4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Úvod (kapitola)

Definice jazyka

Prostředí 4D

Application type → Long Integer	Typ aplikace
Version type → Long Integer	Typ verze
Application version ({*}) → Řetězec	Verze aplikace
Compiled application → Logické	Kompilovaná aplikace
PLATFORM PROPERTIES (platforma{; Systém{; Stroj}})	VLASTNOSTI PLATFORMY
Application file → Řetězec	Soubor aplikace
Structure file → Řetězec	Soubor struktury
Data file {(část)} → Řetězec	Datový soubor
ACI folder → Řetězec	SLOŽKA ACI
DATA SEGMENT LIST (Části)	SEZNAM ČÁSTÍ DAT
ADD DATA SEGMENT	PŘIDAT ČÁST DAT
FLUSH BUFFERS	ULOŽIT BUFFERY
QUIT 4D	UKONČIT 4D
SELECT LOG FILE (logSoubor | *)	VYBRAT LOG SOUBOR

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Array (kapitola)

Array
Vytváření array
Array a objekty formuláře
Vytvoření skupiny posuvných oblastí
Array a jazyk 4D
Array a Ukazatele
Použití prvku nula v array
Dvojrozměrné array
Array a paměť

ARRAY INTEGER (NázevArray; Velikost {; velikost2})	
ARRAY LONGINT (NázevArray; velikost {;velikost2})	
ARRAY REAL (NázevArray; velikost {;velikost2})	
ARRAY STRING(délka řetězce; NázevArray; velikost {;velikost2})	
ARRAY TEXT (NázevArray; velikost {;velikost2})	
ARRAY DATE (NázevArray; velikost {;velikost2})	
ARRAY BOOLEAN (NázevArray; velikost {;velikost2})	
ARRAY PICTURE (NázevArray; velikost {;velikost2})	
ARRAY POINTER (NázevArray; velikost {;velikost2})	
Size of array (array) číslo	Velikost array
SORT ARRAY (array{;array2; ...; arrayN}{; > nebo <})	TŘÍDIT ARRAY
Find in array (array; hodnota {; start}) → číslo	Nalézt v array
INSERT ELEMENT (array; kam; {; kolik})	VLOŽIT PRVEK
DELETE ELEMENT (array; kde{; kolik})	VYMAZAT PRVEK
COPY ARRAY (zdroj; cíl)	KOPÍROVAT ARRAY
LIST TO ARRAY (seznam; array {; OdkPol})	SEZNAM DO ARRAY
ARRAY TO LIST (array; Seznam {; OdkPol})	ARRAY DO SEZNAMU
SELECTION TO ARRAY (pole | tabulka; array {;pole2 | tabulka2;....;poleN | tabulkaN; arrayN})	VÝBĚR DO ARRAY
SELECTION RANGE TO ARRAY(start; konec; pole | tabulka; array {;pole2 | tabulka2; array2;....; poleN | tabulkaN; arrayN})	ROZSAH VÝBĚRU DO ARRAY
ARRAY TO SELECTION (array; pole {;array2; pole2; .....; arrayN;poleN})	ARRAY DO VÝBĚRU
DISTINCT VALUES (pole; array)	ODLIŠNÉ HODNOTY
LONGINT ARRAY FROM SELECTION ({tabulka; }záznamArray{; výběr})	LONGINT ARRAY Z VÝBĚRU
BOOLEAN ARRAY FROM SET (logickéArray{; sada})	LOGICKÉ ARRAY ZE SADY

