnnou. Jinak můžete obdržet v cílových proměnných nesmyslnné hodnoty. V interpretačním módu, jestliže cílová proměnná neexistuje je vytvořena a je jí přiřazen typ a hodnota zdrojové proměnné. Volají proces přiřazuje hodnoty do proměnných cílového procesu (cílový proces není žádným způsobem upozorněn, že jiný proces zapisuje do jeho proměnných).

Omezení

VARIABLE TO VARIABLE nepřijímá místní proměnné jako cílové proměnné.

VARIABLE TO VARIABLE přijímá libovolný typ cílových proměnných procesu a meziprocessních proměnných kromě:

- Ukazatelů
- Array ukazatelů
- Dvoudimenzionálních array

Cílový proces musí být uživatelským procesem, nemůže být procesem jádra. Jestliže cílový proces neexistuje je generována chyba.

27.5. Správné použití uložených procedur

- Apply Formula
- Dotazy a třídění prováděné příkazy Query nebo Order By Formulas
- Komunikace mezi klienty
- Složité dotazy, které mohou být urychleny pomocí manipulace se sadami nebo Clustery
- Import záznamů do databáze
- Export záznamů do souboru

27.6. Na co nepoužívat uložené procedury

- Tisky

27.7. Rychlý pohled do databáze s uloženými procedurami

Ukážeme si několik jednoduchých příkladů na uložené procedury. Těmito příklady si předvedeme jaký je rozdíl mezi prováděním určitých částí kódu ze stroje klienta proti stroji serveru. Kromě toho použijeme příkazy GET PROCESS VARIABLE a SET PROCESS VARIABLE.

Demonstrace č. 1 je jednoduchá metoda pro vytvoření 25 záznamů z klienta nebo ze serveru. Spustíme oddělené procesy, které vytvoří 25 záznamů. Poznámka: Tyto procesy používají parametry předané k vytvoření patřičných záznamů. Zobrazovaný čas trvání je pouze časem nezbytným k vytvoření záznamu a nezahrnuje čas na spuštění nového procesu, alokování paměti atd.

1. Spusťte vytvoření záznamu z počítače klienta.

Druhá část demonstrace použije k vytvoření záznamů server. Na serveru v seznamu procesů uvidíte jak jsou záznamy vytvářeny. Když server skončí, klient přečte čas, který server spotřeboval a zobrazí výsledek na počítači klienta. Je zde použita metoda odložení procesu a komunikaci mezi procesy počítače klienta a serveru.

2. Spusťte vytvoření záznamu pomocí uložené procedury.

Všimněte si času vyjádřeného v množství tiků. Spotřebovaný čas jsme zjistili přes příkazy 4D, které dokáží změřit počet spotřebovaných tiků na CPU. Více si o těchto možnostech 4D řekneme po této kapitole.

Naše další demonstrace se bude týkat především vnitřní komunikace procesů mezi klientem a serverem. První demonstrace nám umožňuje sledovat průběh práce serveru pomocí zpětné odezvy na klienta. Toto demo vytváří 500 záznamů.

3. Spusťte uloženou proceduru ke sledování Observe..

Nakonec si ukážeme řízení serveru z počítače klienta.

4. Spusťte na serveru uloženou proceduru Control Server.

27.8. Začínáme s uloženými procedurami

Uložené procedury by neměli být nadměrně používány. Příliš mnoho uložených procedur spuštěných najednou zpomalí server a zabere velkou část paměti. Na druhé straně správné použití může výrazně zlepšit chování celého prostředí klietn/server. Pouze testování vaší určité aplikace vám pomůže určit nejlepší způsob použití uložených procedur.

Další příklad nám dovolí sledovat chování při příkazu APPLY TO SELECTION. Jsou zde zahrnuty 4 testy na obnovu cen v tabulce[Produkty].

27.8.1. Testování uložených procedur na rychlost a účinnost

Prvním testem je dobře známé použití APPLY TO SELECTION z počítače klienta. Druhý test používá SELECTION TO ARRAY - ARRAY TO SELECTION na počítači klienta. Poslední dva jsou stejné jako předchozí dva testy, ale jsou spuštěny na serveru jako uložené procedury.

1. Vytvořte novou metodu projektu GEN_ShowTimingResults následovně nebo importujte pomocí textového editoru:

```
If (False)
      ` Metoda: GEN_ShowTimingResults (longint;longint)
      ` ACI Univerzita kurzy programování
      ` Vytvořeno: Jim Steinman
      ` Datum:1/1/6/97

      ` Účel: Zobrazí výsledky v čase vyjádřeném v ticích.

   <>fGeneric:=True
   <>f_Version6x20:=True
   <>fJ_Steinman:=True

End if

      ` Declare parameters
C_LONGINT($1)                    ` počáteční tik
C_LONGINT($2)                    ` koncový tik

      ` Declare local variables
C_LONGINT($LTotalTicks)              ` celkem tiků
C_LONGINT($LTicks)               ` zbylé tiky
C_LONGINT($LSeconds)             ` zbývající sekundy
C_LONGINT($LMinutes)             ` zbývající hodiny
C_LONGINT($LHours)               ` celkem hodin
C_STRING(255;$sMessage)              ` zpráva k zobrazení
C_STRING(15;$sAdd)                   ` dočasná část zprávy

$sMessage := ""
$LTotalTicks := $2 - $1

If ($LTotalTicks>59)
   $LSeconds := $LTotalTicks\60
   $LTicks := $LTotalTicks%60

   If ($LSeconds>59)
      $LMinutes := $LSeconds\60
      $LSeconds := $LSeconds%60
```

```
      If ($LMinutes>59)
         $LHours := $LMinutes\60
         $LMinutes := $LMinutes%60

         If ($LHours>1)
            $sAdd := String($LHours) + " Hodin "
         Else
            $sAdd := "1 Hour "
         End if

         $sMessage := $sMessage + $sAdd
      End if
   End if

Else
   $LTicks := $LTotalTicks
End if

$sAdd := ""
If ($LMinutes>1)
   $sAdd := String($LMinutes) + " Minut "

Else
   If ($LMinutes=1)
      $sAdd := "1 Minuta "
   End if
End if

$sMessage := $sMessage + $sAdd

$sAdd := ""
If ($LSeconds>1)
   $sAdd := String($LSeconds) + " Sekund "

Else
   If ($LSeconds=1)
      $sAdd := "1 Sekunda "
   End if
End if

$sMessage := $sMessage + $sAdd

If ($LTicks>1)
   $sMessage := $sMessage+String($LTicks) + " Tiků "

Else
   If ($LTicks=1)
      $sMessage := $sMessage + "1 Tik"
   End if
End if
```

```
ALERT("Celkem spotřebuvaný čas je: " + $sMessage)
  `Konec metody
```

2. Vytvořte novou metodu projektu SPE_ApplySelectionEvaluation následovně nebo importujte pomocí textového editoru:

```
If (False)
      ` Metoda: SPE_ApplySelectionEvaluation
      ` ACI Univerzita kurzy programování
      ` Vytvořeno: Jim Steinman
      ` Datum:1/1/6/97

      ` Účel: Demonstruje různé způsoby použití
      ` APPLY TO SELECTION v prostředí více uživatelů.

   <>fGeneric:=True
   <>f_Version6x20:=True
   <>fJ_Steinman:=True

End if

      ` Declare local variables
C_LONGINT($i)                    ` čítač smyčky
C_LONGINT($LStart)                   ` počateční čas testování
C_LONGINT($LEnd)                 ` konečný čas testování
C_LONGINT($Lpid)                 ` ID procesu
C_LONGINT($LWindowID)                ` ID okna
C_LONGINT($LLocked)                  ` příznak uzamčení
C_BOOLEAN($fReadOnly)                ` stav tabulky

If (Records in selection([Produkty]) # Records in table([Produkty]))
   ALERT("V platném výběru musí být všechny záznamy")
Else

   INPUT FORM([zDialogy];"SPE_ApplyToSelection";*)
   $LWindowID :=WIN_LNewWindow (220; 280; <>WIN_LCentered)
   DIALOG([zDialogs];"SPE_ApplyToSelection")
   CLOSE WINDOW

   If (OK=1)

      $LWindowID :=WIN_LNewWindow (220; 20; <>WIN_LCentered)
      GOTO XY(14;2)
      MESSAGE("Načítám ceny")

      If (sb3=1)
         $Lpid := Execute on
server("SPE_ApplyToSelection";32000;"ApplyToSelectionProducts";(rb1=1);
                  rAmount)
```

```
        Repeat
          PROCESS _MyDelay(Current process;10)
          GET PROCESS VARIABLE($Lpid;LStart;$LStart)
          GET PROCESS VARIABLE($Lpid;LEnd;$LEnd)
        Until ($LEnd# 0)

      Else

        If (sb4=1)
          $Lpid :=Execute on
server("SPE_ArrayToSelection";32000;"ArrayToSelectionProducts";(rb1=1);
                  rAmount)

          Repeat
            PROCESS _MyDelay(Current process;10)
            GET PROCESS VARIABLE ($Lpid;LStart;$LStart)
            GET PROCESS VARIABLE($Lpid;LEnd;$LEnd)
          Until ($LEnd # 0)

        Else

          $fReadOnly:=Read only state([Produkty])` ujistit se o odemčenosti tabulky
          LOCK_Tables2ReadWrite (->[Produkty])

          $LStart:=Tickcount

          Case of
            : (sb1=1)
              START TRANSACTION        ` všechy změny nebo žádné

              If (rb1=1)
                APPLY TO SELECTION([Produkty];[Produkty]Cena
:=Round([Produkty]Cena+
                        rAmount;2))
              Else
                APPLY TO SELECTION([Produkty];[Produkty]Cena
:=Round([Produkty]Cena+
                          [Produkty]Cena * rAmount/100;2))
              End if

              UNLOAD RECORD([Produkty])

            : (sb2=1)
              START TRANSACTION                ``  všechy změny nebo žádné

              SELECTION TO ARRAY([Produkty]Cena;arAmount)  ` všechny ceny
současně

              For ($i;1;Size of array(arAmount))        ` smyčka změn cen
                If (rb1=1)
                  arAmount{$i} := arAmount{$i}+rAmount
```

```
            Else
               arAmount{$i} := Round(arAmount{$i}+arAmount{$i}*rAmount/100;2)
            End if
         End for

         GOTO XY(14;2)
         MESSAGE("Ukládám ceny")

         ARRAY TO SELECTION(arAmount;[Produkty]Cena) ` uložit všechny změny

      End case

      If ((sb1=1) | (sb2=1))
         If (Records in set("LockedSet")=0)
            VALIDATE TRANSACTION
            $LLocked := 1                ` kladné jako úspěch
         Else
            CANCEL TRANSACTION
            $LLocked := -1               ` záporné jako neúspěch
         End if

         $LEnd :=Tickcount * $LLocked
      End if

      If ($fReadOnly)                    ` zpět uzamčenost byla-li
         LOCK_Tables2ReadOnly(->[Produkty])
      End if
      UNLOAD RECORD([Produkty])

   End if
End if
CLOSE WINDOW

If ($LLocked<0)
   ALERT("Některé záznamy uzamčeny, změny neprovedeny!")
End if
GEN_ShowTimingResults ($LStart;Abs($LEnd))

  End if
End if
   `Konec metody
```

3. Vytvořte tlačítko ve formuláři [Produkty];"Výstupní" s následujícími vlastnostmi:

Název:	bChangePrices
Typ:	Button
Automatická akce:	No Action
Text tlačítka:	Change Prices…
Události:	On Click