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

BLOB

Příkazy BLOB	
SET BLOB SIZE (blob; velikost {; vyplnit})	NASTAVIT VELIKOST BLOB
BLOB size (blob) → Číslo	Velikost BLOB
COMPRESS BLOB (blob{; komprese})	ZABALIT BLOB
EXPAND BLOB(blob)	ROZBALIT BLOB
BLOB PROPERTIES (blob; zabaleno{; rozbalenáVel{; AktVelikost}})	VLASTNOSTI BLOB
DOCUMENT TO BLOB (dokument; blob{; *})	DOKUMENT DO BLOBU
BLOB TO DOCUMENT (dokument; blob{; *})	BLOB DO DOKUMENTU
VARIABLE TO BLOB (proměnná; blob{; *})	PROMĚNNÁ DO BLOBU
BLOB TO VARIABLE (blob; proměnná{; offset})	BLOB DO PROMĚNNÉ
LIST TO BLOB (seznam; blob{; *})	SEZNAM DO BLOBU
BLOB to list (blob{; offset}) → OdkSeznam	BLOB do seznamu
INTEGER TO BLOB (integer; blob; PořadíBytů{; offset | *})	INTEGER DO BLOBU
LONGINT TO BLOB (longInt; blob; PořadíBytů{; offset | *})	LONGINT DO BLOBU
REAL TO BLOB (real; blob; FormátReal{; offset | *})	REAL DO BLOBU
TEXT TO BLOB (text; blob; FormátText{; offset | *})	TEXT DO BLOBU
BLOB to integer (blob; pořadíBytů{; offset}) → Číslo	BLOB do integer
BLOB to longint (blob; pařadíBytů{; offset}) → Číslo	BLOB do longint
BLOB to real (blob; FormátReal{; offset})	BLOB do rela
BLOB to text (blob; FormátText{; offset{; DélkaTextu}})	BLOB do textu
INSERT IN BLOB (blob; offset; délka{; vyplnit})	VLOŽIT DO BLOBU
DELETE FROM BLOB (blob; offset; délka)	VYMAZAT Z BLOBU
COPY BLOB (zdrBLOB; cílBLOB; zdrOffset; cílOffset; délka)	KOPÍROVAT BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Datum a Čas

Current date ({*}) → Datum	Platné datum
Day of (datum) → Číslo	Den z
Month of (datum) → Číslo	Měsíc z
Year of (datum) → Číslo	Rok z
Day number (datum) → Číslo	Číslo dne
Add to date (datum; roků; měsíců; dní) → Datum	Přidat k datumu
Date (ŘetězecDatum) → Datum	Datum
Current time ({*})→ Čas	Platný čas
Time string (sekund) → Čas	Časový řetězec
Time (časovýŘetězec) → Čas	Čas
Tickcount → Číslo	Počet tiků
Milliseconds → Číslo	Milisekund
SET DEFAULT CENTURY (století{; dělícíRok})	NASTAVIT VÝCHOZÍ STOLETÍ

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Dotazy

QUERY BY EXAMPLE ({tabulka}{;}{*})	DOTAZ DLE PŘÍKLADU
QUERY ({tabulka}{; argumentdotazu }{; *})	DOTAZ
QUERY SELECTION ({tabulka}{; argumentdotazu }{; *})	DOTAZ VE VÝBĚRU
QUERY BY FORMULA ({tabulka}{; VýrazDotazu })	DOTAZ DLE VÝRAZU
QUERY SELECTION BY FORMULA ({tabulka}{; VýrazDotazu })	DOTAZ VE VÝBĚRU DLE VÝRAZU
SET QUERY DESTINATION (cílovýTyp{; cílovýObjekt})	NASTAVIT CÍLENÍ DOTAZŮ
SET QUERY LIMIT (limit)	NASTAVIT OMEZENÍ DOTAZU
Find index key (indexovanéPole; hodnota) → LongInt	Nalézt v indexech pole
ORDER BY ({tabulka}{; pole}{; >nebo<}{; pole2; >nebo<2; ...; poleN; >nebo<N})	TŘÍDIT DLE
ORDER BY FORMULA (tabulka{; Výraz}{; >nebo<}{; Výraz2; >nebo<2; ...; VýrazN; >nebo<N})	TŘÍDIT DLE VÝRAZU