4. Pro toto tlačítko vytvořte následující metodu objektu:

```
If (False)
      ` Object Metoda: bChangePrices
      ` ACI Univerzita kurzy programování
      ` Vytvořeno: Jim Steinman
      ` Datum: 16/1/97

      ` Účel: Umožní testování uložených procedur pro změny cen.

   <>f_Version6x20 :=True
   <>fJ_Steinman := True

End if

SPE_ApplySelectionEvaluation
   ` Konec metody
```

5. Vytvořte novou metodu projektu SPE_ApplyToSelection následovně nebo importujte pomocí textového editoru:

```
If (False)
      ` Metoda: SPE_ApplyToSelection
      ` ACI Univerzita kurzy programování
      ` Vytvořeno: :  Jim Steinman
      ` Datum:16/1/97

      ` Účel: Demonstruje princip uložených procedur na
      ` APPLY TO SELECTION v prostředí více uživatelů.

   <>f_Version6x20:=True
   <>fJ_Steinman:=True

End if

      ` deklarace parametrů a jejich přiřazení lokálním proměnnm
C_BOOLEAN($1;$fFixedAmount)       ` je to konstantní změna
C_REAL($2;rAmount)                      ` změna o procenta

    ` Declare local variables
C_LONGINT(LStart;)                      ` počateční tik
C_LONGINT(LEnd)                         ` koncový tik
C_LONGINT($LLocked)                     ` příznak uzamčenosti

      ` přiřazení parametrů lokálním
$fFixedAmount :=$1                      ` změna o konstantu
$rAmount :=$2                         ` změna o procenta

LEnd := 0
```

```
ALL RECORDS([Produkty])
UNLOAD RECORD([Produkty])

LStart :=Tickcount

START TRANSACTION

If ($fFixedAmount)
   APPLY TO SELECTION([Produkty];[Produkty]Cena :=Round([Produkty]Cena+
                              rAmount;2))
Else
   APPLY TO SELECTION([Produkty];[Produkty]Cena :=Round([Produkty]Cena+
                                [Produkty]Cena * rAmount/100;2))
End if

If (Records in set("LockedSet")=0)
   VALIDATE TRANSACTION
   $LLocked := 1                          ` kladné jako úspěch
Else
   CANCEL TRANSACTION
   $LLocked := -1                         ` záporné jako neúspěch
End if

LEnd :=Tickcount * $LLocked

PROCESS _MyDelay(Current process;500)
  `Konec metody
```

6. Vytvořte novou metodu projektu SPE_ArrayToSelection následovně nebo importujte pomocí textového editoru:

```
If (False)
      ` Metoda: SPE_ArrayToSelection
      ` ACI Univerzita kurzy programování
      ` Vytvořeno: :  Jim Steinman
      ` Datum:16/1/97

      ` Účel: Demonstruje princip uložených procedur na
      ` ARRAY TO SELECTION v prostředí více uživatelů.

   <>f_Version6x20:=True
   <>fJ_Steinman:=True

End if
` deklarace parametrů a jejich přiřazení lokálním proměnnm
C_BOOLEAN($1;$fFixedAmount)       ` je to konstantní změna
C_REAL($2;rAmount)                      ` změna o procenta

    ` Declare local variables
C_LONGINT(LStart;)                      ` počateční tik
```

```
C_LONGINT(LEnd)                         ` koncový tik
C_LONGINT($LLocked)                     ` příznak uzamčenosti

    ` přiřazení parametrů lokálním
$fFixedAmount :=$1                      ` změna o konstantu
$rAmount :=$2                       ` změna o procenta


LEnd := 0

ALL RECORDS([Produkty])
UNLOAD RECORD([Produkty])

LStart :=Tickcount

START TRANSACTION

SELECTION TO ARRAY([Produkty]Cena;arAmount)         `všechny ceny současně

For ($i;1;Size of array(arAmount))          ` smyčka změn cen
  If ($fFixedAmount)
    arAmount{$i} := arAmount{$i}+rAmount
  Else
    arAmount{$i} := Round(arAmount{$i}+arAmount{$i}*rAmount/100;2)
  End if
End for

ARRAY TO SELECTION(arAmount;[Produkty]Cena)         ` uložení změn
If (Records in set("LockedSet")=0)
  VALIDATE TRANSACTION
  $LLocked := 1                         ` kladné jako úspěch
Else
  CANCEL TRANSACTION
  $LLocked := -1                        ` záporné jako neúspěch
End if

LEnd :=Tickcount * $LLocked

PROCESS _MyDelay(Current process;20)
 `Konec metody
```

8. Přejděte do prostředí Vlastní nabídky a vyzkoušejte zvlášť každý ze čtyř testů .

Co jste očekávali? Proč nebylo podstatně větší zlepšení? Co lze říci o snížení provozu na síti? Kterou z možností pravděpodobně vyberete?

Použijte tyto příklady uložených procedur a triggerů jako návod jak si rozmyslet rozvržení provádění vaší databáze.

27.8.2. Tickcount -> Číslo

Tento příkaz navrátí počet tiků (1/60 sekundy) spotřebovaný procesem od jeho spuštění.

28. Ovládání diskových souborů nastavení

28.1. Pokud možnost umístěte složku s nastaveními na server jiný než počítač 4D Server

Sady

Rychlé zprávy

Kritéria dotazů

28.2. Vzory dokumentů pro moduly

Jako výchozí je nastaveno ukládání vzorů dokumentů modulů (4D Write, 4D Calc, 4D Draw) na počítači serveru.

Toto nastavení může být pro každý modul zvlášť změněno pomocí Customizer Plus.

29. Zabezpečení systému pomocí hesel

29.1. Soubor cesty

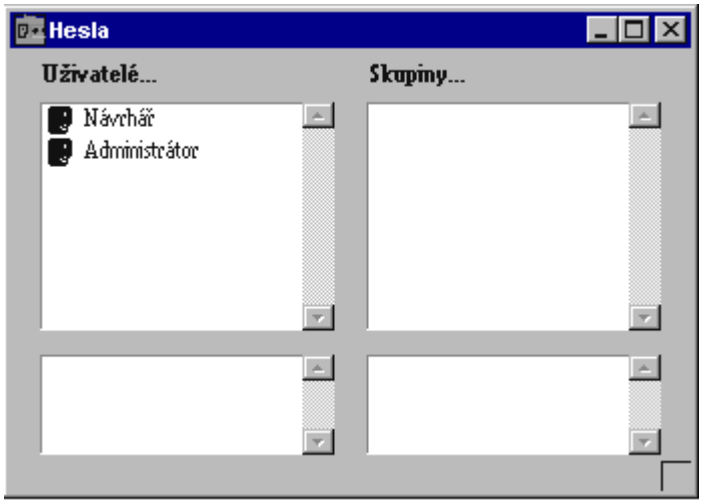
Je to soubor, který může být umístěn na počítač klienta a odkazovat na určitou databázi. Vytvořením souboru cesty můžete vyloučit dialog Při spuštění, kde si uživatel vybírá ke které databázi se připojit.

29.2. Vytváření hesel

Návrhář -Návrhář může vytvořit uživatele, kteří budou mít přístup do Prostředí návrháře.

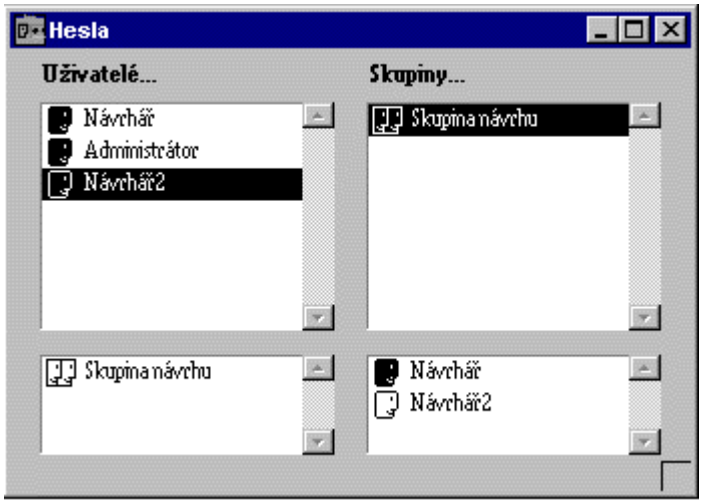
Administrátor- Administrátor by měl vytvářet uživatele, kteří nemají přístup do Prostředí návrháře.

Pouze uživatelké vytvoření administrátorem budou při volbě Hesla ¬ Uložit skupiny v Editoru hesel uloženy do diskového souboru:



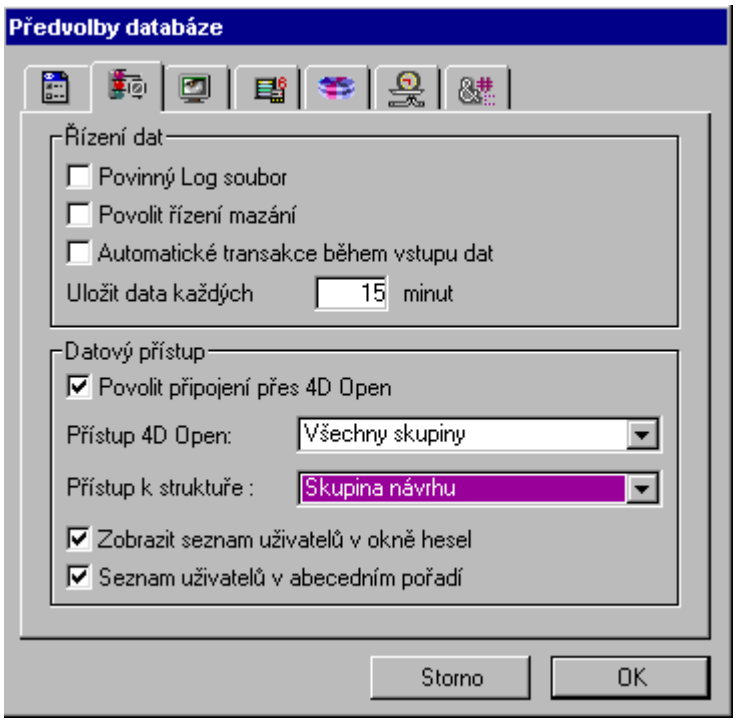
29.3. Přístup do Prostředí návrháře

Do Prostředí návrháře můžete povolit přístup více uživatalům a nebo jako výchozí možnost můžete zakázat přístup ostatních uživatelů do Prostředí návrháře.



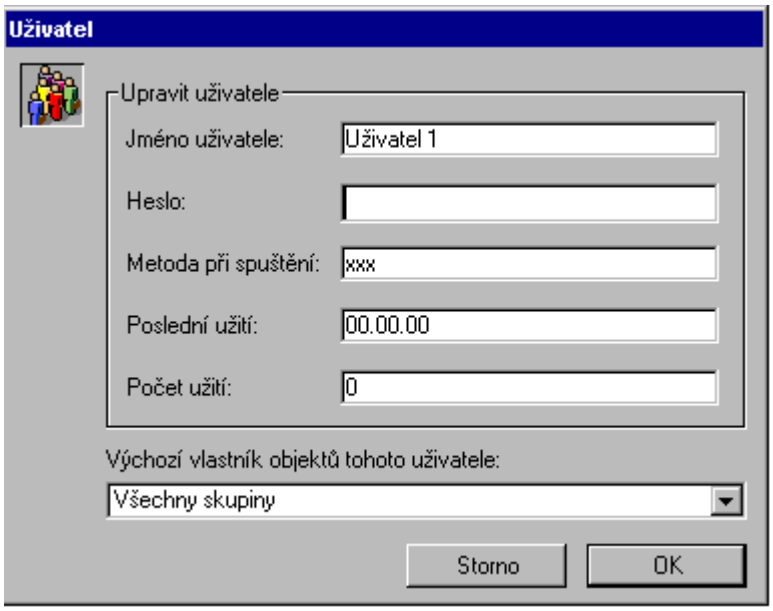
29.3.1. Nastavení systému hesel

1. Vraťte se do databáze ACI Video.
2. Přejděte do Prostředí návrháře.
3. Zvolte z nabídky Návrh ->Hesla.
4. Přidělte heslo uživateli Návrhář (toto heslo si určitě zapamatujte nikdo jiný jej již nemůže změnit a pokud jej zapomenete nebudete mít již přístup do databáze).
5. Vytvořte jednoho dodatečného uživatele a nazvěte jej Návrhář2.
6. Vytvořte skupinu nazvanou Skupina návrhu.
7. Přeneste uživatele Návrhář a Návrhář2 do Skupina návrhu.
8. Zvolte Soubor ¬ Předvolby databáze….
9. Změňte na stránce Řízení dat a přístup, přístup ke struktuře na Skupina návrhu.
10. Restartujte několikrát databázi a připojte se jako Návrhář, Administrátor a Návrhář2 a sledujte přístup do Prostředí návrháře.