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Diagramy

GRAPH (oblastgraf; grafčíslo; xpopis; yprvky{; yprvky2; ...; yprvkyN})	DIAGRAM
GRAPH SETTINGS (graf; xmin; xmax; ymin; ymax; xprop; xsíť; ysíť; titul{; titul2; ...; titulN})	NASTAVENÍ DIAGRAMU
GRAPH TABLE ({tabulka; }grafTyp; xpole;ypole{;ypole2; ...; ypoleN})	VYTVOŘIT DIAGRAM Z TABULKY

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Hierarchické seznamy

Load list (NázevSeznamu) → ListRef	Načíst seznam
SAVE LIST (list; listNázev)	ULOŽIT SEZNAM
New list → ListRef	Nový seznam
Copy list (list) → ListRef	Kopírovat seznam
CLEAR LIST (listRef{; *})	ODSTRANIT SEZNAM
Count list items (list) → Long Integer	Počet položek seznamu
Is a list (list) → Logické	Je seznamem
REDRAW LIST (list)	PŘEKRESLIT SEZNAM
SET LIST PROPERTIES (list; vzhled{; ikona{; výškařád}})	NASTAVIT VLASTNOSTI SEZNAMU
GET LIST PROPERTIES (list; vzhled{; ikona{; výškařád}})	ZÍSKAT VLASTNOSTI SEZNAMU
SORT LIST (list{; > nebo <})	TŘÍDIT SEZNAM
APPEND TO LIST (list; PoložkaText; PoložkaRef{; podlist{; rozšiř}})	PŘIPOJIT K SEZNAMU
INSERT LIST ITEM (list; PředPolRef | *; PoložkaText; PoložkaRef{; podlist{; rozšiř}})	VLOŽIT POLOŽKU SEZNAMU
SET LIST ITEM PROPERTIES (list;PoložkaRef; dostup; styl; ikona)	NASTAVIT VLASTNOSTI POLOŽKY SEZNAMU
GET LIST ITEM PROPERTIES (list;PoložkaRef;dostup{;styl{;ikona}})	ZÍSKAT VLASTNOSTI POLOŽKY SEZNAMU
List item position (list; PoložkaRef) → Číslo	Pozice položky v seznamu
List item parent (list; PoložkaRef) → Číslo	Rodič položky seznamu
DELETE LIST ITEM (List; položkaRef | *{; *})	VYMAZAT POLOŽKU SEZNAMU
GET LIST ITEM (list; PoložkaPoz; PoložkaRef; PoložkaText{; podlist{; rozšiř}})	ZÍSKAT POLOŽKU SEZNAMU
SET LIST ITEM (list; PoložkaRef; NovýPolText; NováPolRef{; podlist{; rozšiř}})	NASTAVIT POLOŽKU SEZNAMU
Selected list item (list) → Long Integer	Označená položka seznamu
SELECT LIST ITEM (list; PoložkaPoz)	OZNAČIT POLOŽKU SEZNAMU
SELECT LIST ITEM BY REFERENCE (list; PoložkaRef)	OZNAČIT POLOŽKU SEZNAMU PODLE ODKAZU

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Hlášení

MESSAGES OFF	VYPNOUT HLÁŠENÍ
MESSAGES ON	ZAPNOUT HLÁŠENÍ
ALERT (zpráva{; text tlačítka OK})	UPOZORNIT
CONFIRM (zpráva{; titul tlačítka OK{; titul tlačítka Storno}})	POTVRDIT
Request (zpráva{; výchozí odpověď{; titul OK{; titul Storno}}}) → Řetězec	Žádost
MESSAGE (zpáva)	HLÁŠENÍ
GOTO XY (x;y)	JÍT NA XY

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Import a Export

IMPORT TEXT ({tabulka; }dokument)	IMPORTOVAT TEXT
EXPORT TEXT ({tabulka; }dokument)	EXPORTOVAT TEXT
IMPORT SYLK ({tabulka; }dokument)	IMPORTOVAT FORMÁT SYLK
EXPORT TEXT ({tabulka; }dokument)	EXPORTOVAT FORMÁT SYLK
IMPORT DIF ({tabulka; }dokument)	IMPORTOVAT FORMÁT DIF
EXPORT DIF ({tabulka; }dokument)	EXPORTOVAT FORMÁT DIF
IMPORT DATA (SouborNázev{; projekt{; *}})	IMPORTOVAT DATA
EXPORT DATA (SouborNázev{; projekt{; *}})	EXPORTOVAT DATA