29.4. Přístup do Prostředí uživatele

Můžete omezit přístup uživatelů do Prostředí uživatele. Toto se provádí přiřazením metody Při startu jednotlivým uživatelům. Tato metoda pokud existuje bude provedena po standartní metodě databáze Při spuštění. Jestliže tato metoda neexistuje stačí to, že byla uvedena k zakázání přístupu uživatele do Prostředí uživatele.



29.4.1. Zakázání přístupu uživatele do Prostředí uživatele

1. Navraťte se do databáze ACI Video jako administrátor.
2. Přejděte do Prostředí návrháře (předtím si povolte přístup).
3. Zvolte Návrh ->Hesla.
4. Vytvořte dalšího uživatele nazvaného Uživatel1.
5. Přiřaďte tomuto uživateli metodu Při startu nazvanou XXX.
6. Restartujte databázi a přihlašte se jako Uživatel1 a sledujte možnost přístupu do Prostředí uživatele.

29.5. Přístup k modulům

Předpokládejme, že máte svůj systém 20 uživatelů. Pouze 6 z nich potřebuje přístup k určitým modulům. Protože přístup k modulům je zaručen při spuštění je možné, že těchto 6, kteří potřebují modul nebude prvních 6, kteří se přihlásí k databázi. Takže pak potřebujete nastavit skupinu, které přístup k modulům dovolen. Pak těchto 6 uživatelů, kteří potřebují přístup k modulům přiřaďte do takovéto skupiny. Tímto omezíte potřebu 20 licencí na moduly jen na počet licencí potřebných pro uživatele ve skupině.

29.6. Příkaz EDIT ACCESS

Tento příkaz dovoluje uživatelům přístup do systému hesel. Návrhář a administrátor mohou přidávat nové uživatele a rovněž je přiřazovat do skupin. Uživatelé mohou být vkládáni a odstraňováni ze skupin. Skupiny mohou být upravovány libovolným

uživatelem, kterému bylo dáno právo vlastníka této skupiny. Pouze uživatelé, kteří jsou vlastníky nějaké skupiny mohou vstupovat do Editoru hesel.

29.7. Příkaz CHANGE ACCESS

Tento příkaz zobrazí uživateli dialogové okno přihlášení do databáze a dovolí mu přihlásit se jako jiný uživatel.

29.8. Příkaz CHANGE PASSWORD

Tento příkaz dovolí přihlášenému uživateli změnit své heslo.

29.9 Rozhraní, které umožní uživatelům měnit svá hesla

Uživatelé chtějí často měnit svá hesla. Dává jim to pocit bezpečí a jistoty. Příkaz CHANGE PASSWORD pouze změní heslo. Samotné rozhraní pro případné potvrzování tohoto hesla musíte vytvořit samy. Vytvoříme jednoduchý systém, který se zeptá uživatele na jeho heslo dvakrát než jej skutečně změní. Bohužel neexistuje příkaz Číst heslo takže se uvnitř systému nemůžeme dotázat na staré heslo uživatele, protože je neznáme. Můžete vytvořit tabulku uživatelů, která by ukládala uživatele a jejich hesla, ale v tomto okamžiku se hesla stanou součástí dat a databáze bude méně bezpečná.

Náš nový systém nabídek dovolí rovněž Návrháři nebo Administrátorovi měnit systém hesel bez vstupu do Prostředí návrháře.

29.9.1 Vytvoření systému přístupu na heslo

1. Vytvořte novou metodu projektu M_GEN_ChangePassword následujícím způsobem:

```
If (False)
      ` Metoda: M_GEN_ChangePassword
      ` ACI Univerzita kurzy programování
      ` Vytvořeno: Jim Steinman
      ` Datum: 16/1/97

      ` Účel: Vytváří systému přístupu na heslo.

   <>fGeneric:=True
   <>f_Version6x20 :=True
   <>fJ_Steinman :=True

End if

      ` Declare local variables
C_STRING(31;$sUserName)                ` jméno připojeného uživatele
C_STRING(31;;$sPassword)             ` první vstup hesla
```

```
C_STRING(31;;$sPassword2)                ` druhý vstup hesla
C_BOOLEAN($fOK                          ` OK příznak
C_LONGINT($LLength)                      ` délka hesla
C_LONGINT($i)                           ` čítač smyčky

$sUserName:=Current user

If (($sUserName="Návrhář") | ($sUserName="Administrator"))
   EDIT ACCESS                  ` úparavy systému hesel

Else

   $sPassword:=Request("Vložte nové heslo!")              ` první vložení hesla
   If(OK = 1)
      $sPassword2:=Request("Zopakujte vaše nové heslo!")        ` zopakovat pro potvrzení

      If(OK = 1)
         $fOK:=True
         $LLength:=Length($sPassword)

         If ($LLength#Length($sPassword2))                     ` kontrola délky řetězců
            $fOK:=False
         Else

            For ($i;1;$LLength)
               If (Ascii($sPassword[[$i]])#Ascii($sPassword2[[$i]]))   ` kontrola znak po znaku
                  $fOK:=False
               End if
            End for

            If ($fOK)
               CHANGE PASSWORD($sPassword)
               ALERT("Heslo bylo změněno!")
            Else
               ALERT("Hesla, která jste vložili si neodpovídají!")
            End if

         End if
      End if
   End if
End if
  `Konec metody
```

2. Otevřete Záhlaví #1 a do nabídky Soubor přidejte následující položky:

Položka nabídky	Metoda
Paleta tabulek	M_PROCESS_TablesPalette
Kalendář…	M_Calendar
Změnit heslo…	M_GEN_ChangePassword
-	
Konec	M_GEN_Quit

3. Váš systém hesel je nyní připraven k použití.

29.10. Vymazání uživatele

Od 4D v6 můžete "vymazat" uživatele ze 4D. Je to způsob pseudo vymazání. Uživatel zůstává ve 4D proto, aby zůstala zachována jedinečnost ID, ale jestliže byl vymazán není již dále zobrazován v dialogu pro připojení či v okně CHANGE ACCESS. Uživatel je vidět v Editoru hesel. Vymazaní uživatelé jsou zobrazování v Editoru hesel zelenou barvou. Pokud byl uživatel vymazán nemůže být již nikdy obnoven. Takže uživatele mažte pouze velice opatrně. Na druhou stranu vám tato možnost poskytne způsob jak vymazat ty uživatele, které jste omylem vytvořili jako Návrhář.

29.10.1. DELETE USER(IDUživatele)

Příkaz DELETE USER vymaže uživatele jehož jedinečné číslo IDUživatele jste předali. Musíte předat platné IDUživatele vrácené pomocí příkazu GET USER LIST.

29.10.2. GET USER LIST (asJménaUživatele; asČíslaUživatele)

Příkaz GET USER LIST naplní array asJménaUživatele jmény uživatelů a array asČíslaUživatelů jedinečnými čísly IDUživatele tak jak se objevují v Editoru hesel.

Array je naplněno všemi uživateli, kteří jsou vidět v Editoru hesel včetně těch, kteří již byli vymazáni.

Poznámka: K zjištění zda byl uživatel vymazán použijte příkaz Is user deleted.

29.10.3. Is user deleted(IDUživatele) -> Logické

Tento příkaz vrátí hodnotu True nebo False podle toho jestli uživatel s daným ID byl či nebyl vymazán.

29. Použití sítě, které může ovlivnit chování

Následující akce mohou poměrně značně zatěžovat síť.

29.1. Tisk

Velké tiskové soubory přenášené po síti v době tisku můžou zatěžovat síť po dlouhý čas. Tisk souborů s obrázky (Word, Photoshop, Quark, Acrobat reader) zatěžuje síť obzvlášť.

29.2. Přenosy souborů

Přenosy souborů po síti z disku na disk rovněž způsobují velké zatížení sítě.

29.3. Elektronická pošta

Elektronická pošta je často hlavním účelem sítě.

Pokud máte 50 uživatelů a všem pošlete zprávu a jako odpověď všichni pošlou opět zprávu všem může to vyvolat značný zatížení sítě. Uživatelé musí být vyškolení v tom jak správně používat e-mail.

29.4. Přístup k síťovým zařízením

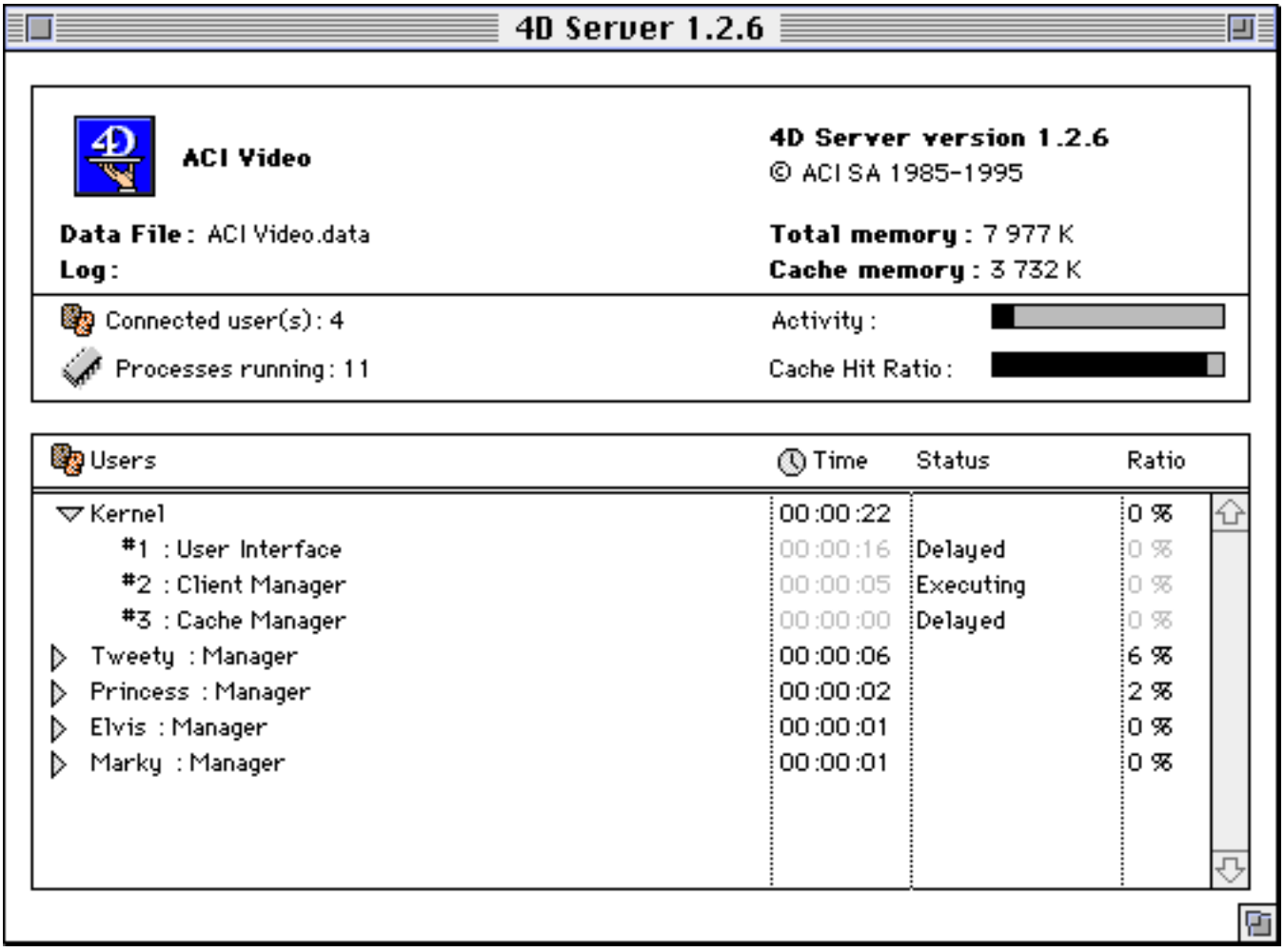
Síťové modemy, zařízení pro ukládání dat atd. mohou způsobovat zatížení sítě.