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Jazyk

Count parameters → Číslo	Počet parametrů
Type (PoleProm) → Číslo	Typ dat
Self → Ukazatel	Samoukazatel
RESOLVE POINTER (ukazatel; PromNazev; tabČíslo; poleČíslo)	UKAZUJE NA
Nil (Ukazatel) → Logické	Prázdný
Is a variable (Ukazatel) → Logické	Je proměnnou
Get pointer (NázevProm) → Ukazatel	Získat ukazatel
Poznámka: Tento příkaz budete používat jen vyjímečně.	PROVÉST
Command name (příkaz) → Řetězec	Název příkazu
TRACE	KROKOVAT
NO TRACE	NEKROKOVAT

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Kompilátor

Příkazy kompilátoru
C_BLOB ({metoda; }proměnná{; proměnná2;...;proměnnáN})
C_BOOLEAN ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_DATE ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_GRAPH ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_INTEGER ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_LONGINT ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_PICTURE ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_POINTER ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_REAL ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_STRING ({metoda}velikost; proměnná{; proměnná2;...;proměnnáN})
C_TEXT ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_TIME ({metoda}proměnná{; proměnná2;...;proměnnáN})
IDLE

PRODLEVA

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Komunikace

SET CHANNEL (port | operace; nastavení | dokument)	NASTAVIT KANÁL
SET TIMEOUT (sekundy)	NASTAVIT ČASOVÝ LIMIT
USE ASCII MAP (mapa | *{; mapInOut})	POUŽÍT ASCII MAPU
SEND PACKET ({odkDok; }balík)	POSLAT DATOVÝ BALÍK
RECEIVE PACKET ({odkDok;}dplProm; stopZnak | počZnak)	NAČÍST DATOVÝ BALÍK
RECEIVE BUFFER (doplProm)	PŘIJMOUT BUFFER
SEND VARIABLE (proměnná)	POSLAT PROMĚNNOU
RECEIVE VARIABLE (proměnná)	NAČÍST PROMĚNNOU
SEND RECORD ({tabulka})	POSLAT ZÁZNAM
RECEIVE RECORD ({tabulka})	NAČÍST ZÁZNAM

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Kontrola vstupu

ACCEPT	PŘIJMOUT
CANCEL	ZRUŠIT
Keystroke → řetězec	Stisknuta klávesa
FILTER KEYSTROKE (ZnakFiltru)	FILTR ÚHOZU NA KLÁVESU
GOTO AREA ({*; }objekt)	JÍT NA OBLAST
REJECT {(pole)}	ODMÍTNOUT

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Ladění (kapitola)

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Logické

Logické příkazy

True → Logické	Pravda
False → Logické	Nepravda
Not (Logické) → Logické	Opak

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Matematické řady (kapitola)

Matematické řady	
Sum (čís.řada) → Číslo	Součet
Average (čís.řada) → Číslo	Průměr
Min (čís.řada) → Číslo	Minimum
Max (čís.řada) → Číslo	Maximum
Std deviation (čís.řada) → Číslo	Standardní odchylka
Variance (čís.řada) → Číslo	Variance
Sum squares (čís.řada) → Číslo	Součet čtverců

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Matematika

Abs (číslo) → Číslo	Absolutní hodnota
Int (číslo) → Číslo	Celá část
Dec (číslo) → Číslo	Desetinná část
Round (zaokr; míst) → Číslo	Zaokrouhlit
Trunc (kořízn; míst) → Číslo	Oříznout
Random → Číslo	Náhodné
Mod (číslo1; číslo2) -→ Číslo	Zbytek
Square root (číslo) → Číslo	Druhá odmocnina
Log(číslo) → Číslo	Log
Exp (číslo) → Číslo	Exp
Sin (číslo) → Číslo	Sinus
Cos (číslo) → Číslo	Cosinus
Tan (číslo) → Číslo	Tangens
Arctan (číslo) → Číslo	Arcustangens
SET REAL COMPARISON LEVEL (epsilon)	NASTAVIT ÚROVEŇ POROVNÁVÁNÍ
Zobrazení reálných čísel	
Euro Converter (hodnota; zMěny; doMěny) → Real	Převést euro měny