29.5. Nesprávně ukončené uzly nebo poškozené kabely

Nejvážnější ovlivnění chování sítě jsou obvykle fyzické chyby ve vedení a propojení kabelů. Interference ze silných elektrických nebo magnetických zařízení může síť i zcela přerušit. Neukončené uzly nebo poškozené kabely mohou síť velice zpomalit až téměř k neprůchodnosti Tyto vady způsobují, že se packety po síti několikanásobně odrážejí a putují sítí tam a zpět.

30. Okno 4D Server

Okno 4D Server zobrazuje všechny procesy včetně procesů, které jsou spuštěny pouze na serveru.



30.1. User Interface

Tento proces ovládá samotné okno 4D Server a nabídky. Toto je proces, který je zodpovědný za to, že vám sdělí, kterou databázi a datový soubor používáte kolik uživatelů je připojeno, jaké používají globální procesy atd.

30.2. Cache Manager

Tento proces ovládá uvolňování vyrovnávací paměti 4D na disk. Ovládá ukládání nových a upravených záznamů na disk. Tato operace je kompletně asynchronní a nemá vliv na rychlost dalších operací prováděných v databázi (za předpokladu, že je pevný disk rovněž asynchronní - SCSI). Rovněž ošetřuje pořadí dat v paměti požadovaných uživatelem.

30.3. Client Manager

Tento proces připojuje a odpojuje uživatele serveru. Ovládá předávání zdrojů na klienty, když se připojují poprvé a pokud jsou tyto zdroje změněny. Rovněž ovládá předávání informací na klienty jako odpověď na žádost klienta o obnovu formulářů a metod.

30.4. Indexing

Tento prces ovládá indexování polí. Tato operace je kompletně asynchronní a brání ostatním operacím v provádění dotazů, modifikaci dat na základě nového stavu dokud běží indexování.

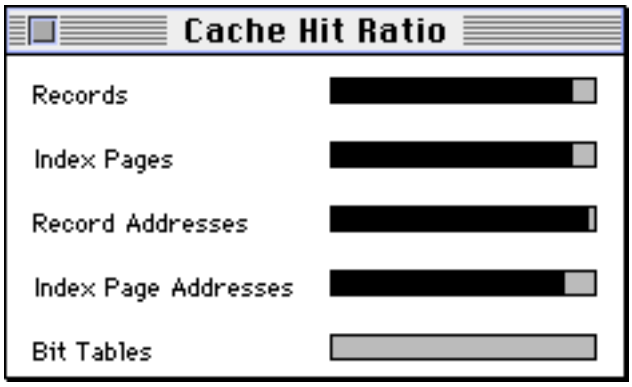
30.5. Cache/Hit Ratio

Tento teploměr vám říká kolikrát jste obdrželi data nebo prvky struktury z paměti než přímo z disku. Na rozdíl od jiných teploměrů zde platí čím více tím lépe. Jestliže je tento teploměr nad 75% znamená to, že vaše databáze se chová normálně v přijatelných mezích. Pod 50% znamená, že určitě potřebujete přidělit více RAM pro vyrovnávání dat do paměti serveru.

Použití tohoto teploměru pro hodnocení chování databáze má význam při dlouhodobém spuštění a používání přibližně alespoň týden. Více než jeden týden je vhodné, protože čím delší období hodnotíme tíme je naše statistika přesnější.

30.6. Okno Cache/Hit Ratio

Toto okno nám dává více informací než pouhý teploměr v základní obrazovce. Dozvíme se zde jaké informace přicházejí z paměti a jaké z disku.



Ideální stav nastane, když se vaše databáze blíží stavu uvedeném v následující tabulce:

Records - 75%
Index Pages - 100%
Record Addresses - 100%
Index Page Addresses - 100%
Bit tables - liší se

Databáze ukládají data buď v záznamech o pevné délce nebo proměnné délce. 4D není ani jedním z těchto případů. 4D ukládá data v kombinaci těchto dvou metod. Ve 4D jsou data ukládána na disk v blocích po 128 bytech. Existují bitové tabulky, které říkají, který blok je používán. Výhoda záznamů o pevné délce je, že přístup k datům je velice rychlý. Na druhou stranu existuje mnoho volného místa uvnitř dat a data nejsou zrovna ideálně ukládána. Záznamy o proměnné délce jsou dobré pro ukládání, ale obtížně se vybavují. Takže bitová tabulka je kombinací bloků pevné délky a záznamů o proměnné délce.

31. Použití paměti na serveru

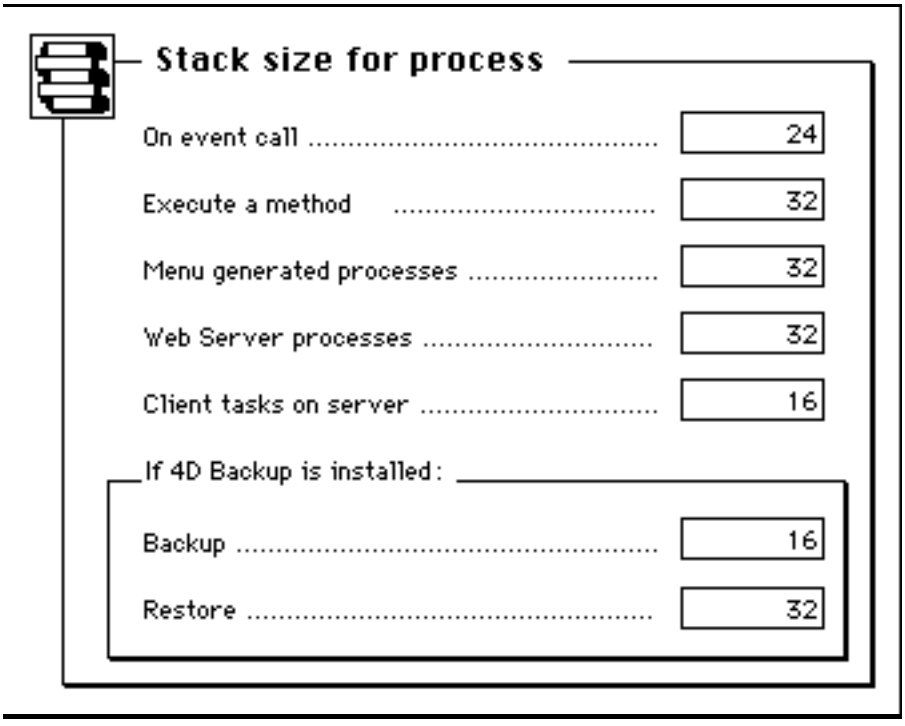
31.1. Uživatelé, procesy a výběry

50k na uživatele

Minimum 50k (na serveru) je požadováno k připojení uživatele. (to je část paměti přidělená aplikaci 4D Server).

4k na proces

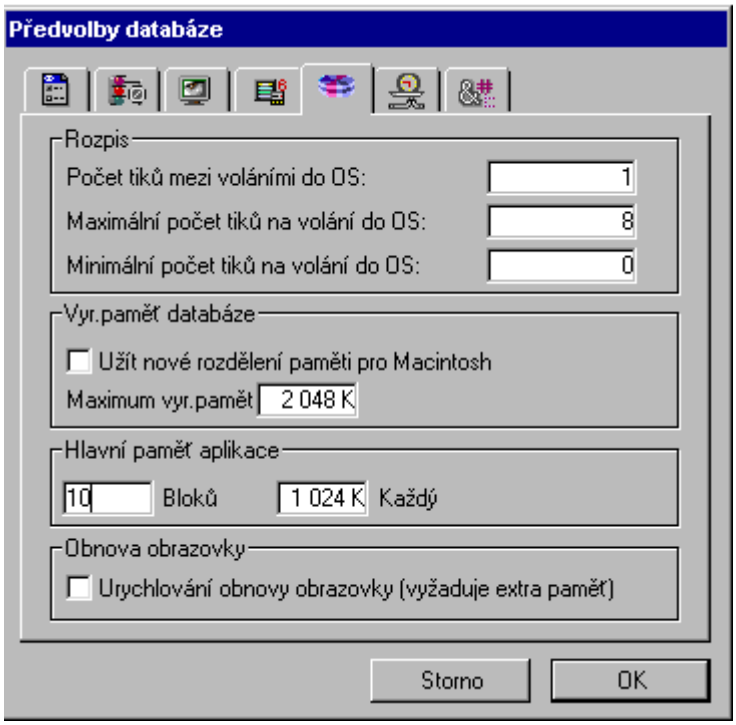
Každý globální proces vytvořený na klientu má svůj protějšek na 4D Serveru. Tento proces má výchozí hodnotu stack 16k. Jestliže plánujete triggery tato minimální velikost stack se zvýší na 32k. Jestliže plánujete caskádově spouštěné triggery, budete pravděpodobně potřebovat větší stack než 32k. Zvětšete toto nastavení pomocí Customizer Plus.



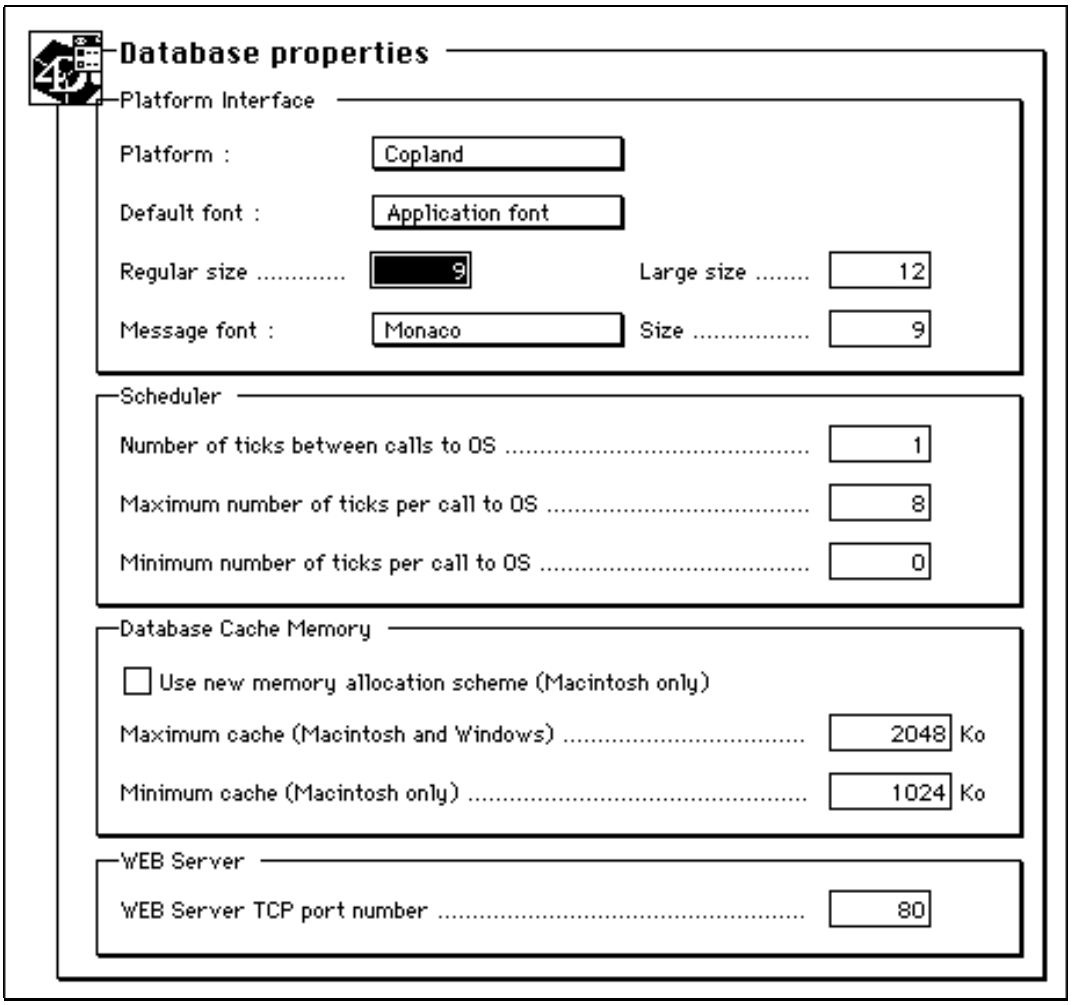
4 byty na záznam ve výběru

Každý z těchto sesterských procesů používá paměť pro uložení platného výběru tabulky a libovolného pojmenovaného výběru. Výběry potřebují 4 byty na záznam ve výběru.

31.2. Nastavení předvoleb databáze, keré má vliv na paměť serveu a chování



Okno Vyladění v Předvolbách databáze



Okno Předvoleb databáze v Customizer Plus

31.2.1. Scheduler (Řízení volání)

- Počet tiků mezi voláními OS (Number of ticks between calls to OS). Tato hodnota vám dovolí upravit počet tiků mezi voláními 4th Dimension do operačního systému (Windows95,8 nebo MacOS). Čím vyšší je toto číslo, tím méně času bude dáno ostatním funkcím počítače.

- Maximální počet tiků na volání do OS (Maximum number of ticks per to call to OS) (Pouze Windows NT) Tato hodnota má vliv na množství času, který je 4D ochotna čekat, dokud systém opět nezavolá 4D. Čím vyšší je toto číslo tím pomaleji 4D poběží.

- Minimální počet tiků na volání do OS (Minimum number of ticks per call to OS) (Pouze Windows NT) Tato hodnota má vliv na minimální počet tiků po kterých systém zavolá 4D i když má volný čas. Čím vyšší je toto číslo tím pomaleji 4D poběží.

31.2.2. Vyrovnávací paměť databáze

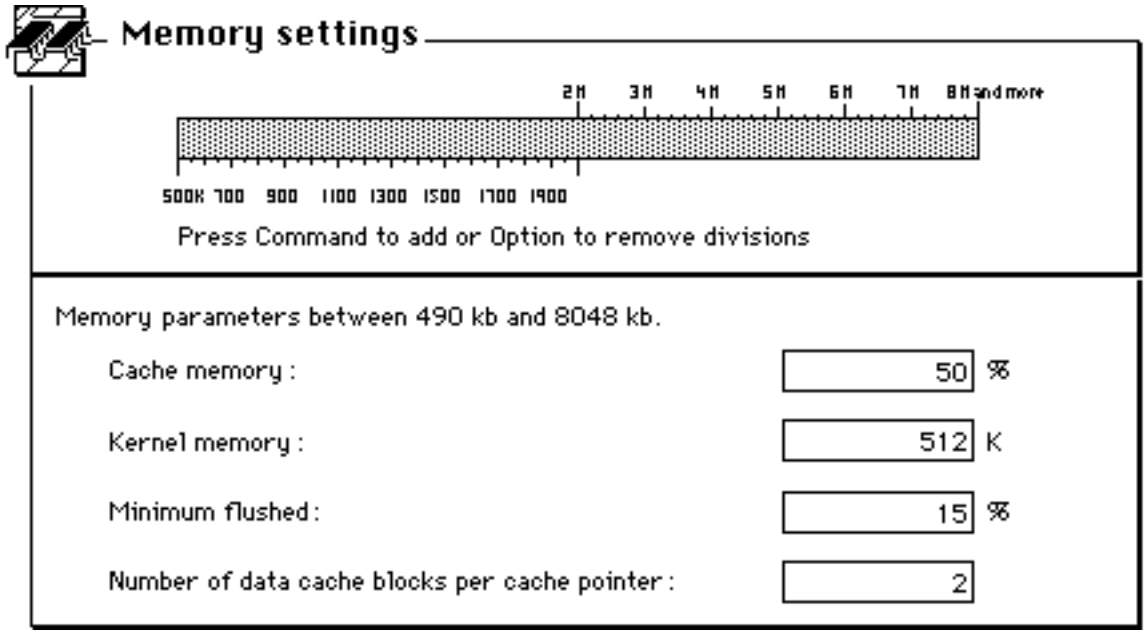
Vyrovnávací paměť (Cache) je část paměti, kde 4D ukládá zázanmy, tabulky adres záznamů a indexy. Čím více paměti je přiděleno pro vyrovnávací paměť tím více dat může 4D Server držet v paměti a tím méně často je musí zavádět z disku.

Nový systém přidělování paměti (na Mac) je políčko zatržení, které dovolí vzít pamě't z MultiFinderu systému Macintosh místo paměti přidělení samotné aplikaci (Inforace). Paměť přidělená 4D bude použita pouze tehdy jestliže není dost volné paměti na úrovni Finder.

31.2.3. 4D hlavní paměť(Main Memory- pouze Windows)

Tato oblast dovolí definovat jak mnoho paměti přidělit 4thDimension pro databázi spuštěnou na Windows. Tato možnost byla dříve pouze pomocí Customizer Plus.

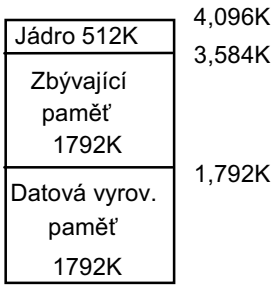
31.2.4. Tradiční nastavení paměti pro Macintosh



Nastavení Customizer Plus (Staré schéma)

31.3. Halda aplikace 4DServeru

Paměť halda aplikace 4D Server (Application Heap) je rozdělena do dvou oblastí, vyrovnávací paměť (cache) pro data a zbývající paměť



Když se zavádí prvních 512K jádro zavádí samo sebe.

Výchozí nastavení je, že zbytek paměti je rozdělen na 50% pro vyrovnávací paměť a 50% pro zbytek.

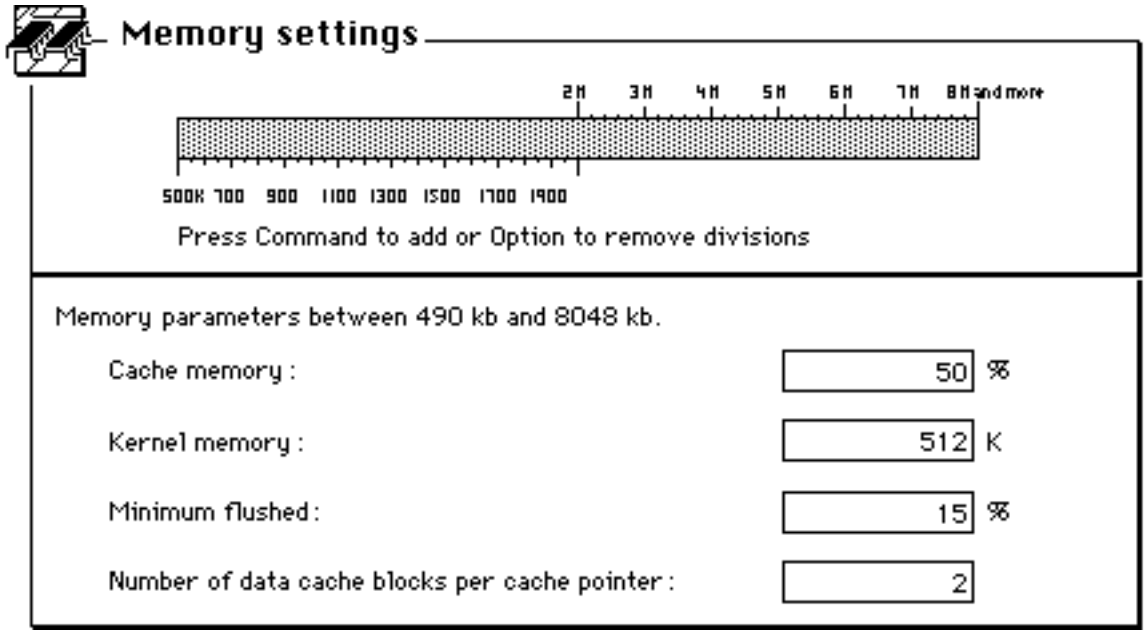
Zbytek paměti je používán k ukládání formulářů, nabídek, array a dalších komponent struktury. Prvky struktury jsou nejdříve načteny do paměti serveru a pak předávány klientu na jeho žádost. Například formulář, jestliže je jeho metoda objektu vybrána jen pro určitou událost (Např. Při aktualizaci) je zavedena do paměti serveru, když je poprvé použita, rovněž metoda formuláře. Metoda projektu přiřazená položce nadídky je zavedena do paměti, když je poprvé použita na klientu (když je vybrána položka nabídky). Tyto objekty jsou na serveru v paměti v zkomprimovaném tvaru. Na klinta jsou zaslány v tomto tvaru a kopie je rozbalena v paměti vykonána a pak zapsána na lokální disk. Po tom co klient použil daný prvek struktury poprvé je dále přebírán z paměti nebo lokálního disku klienta. Do paměti serveru bude opět zavedena a zaslána klientu, pokud se změní od jejího posledního užití klientem.

Poznámka.: Předchozí část platí především pro nezkompilované aplikace, kde se dá očekávat vývoj za běhu aplikace. U zkompilovaných databází jsou všechny zdroje (tj. i prvky struktury) přebrány v době prvního spuštění aplikace a na klientu pak čteny z lokálního disku.

Zbývající paměť je oblast, kde jsou ovládány všechny procesy. Ukládá se zde platný výběr na tabulku, proces a uživatele, ukazatel na platný záznam a rovněž pojmenované výběry. Procesy používané samotným 4D Serverem jsou zde rovněž umístěny.

Vyrovnávací paměť je část paměti kde 4D ukládá záznamy, tabulky adres záznamů a indexy. Čím více paměti je přiděleno pro vyrovnávací pamět tím více záznamů, tabulek

adres záznamů a indexů bude 4D Server držet v paměti a tím méně často se bude obracet na disk a zavádět je z disku.