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Metody databáze (kapitola)

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Nabídky

Řízení nabídek	
MENU BAR (záhlnabídky{; proces}{; *})	ZÁHLAVÍ NABÍDEK
HIDE MENU BAR	SKRÝT ZÁHLAVÍ NABÍDEK
SHOW MENU BAR	UKÁZAT ZÁHLAVÍ NABÍDEK
SET ABOUT (TextPoložky; metoda)	NASTAVIT O APLIKACI
Menu selected → Číslo	Označená nabídka
Count menus ({proces}) → Číslo	Počet nabídek
Count menu items (nabídka{; proces}) → Číslo	Počet položek nabídky
Get menu title (nabídka{; proces}) → Řetězec	Získat název nabídky
Get menu item (nabídka; položkaNab{; proces}) → Řetězec	Získat text položky nabídky
SET MENU ITEM (Nabídka; PoložkaNab; TextPoložky{; proces})	NASTAVIT POLOŽKU NABÍDKY
Get menu item style (nabídka; položkaNab{; proces}) → Číslo	Získat styl položky nabídky
SET MENU ITEM STYLE (nabídka; položkaNab; PoložkaStyl {; proces})	NASTAVIT STYL POLOŽKY NABÍDKY
Get menu item mark (nabídka; položkaNab{; proces}) → Řetězec	Získat značku položky nabídky
SET MENU ITEM MARK (nabídka; položka; značka{; proces})	NASTAVIT ZNAČKU POLOŽKY NABÍDKY
Get menu item key (nabídka; položkaNab{; proces}) → Číslo	Získat zkratku k položce nabídky
SET MENU ITEM KEY (nabídka; položkaNab; položkaKlíč{;proces})	NASTAVIT ZKRATKU K POLOŽCE NABÍDKY
DISABLE MENU ITEM (nabídka; položkaNab{; proces})	ZNEPŘÍSTUPNIT POLOŽKU NABÍDKY
ENABLE MENU ITEM (nabídka; položkaNab{; proces})	ZPŘÍSTUPNIT POLOŽKU NABÍDKY
APPEND MENU ITEM (nabídka; položkaText{; proces})	PŘIPOJIT POLOŽKU NABÍDKY
INSERT MENU ITEM (nabídka; zaPoložku; PoložkaText{; process})	VLOŽIT POLOŽKU NABÍDKY
DELETE MENU ITEM (nabídka; položkaNab{; proces})	VYMAZAT POLOŽKU NABÍDKY

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Obrázky (kapitola)

Obrázky	
COMPRESS PICTURE (obrázek; metoda; kvalita)	ZABALIT OBRÁZEK
LOAD COMPRESSED PICTURE FROM FILE (dokument; metoda; kvalita; obrázek)	NAČÍST ZABALENÝ OBRÁZEK ZE SOUBORU
COMPRESS PICTURE FILE (dokument; metoda; kvalita)	ZABALIT SOUBOR OBRÁZKU
SAVE PICTURE TO FILE (dokument; obrázek)	ULOŽIT OBRÁZEK DO SOUBORU
PICT TO GIF (obr; blobGIF)	OBRÁZEK DO GIF
Picture size (obrázek) → Číslo	Velikost obrázku
PICTURE PROPERTIES (obrázek; šířka; výška{; hOffset{; vOffset{; mod}}})	VLASTNOSTI OBRÁZKU
PICTURE LIBRARY LIST (picRefs; picNázvy)	SEZNAM OBRÁZKŮ Z KNIHOVNY
GET PICTURE FROM LIBRARY (picRef; obrázek)	ZÍSKAT OBRÁZEK Z KNIHOVNY
SET PICTURE TO LIBRARY (obrázek; picRef; picNázev)	PŘIDAT OBRÁZEK DO KNIHOVNY
REMOVE PICTURE FROM LIBRARY (picRef)	ODSTRANIT OBRÁZEK Z KNIHOVNY