Použitím Customizer Plus můžete určit jaké procento RAM 4D Server bude přiděleno pro vyrovnávací paměť a kolik zbyde. Kolik paměti přidělit záleží na chování databáze. Pokud je vaše aplikace především struktura a každý klient přistupuje často k různým prvkům struktury a každý klient potřebuje jen velice málo záznamů asi přidělíte více procent zbytku paměti. Jestliže vaše aplikace jsou především záznamy a máte malou strukturu asi přidělíte více paměti vyrovnávací paměti. Většina databází je především orientována na záznamy. Proto je pro vyrovnávací paměť přidělováno od 50% výše. Pokud máte k dispozici více než 5MB můžete pro vyrovnávací paměť přidělit až 75%. Tato čísla jsou pouze orientační konkrétní nastavení musí být vyladěno podle chování vaší databáze. Jestliže používáte Windows nastavujte zvlášť vyrovnávací paměť a paměť pro jádro. Paměť pro jádro (kernel) na Windows slučuje obě předchozí oblasti v paměti - jádro a zbytek dohromady.

31.4. To že můžete ještě neznamená, že máte

Většina uživatelů pokud poprvé slyší o rozdělení paměti a využití paměti se okamžitě snaží měnit nastavení. Není to však vždy dobré.

Paměť je drahá. Nejjednodušším řešením je vybavit váš počítač serveru dostatkem paměti a zvýšit především část vyrovnávací paměti i část jádra tak, aby jste zajistili, že 4D Server může do paměti vyrovnat tolik ze struktury i dat kolik je možné.

31.5. Vyrovnávací paměť systému

Na počítači Macintosh lze nastavit vyrovnávací paměť systému, měla by být nastavena mezi 64K a 128K. Jestliže je nastavena vyšší vyrovnávání i uvolňování paměti systému bude trvat příliš dlouho a běh 4D Server bude dočasně přerušován. Na druhou stranu, jestliže snížíte systémovou vyrovnávací paměť na méně než 64K způsobí to velice časté uvolňování této paměti a 4D Server bude přerušován častěji, to pak bude rovněž mít záporný efekt na běh serveru.

Swap soubory na Windows. Na Windows95 a WindowsNT nedoporučujeme používat Swap soubory. Jestliže je používáte na Windows 3.1. musí být tyto soubory trvalé a ne dočasné.

31.6. Velikost Stack pro procesy klientů na serveru

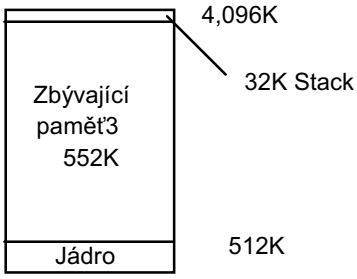
Výchozí velikost Stack je pro procesy klientů na serveru 16K. Jestliže používáte triggery měli by jste ji zvýšit na 32K. Jestliže používáte kaskádovitě spouštěné triggery můžete potřebovat dokonce ještě větší stack.

31.7. Sady

Od 4D v6 jsou sady udržovány na serveru. Počet záznamů na tabulku má vliv na velikost sady (1 bit na záznam). Vymazané záznamy jsou rovněž zahrnuty v sadě (1 bit na vymazaný záznam). Jestliže jsou vaše tabulky velké pak vytváření sad procesů může spotřebovat mnoho paměti. Např. sada vytvořená z tabulky o 500 000 záznamů bude vyžadovat 62,5 K (500 000/8 bitů na byt = 62 500 bytů).

32. Použití paměti na klientu

4D Client má výchozí část paměti 1500K. Tato část se skládá pouze z haldy aplikace, jádra a paměti stack Uživatel/Aplikace. Do paměti klienta nejsou vyrovnávána žádná data proto klient nemá vyrovnávací paměť.



Halda aplikace klienta obsahuje segmenty kódu 4D Client, prvky struktury (nabídky, metody, formuláře atd.) a platný záznam. Array jsou vyrovnávány do haldy aplikace klienta, protože jsou potenciálně používány ve formulářích a 4D Client musí určit zda uživatel nepracuje se zobrazeným array. Platný záznam je držen v haldě aplikace klienta, protože je potenciálně zobrazován na formuláři a musí být dovoleno, aby uživatel měnil hodnoty v polích tohoto záznamu.

Čím více prvků struktury a procesů je užíváno klientem tím více paměti je potřeba pro optimální chování. Na počítači Macintosh ji přidělíte v Informacích. Jestliže nejsou určité objekty v paměti zavedeny, 4D Client bude vyvádět objekty z paměti, aby si udělal místo pro další objekty požadované uživatelem. V tomto případě je potřeba dát 4D Client větší paměť.

32.1. Procesy

Následující množství paměti je požadováno pro každý proces na klientu:

- 32K k ovládání procesu
- 16K až 32k pro stack procesu

Stack udržuje "strom závislostí". Strom závislostí je seznam prvků, které jsou používány procesem. Např. název hlavní metody procesu je umístěn v stack. Názvy podmetod volaných hlavní metodou jsou umístěny v stack. Názvy formulářů používané v procesu jsou umístěny v stack. Předpokládejme, že začnete provádět hlavní metodu procesu, název této metody přijde do stack. V této metodě voláte příkaz MODIFY SELECTION, který použije výstupní formulář a metodu výstupního formuláře. Název formuláře přijde do stack. Z výstupního formuláře poklepáním na záznam přejdete do vstupního formuláře

a jeho metody. Název vstupního formuláře přejde do stack. Metoda vstupního formuláře používá lokální proměnné. Názvy lokálních proměnných a jejich stav přijdou do stack. Nyní poklepete na záznam ve vložené oblasti a přejdete tak do jiného formuláře pro vztaženou tabulku. Dále uzavřete tento formulář a přejděte tak zpět do původního vstupního formuláře. Jak 4D ví do kterého formuláře se vrátit a jaký byl obsah lokálních proměnných pro tento formulář? 4D se podívá do stack pro daný proces a vidí formulář a stav lokálních proměnných, což jí umožní přejít zpět do tohoto formuláře a použít lokální proměnné v tom stavu v jakém jsme je opustili. Když uzavřete vstupní formulář navracíte se do výstupního formuláře uvedeného ve stack. Stack pracuje metodou "poslední dovnitř/první ven". To umožňuje 4D vědět, kde byla když se pohybujete aplikací takže vám umožňuje se navráti zpět odkud jste přišli. Stack je seznam prvků. Pouze názvy prvků (formuláře, metody, podmetody atd.) přicházejí do stack a ne skutečný obsah formuláře či skutečný řádek kódu z podmetody. Jedinou vyjímkou jsou lokální proměnné jejichž obsah přichází do stack

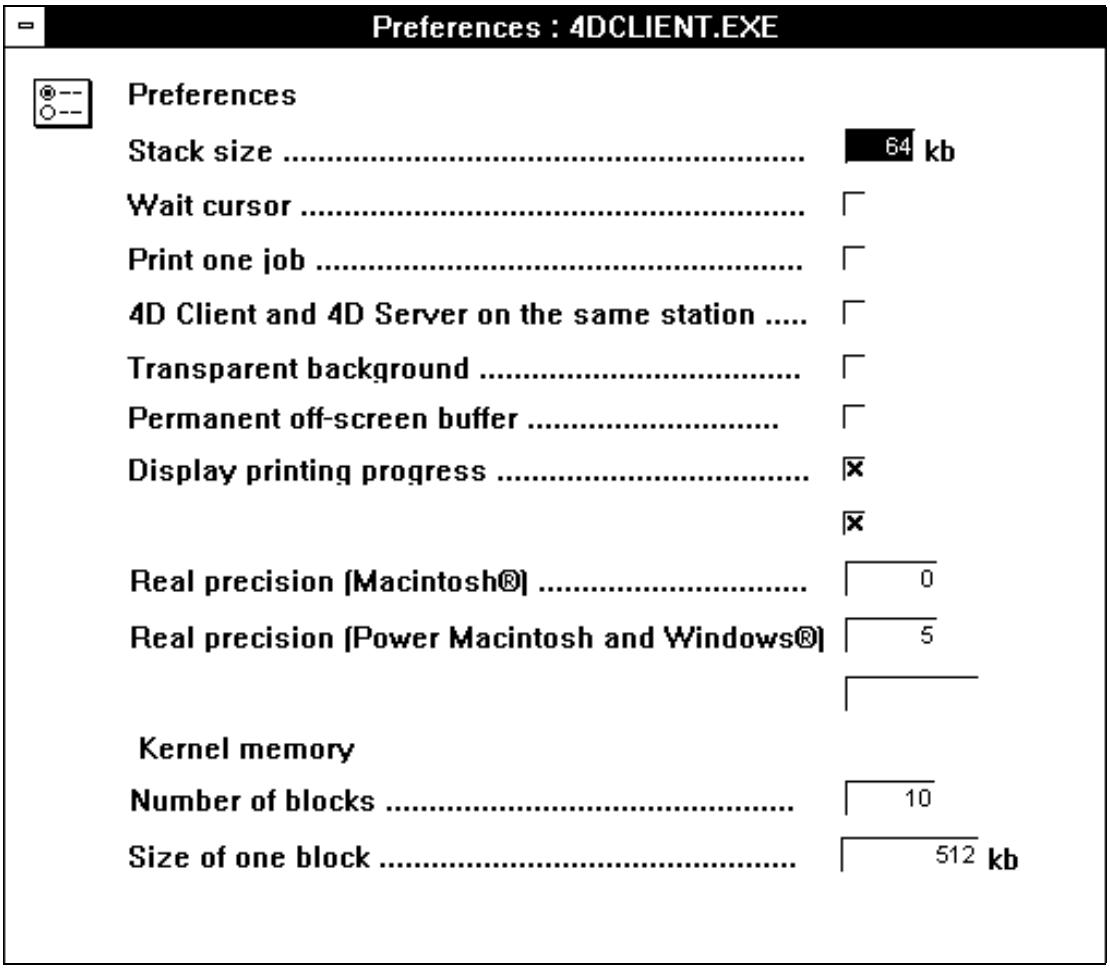
- 1K do ? pro proměnné procesu

32.2. Prvky struktury: Nabídky, Metody, Formuláře, Metody formulářů, objektů, Seznamy atd.

Poprvé, když použijete formulář, metodu, nabídku atd. je zavedena ve zkomprimované formě z paměti na serveru do haldy aplikace na klientu, pak je zapsána na disk na klientu do souboru .rex. Zkomprimovaná forma objektu je vyrovnána do paměti na klientu a rozbalená kopie objektu je v paměti použita procesem. Jestliže tentýž objekt (např. formulář) je použit dvěma různými procesy na stejném klientu. Je v paměti přítomna jedna zkomprimovaná kopie objektu a dvě rozbalené kopie objektu. Když skončíte s používáním prvku struktury a jestliže je proces požaduje opět, 4D Client zkontroluje jestli se objekt na serveru změnil. Jestliže se prvek změnil je zaslána ze serveru nová verze do paměti klienta a zapsána do souboru .rex. Jestliže se prvek nezměnil 4D Client jej nejdříve hledá v paměti a pak pokud byl z paměti vyveden jej zavede z pevného disku do paměti.

Jestliže je množství přidělené paměti pro 4D Client příliš malé, nebude 4D Client schopen držet všechny prvky struktury požadované uživatelem v paměti. Následně proto budou některé z těchto prvků struktury vyváděny z paměti a jiné zaváděny. To bude mít vliv na to jak rychle se formuláře budou zobrazovat na obrazovku a jak rychle se začnou položky nabídky a tlačítka vykonávat po klepnutí.

Poznámka!!!: Kompilovaná databáze zvýší účinek tohoto procesu. Struktura jako celek má otisk datumu a času. Klient obdrží kopii struktury s otiskem datumu a času a porovná tento otisk s kopií uloženou na disku. Jestliže je stejný používá se lokální kopie, jestliže je otisk odlišný server požádá o nové kopie a nepotřebuje se znovu a znovu ptát na další kopie dokud není struktura znovu zkompilována. .



32.3. Nastavení použití paměti

Nastavení paměti 4D Client na Windows je zcela odlišné od nastavení paměti 4D Client na Macintosh. Na Windows musí být paměť nastavena pomocí 4D Customizer Plus. Ke zvýšení paměti přidělené 4D Client nastavte vyšší počet bloků v Kernel memory.

32.4. Poznámky k souboru .rex

V předchozích částech jsme popsali způsob jakým 4D Client používá prvky struktury a dalších zdrojů tak, aby zatěžoval 4D Server co nejméně. Zde je nutno především mít na paměti, že používání těchto prvků je odlišné v kompilované databázi a v databázi, která běží v interpretačním módu. Ve všech případech se 4D Client obrací po prvním překopírování souborů struktury především na své lokální kopie na disku. Pokud vzniknou nějaké problémy s chováním jednoho klienta oproti jiným může to někdy být způsobeno poškozenou kopií souboru struktury na lokálním disku. V těchto případech je

nejlepší otevřít složku ACI v systémové složce a vymazat soubory databáze.rex, databáze.res a složku s názvem databáze. Při dalším spuštění 4D Client a přihlášení se k databázi 4D Client převezme ze 4D Server nové kopie souborů struktury a modulů.

Poznámka: Tento první krok 4D Client při přihlášení k databázi na serveru je ekvivalentní instalaci aplikace klienta u jiných řešení klient/server. Porovnáte-li složitost těchto instalací pochopíte i oblíbenost řešení pod 4D u mnoha administrátorů sítí.

32.5. Rychlejší překreslování obrazovky

Zaškrtnete-li v předvolbách databáze políčko Rychlejší překreslování obrazovky vyžaduje extra paměť, budou při používání formulářů vytvářeny speciální bit mapy vašich obrazovek k urychlení překreslování. Množství dodatečné paměti závisí na velikosti vašeho monitoru (počtu bodů) a na hloubce bodů (počtu barev). Výraz pro výpočet velikost bit mapy je:

Velikost v KB=(Šířka obrazovky x Výška obrazovky x Hloubka obrazovky) / 8 / 1024.

33. Jiné produkty 4D používané se 4D Server

33.1. 4D Backup

4D Backup je aplikace, která může provádět následující:

Úplnné zálohování souborů databáze (data, struktura atd.) do několika souborů řízených aplikací. Tato operace může být provedena dokonce v době používání databáze.

Obnova databáze ze zálohy.

Vytváření log souboru, který obsahuje všechny změny provedené v databázi od poslední úplnné zálohy. Tento soubor může být použit programem 4D Backup k obnově databáze, k určité době tak, že je emulováno od poslední úplnné zálohy postupné pořizování dat a všechny další změny až do určené doby. Při provádění této obnovy mohou být sledovány operace typu: přidávání, mazání, úpravy záznamů.

Automatická údržba zrcadlového datového souboru na odděleném disku.

Operace mohou být řízeny nastavením aplikace nebo pomocí metod programovaných ve 4D.

33.2. 4D Open

Zásuvný modul oddělených metod a příkazů, který umožňuje jiným aplikacím než 4D Client fungovat jako klient ke 4D Server

4D Open je knihovna příkazů, která může být přiřazena buď k jiným aplikacím 4D a nebo k aplikacím od jiných výrobců umožňujících psaní maker.

- 4D Open for 4D
- 4D Open (C;C++;Pascal)

Na co lze použít 4D Open?

- Připojit se k databázi jako klient a odpojit se
- Dotazovat se a třídit
- Vytvářet, upravovat a mazat záznamy

4D Open nedovoluje spouštět metody 4D nebo otevírat formuláře 4D, ale dovoluje přístup k záznamům.

33.3. Moduly produktivity

Moduly produktivity jsou zásuvné moduly pro ukládání nestrukturovaných dat v jednom poli záznamů:

- 4D Write
- 4D Calc
- 4D Draw

33.4. Moduly připojení

Moduly připojení jsou zásuvné moduly, které umožňují propojit 4D Server s jinými zdroji dat klient/server založenými na technologii SQL. Spojení je umožněno k:

Oracle

Sybase

ODBC

MS SQL

Umožňují propojení k několika serverům SQL najednou.

34. Použití našich vlastních nástrojů k rozvíjení aplikace

Když přidáme novou tabulku do naší databáze je několik věcí, které musíme udělat, abychom využili navržené generické metody a rychle zahrnuli tuto tabulku do programu.

1. Vytvořte tabulku ve struktuře.
2. Vytvořte všechny potřebné vztahy.
3. Vytvořte formuláře pro vstup a výstup nazvané Vstupní a Výstupní. (buď přejděte do Prostředí uživatele a zpět do Návrháře nebo vytvořte každý formulář ručně s použitím připravených vzorů.)
4. Prověřte, že se tyto formuláře nazývají Vstupní a Výstupní.
5. Vytvořte metodu formuláře pro vstupní formulář, která volá WIN_InputWindowTitle při události Při zavedení.
6. Přiřaďte záhlaví # -2 k formulářům vstupní i výstupní.
7. Vytvořte další formuláře pro použití v položkách Záznamy ¬ Import a Záznamy ¬ Export.
8. Pojmenujte formulář pro Import/Export jako Import/Export.
9. Vytvořte vstupní formulář pro užití se záznamy Záznamy ¬ Dotaz dle příkladu.
10. Pojmenujte tento formulář Dotazy.
11. Vytvořte vstupní formulář pro užití s Zprávy ¬ Speciální zprávy (pojmenujte formulář TiskVýstup).
12. Proveďte vše co je potřeba pro přiřazení výchozích hodnot pro nové záznamy.
13. Proveďte vše co je potřeba pro přiřazení jednoznačné identifikace.
14. Proveďte vše co je potřeba pro přiřazení hodnot polí k vztažení nových záznamů rodičovským záznamům.
15. Jestliže používáte palety se seznamy oken zahrňte patřičný kód do všech formulářů.
16. Otevřete libovolné záhlaví nabídek a nabídku Soubor, vložte položku pro novou tabulku.
17. Pro novou položku nabídky přiřaďte metodu M_Tabulka.
18. Vytvořte novou metodu M_Tabulka následovně:

```
If (False)
    ` Metoda: M_Tabulka
    ` ACI Univerzita kurzy programování
    ` Vytvořeno: YourNameGoesHere
    ` Datum: DateCreatedGoesHere

    ` Účel: Nastaví  pTable na [Tabulka] pro použití všech metodách
    ` a voláních výstupního formuláře.

End if

Repeat
```

```
   INITIALIZE_Process (->[Tabulka])
   LOCK_Tables2ReadWrite (pTable)
         ` viz níže

   GEN_ModifySelection

   GEN_ClearSelection(pTable)

   fProcessAvailable := True
   PAUSE PROCESS(Current process)
Until (<>fQuit)

   ` Konec metody
```

19. Před voláním GEN_ModifySelection vložte do výše uvedeného kódu kód k nastavení hlavní tabulky READ WRITE a nastavení READ WRITE pro ostatní tabulky, ve kterých bude prováděn vstup pomocí vložených formulářů ze vstupního formuláře. Např.:

```
LOCK_Tables2ReadWrite (pTable; ->[VložTabulka1]; ->[ VložTabulka2])
```

35. Závěr

Pokusili jsme se vám ukázat v průběhu tohoto kurzu mnoho různých způsobů programování. Vše co jsme zde probrali by rozhodně nemělo být považováno za jediný nebo dokonce nejlepší způsob pro splnění patřičných úloh. Každá databáze je jiná a je potřeba ji vyladit jiným způsobem.

Příkazy a funkce probrané v tomto kurzu mají mnoho dalších použití nejen ty, které jsme zde probrali. Vyzkoušejte si je samy. Proberte si ještě jednou kód zde použitý a vyzkoušejte si do něho vlastní zásahy.

Nejdůležitější, ale je: Nebojte se zkusit vše co vás napadne, pouze chybami se člověk učí.

Příloha A: Optimalizace Windows NT pro zlepšení chování 4D Server

Úvod

4D Server běžící pod Windows NT Workstation nebo Windows NT Server je velmi účinná kombinace. Optimalizace Windows NT je však poněkud složitější než na Mac. Tato příloha obsahuje způsoby, které vám pomohou optimalizovat 4D Server pro Windows NT. Tyto způsoby zahrnují diskusi na následující témata:

- Použití systému souborů NT
- Optimalizace Pagefile Size Settings
- Komprese složky obsahující datový soubor 4D
- Spuštění 4D Server z batch souboru v REALTIME mode
- Konfigurace zobrazení
- Konfigurace plochy (desktop)

Upozornění

Za prvé, způsoby diskutované v této příloze jsou vytvořeny na základě doporučení Microsoft a provedeny a testovány ACI US. Výsledky na vašem konkrétním zařízení mohou být však odlišné.

Za druhé, všechny změny do systému databáze klient/server by pečlivě otestovány. Otestujte každý z těchto způsobů na vašem vlastním zařízení předtím než je do svého řešení zahrnete na trvalo.

Použití systému souborů NT (NTFS)

Systém NTFS je rychlejší a spolehlivější než FAT nebo HPFS. NTFS rovněž podporuje do NT vestavěný model bezpečnosti, který vám poskytuje více možností zakázat přístup souborům na NT Serveru.

Jsou zcela zřejmé věci jako RAM, rychlost procesoru, ale jedna z velice důležitých možností vylepšení, kterou můžete provést téměř ihned je spuštění 4D Server z NTFS disku. Tj. umístění souborů struktury a dat na NTFS.

Optimalizace Pagefile Size Settings

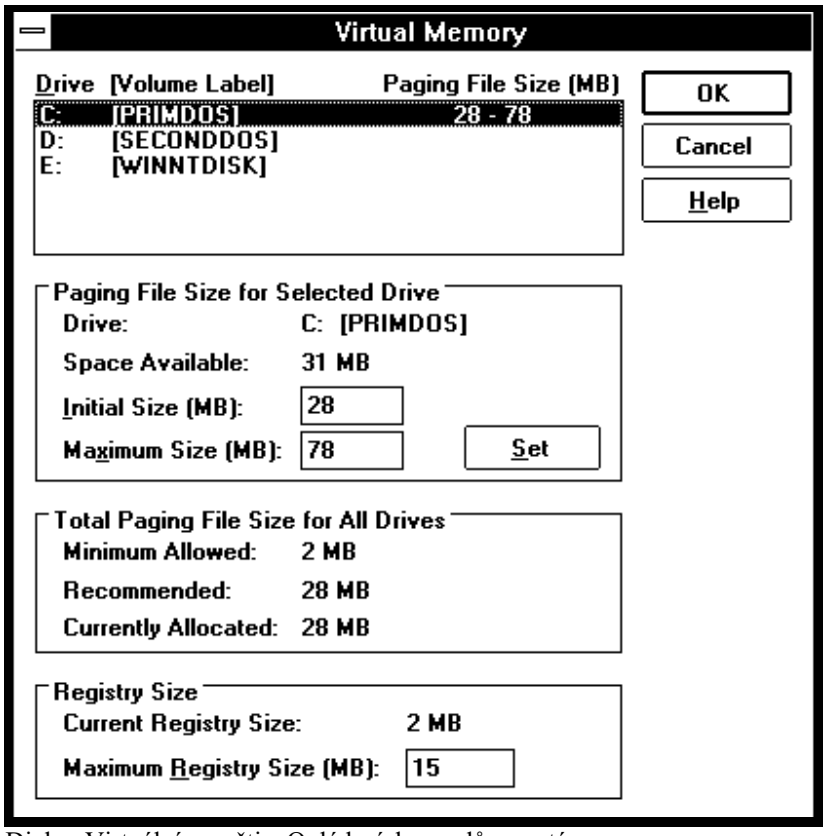
Soubor stránek (Pagefile) je průběžný blok místa na disku používaný pro virtuální paměť a jiné systémové funkce. Příležitostně NT zapisuje procesy na disk do tohoto souboru pokud má nedostatek paměti. Rovněž NT zapisuje do tohoto souboru tiskové úlohy.