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Okna

Řízení oken	
Open window (levá; vrch; pravá; spodní{; typ{; titul{; řídícíokénko}}}){ → WinRef }	Otevřít okno
Typy oken	
Open external window (levá; vrch; pravá; spodní; typ; titlul; OblastPlugIn) → WinRef	Otevřít externí okno
SHOW WINDOW ({okno})	UKÁZAT OKNO
HIDE WINDOW ({okno})	SKRÝT OKNO
CLOSE WINDOW {(RefExtOkno)}	ZAVŘÍT OKNO
ERASE WINDOW {(okno)}	VYMAZAT OKNO
REDRAW WINDOW {(okno)}	PŘEKRESLIT OKNO
DRAG WINDOW	PŘETÁHNOUT OKNO
Get window title {(okno)} → Řetězec	Získat název okna
SET WINDOW TITLE (titul{; okno})	NASTAVIT NÁZEV OKNA
HIDE TOOL BAR	SKRÝT LIŠTU NÁSTROJŮ
SHOW TOOL BAR	UKÁZAT LIŠTU NÁSTROJŮ
WINDOW LIST (okna{; *})	SEZNAM OKEN
Window kind {(okno)}	Typ okna
Window process {(okno)} → Číslo	Proces okna
GET WINDOW RECT (levá; vrch; pravá; spodní{; okno})	ZÍSKAT SOUŘADNICE OKNA
SET WINDOW RECT (levá; vrch; pravá; spodní{; okno})	NASTAVIT RÁMEC OKNA
Frontmost window {(*)} → WinRef	Okno na popředí
Next window (okno) → Číslo	Další okno
Find window (levá; vrch{; oknočást}) -→> WinRef	Najít okno
MAXIMIZE WINDOW {(okno)}	MAXIMALIZOVAT OKNO
MINIMIZE WINDOW {(okno)}	MINIMALIZOVAT OKNO
Open form window ({tabulka; }formulářNázev{; typ{; hPoz{; vPoz{; *}}}}) → WinRef	Otevřít okno formuláře
GET FORM PROPERTIES ({tabulka; }formulář; šířka; výška{; počStran{; PromŠíř{; PromVýš{; titul}}}})	VZÍT VLASTNOSTI FORMULÁŘE

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Operátory (kapitola)

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Podzáznamy

CREATE SUBRECORD (podtabulka)	VYTVOŘIT PODZÁZNAM
DELETE SUBRECORD (podtabulka)	VYMAZAT PODZÁZNAM
ALL SUBRECORDS (podtabulka)	VŠECHNY PODZÁZNAMY
Records in subselection (podtabulka) → Číslo	Záznamů v podvýběru
APPLY TO SUBSELECTION (podtabulka; tvrzení)	POUŽÍT NA PODVÝBĚR
FIRST SUBRECORD (podtabulka)	PRVNÍ PODZÁZNAM
LAST SUBRECORD (podtabulka)	POSLEDNÍ PODZÁZNAM
NEXT SUBRECORD (podtabulka)	DALŠÍ PODZÁZNAM
PREVIOUS SUBRECORD (podtabulka)	PŘEDCHOZÍ PODZÁZNAM
Before subselection (podtabulka) > Logické	Před podvýběrem
End subselection (podtabulka) → Logické	Za podvýběrem
ORDER SUBRECORDS BY (podtabulka; podpole{; >nebo<}{; podpole2; >nebo<2; ...; podpoleN; >nebo<N})	TŘÍDIT PODVÝBĚR DLE
QUERY SUBRECORDS (podtabulka; VýrazDotazu)	DOTAZ NA PODZÁZNAMY