Protože Pagefile je jeden z nejčastěji používaných souborů na stroji NT, má nastavení tohoto souboru podstatný vliv na chování počítače.

Nastavení Pagefile může být nalezeno v:

Control Panel -> System -> Virtual Memory

Následující obrázek je obrazovka dialogu virtuální paměti z Ovládacích panelů v systému:



Dialog Virtuální paměti z Ovládacích panelů v systému

Pamatujte si, že musíte být přihlášení jako administrátor, aby jste mohli provádět změny v ovládacích panelech.

Optimální nastavení velikosti Pagefile je následující:

optimální velikost = velikost fyzické paměti + 12 MB

Např. předpokládejme, že máte fyzickou velikost RAM 24 MB:

optimální velikost = 24 MB + 12 MB = 36 MB

Jestliže máte fyzicky více pevných disků (ne jenom dělených) je doporučeno vytvořit pagefiel na každém fyzickém disku. Optimální velikost každého souboru pagefile na každém disku je:

pagefile na disk = optimální velikost / (počet disků - 1)

Např. předpokládejme, že máte fyzicky 3 pevné disky a 24 MB RAM:

pagefile na disk = 36 MB / (3 - 1) = 18 MB

Z tohoto pravidla existuje jedna vyjímka. Jestliže máte několik disků, soubor pagefile by neměl být umístěn na stejném disku se systémovou složkou WINNT. Proto se dělí počtem disků -1. Důvod je zřejmý. Tím, že máte Windows NT na odděleném disku může být hlava disku umístěna nad systémovými soubory po většinu času. Protože soubor pagefile je používán velice často pak je-li na stejném disku jako Windows NT musí hlava disku udělat mnohem víc pohybů, které zhorší chování.

Proto jestliže máte pouze jeden disk, vytvořte pouze jeden pagefile. Na druhou stranu jestliže máte více disků, rozdělte pagefile na všechny disky kromě disku obsahujícího systém.

Mimochodem jestliže máte na počítači pouze jeden disk můžete dosáhnout poměrně značného zlepšení přidáním jiného disku do vašeho systému a umístěním souboru pagefile na tento nový disk.

Komprese složky obsahující datový soubor 4D

Jinou technikou urychlení 4D Server je umístění datového souboru do komprimované složky. Na NTFS disku můžete komprimovat libovolnou složku nebo soubor z File Manager výběrem Compress z nabídky soubor. Poznámka: to je druhý dobrý důvod k umístění 4D na NTFS disk.

Výhodou tohoto způsobu je, že se datový soubor obvykle zkomprimuje na polovinu své velikosti. A protože čas pro rozbalení dat je mnohem kratší než čas, který zabere přečtení dat z disku dosáhneme poměrně značného zlepšení. Vzpomeňte si, že pevný disk je vždy nejpomalejší složka vašeho systému. Takže všechno co sníží dobu přístupu k disku zvýší celkové chování systému.

Poznámka: Pokud použijete tuto techniku doporučuje se zvýšit rychlost procesoru o 12 MHz a přidat dodatečných 8 MB RAM k vyladění systému při dekompresi souboru.

Spuštění 4D Server z batch souboru v REALTIME Mode

V REALTIME by měla být nastavena pro 4D Server priorita. Efekt tohoto nastavení je, že se zvýší počet cyklů, které budou přiděleny 4D Server. Jestliže jen jednoduše spustíte aplikaci NT předpokládají, že aplikace nemá větší požadavky na systém než jakákoliv jiná další běžící aplikace. Nastavením priority v REALTIME dáte NT vědět, že 4D Server má větší požadavky na dělbu CPU i než systém sám či jakákoliv jiná běžící aplikace.

Zde je soubor batch, který spustí 4D Server v módu REALTIME. Tento batch soubor by měl být umístěn v téže složce jako 4DSERVER.EXE:

```
start /REALTIME 4DSERVER.EXE
exit
```

Můžete volat batch soubor z příkazového řádku MS/DOS nebo použitím příkazu Run z nabídky Soubor v Program Manager.

Pro optimální chování by při běhu 4D Server neměla být spuštěna žádná další aplikace. Je zřejmé, že všechny další běžící aplikace se budou dělit o CPU a budou bránit 4D Server v přístupu k CPU.

Konfigurace obrazovky

Obrazovka na počítači 4D Server by měla být nastavena na 640 x 480 x 256 barev. Toto nastavení zabrání přílišnému zatěžování CPU při překreslování obrazovky. Důvod je zřejmý, protože množství paměti požadované pro zobrazování na libovolném monitoru je rovné celkovému počtu bodů násobenému barevnou hloubkou je požadavek na RAM pro nastavení výše:

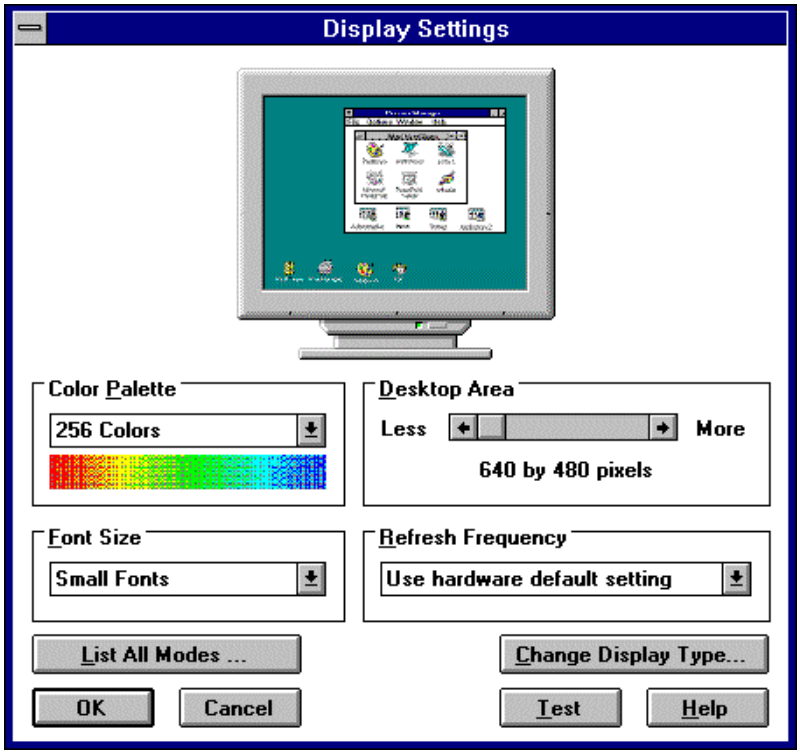
640 x 480 x 8 = 2,457,600 bitů nebo 307,200 bytů

Tentýž výpočet pro monitor zobrazující 1152 x 864 x 65,536 barev je:

1152 x 864 x 16 = 15,925,248 bitů nebo 1,990,656 bytů

Na deskách, které nemají grafický akcelerátor je CPU zodpovědná za většinu ovládání videozobrazení. Takže při více bodech a větší hloubce podstatně více zatěžujeme CPU. Tato nastavení jsou určitě zbytná pro počítač 4D Server.

Ušetříte si RAM a CPU pro mnohem užitečnější funkce, když nastavíte obrazovku na nejmenší možné rozlišení. Nastavení obrazovky může být nalezeno v Control panel - Display. Níže vidíte obrázek obrazovky tohoto řídícího panelu se správným nastavením:

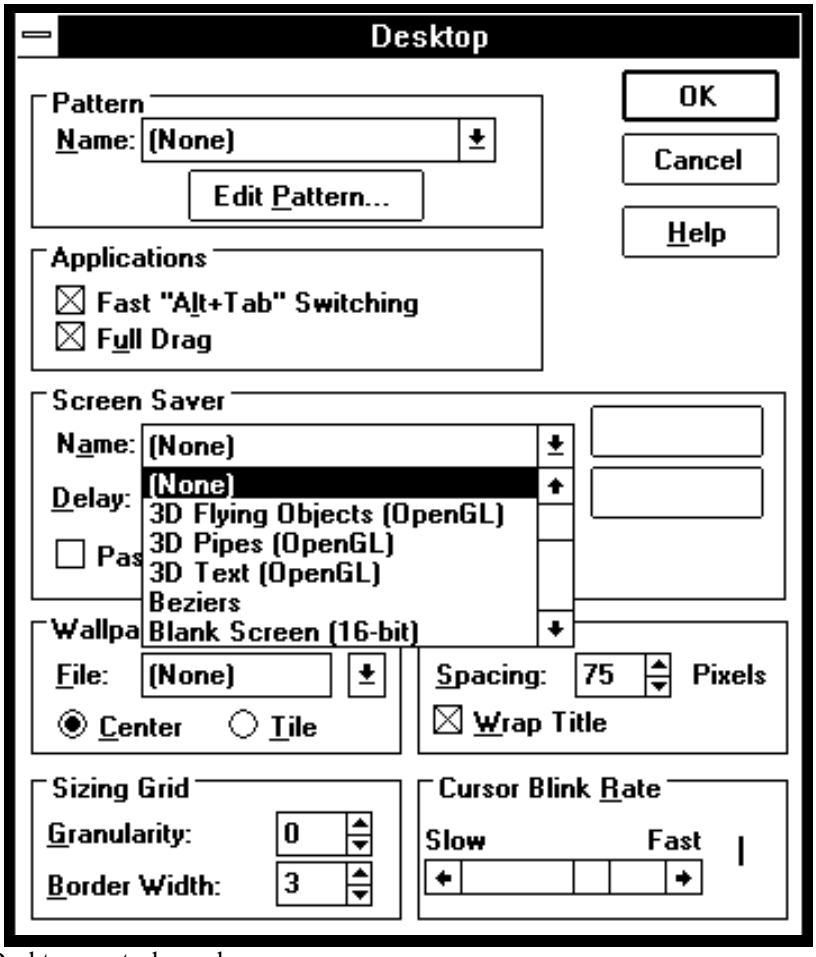


Display control panel

Nastavení plochy

Mělo by být použito nejjednodušší šetření obrazovky. Nejjednodušší šetření obrazovky je prázdná obrazovka nebo vybráno None. Pokude vyberete None můžete jednoduše monitor vypnout.

Tyto možnosti lze nastavit v řídícím panelu Desktop. Následující obrázek je snímek obrazovky z tohoto panelu:



Desktop control panel

Závěr

Windows NT je silná platforma na které lze spustit 4D Server a další produkty 4D. V současné době je Windows NT optimální platforma pro spuštění 4D Server. Windows NT může být optimalizován pomocí technik uvedených v této příloze. Tyto techniky zahrnují:

- Použití systému souborů NT
- Optimalizace Pagefile Size Settings
- Komprese složky obsahující datový soubor 4D
- Spuštění 4D Server z batch souboru v REALTIME mode
- Konfigurace zobrazení
- Konfigurace plochy (desktop)

Rejstřík

This index does not reflect every occurrence of the items listed in the index. Rather where the items are discussed and when appropriate good examples of how to use them in actual code. All code segments are listed every time they are created, modified or finalized. If you want to locate where project methods are called in other code segments, you will need to use 4D Insider.

4D Open,256

stack,252