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Pojmenované výběry

Pojmenovaný výběr	
COPY NAMED SELECTION ({tabulka; }název)	KOPÍROVAT POJMENOVANÝ VÝBĚR
CUT NAMED SELECTION ({tabulka; }název)	VYJMOUT POJMENOVANÝ VÝBĚR
USE NAMED SELECTION (název)	POUŽÍT POJMENOVANÝ VÝBĚR
CLEAR NAMED SELECTION (název)	VYMAZAT POJMENOVANÝ VÝBĚR
CREATE SELECTION FROM ARRAY ({tabulka;}záznArray{; název})	VYTVOŘIT VÝBĚR Z ARRAY

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Procesy (kapitola)

Procesy	
New process (metoda; stack{; název{; param{; param2; ...; paramN}{; *}}}) → Číslo	Nový proces
Execute on server (metoda; stack{; název{; param{; param2; ...; paramN}{; *}}}) → Číslo	Provést na serveru
DELAY PROCESS (proces; trvání)	ODLOŽIT PROCES
PAUSE PROCESS (proces)	POZASTAVIT PROCES
RESUME PROCESS (proces)	NAVRÁTIT PROCES
Process aborted → Logické	Proces odstraněn
Current process → Číslo	Platný proces
Process state (proces) → Číslo	Stav procesu
PROCESS PROPERTIES (proces; procNázev; procStav; procČas{; procVidět{; jedinečnéID{; původ}}})	VLASTNOSTI PROCESU
Process number (název{; *}) → Číslo	Číslo procesu
Count users → Integer	Počet uživatelů
Count tasks → Integer	Počet procesů
Count user processes → Integer	Počet procesů uživatele
EXECUTE ON CLIENT (NázevKlient; metoda{; param}{; param2; ...; paramN})	PROVÉST NA KLIENTOVI
REGISTER CLIENT (NázevKlienta{; perioda{; *}})	REGISTROVAT KLIENTA
UNREGISTER CLIENT	ODREGISTROVAT KLIENTA
GET REGISTERED CLIENTS (seznamKlientů; metody)	ZÍSKAT REGISTROVANÉ KLIENTY

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Procesy (komunikace)

Semaphore (semafor{; počet tiků}) → Logické	Semafor
CLEAR SEMAPHORE (semafor)	ODSTRANIT SEMAFOR
Test semaphore (semafor) → Logické	Testovat semafor
CALL PROCESS (proces)	ZAVOLAT PROCES
GET PROCESS VARIABLE (proces; zdrojProm; cílProm{; zdrojProm2; cílProm2; ...; zdrojPromN; cílPromN})	ZÍSKAT PROMĚNNOU Z PROCESU
SET PROCESS VARIABLE (proces; cílProm; výraz{; cílProm2; výraz2; ...; cílPromN; výrazN})	NASTAVIT PROMĚNNOU DO PROCESU
VARIABLE TO VARIABLE (proces; cílProm; zdrProm{; cílProm2; zdrProm2; ...; cílPromN; zdrPromN})	PROMĚNNÁ DO PROMĚNNÉ

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Procesy (rozhraní)

HIDE PROCESS (proces)	SKRÝT PROCES
SHOW PROCESS (proces)	UKÁZAT PROCES
BRING TO FRONT (proces)	PŘENÉST NA POPŘEDÍ
Frontmost process {(*)} → Integer	Proces na popředí

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Proměnné

SAVE VARIABLES (dokument; prom{; prom2; ...; promN})	ULOŽIT PROMĚNNÉ
LOAD VARIABLES (dokument; prom{; prom2; ...; promN})	NAČÍST PROMĚNNÉ
CLEAR VARIABLE (proměnná)	VYMAZAT PROMĚNNOU
Undefined (proměnná) → Logické	Nedefinováno

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Prostředí systému

Screen height {(*)} → Číslo	Výška obrazovky
Screen width {(*)} → Číslo	Šířka obrazovky
Count screens → Číslo	Počet obrazovek
SCREEN COORDINATES (levá; vrch; pravá; spodní{; obrazovka})	SOUŘADNICE OBRAZOVKY
SCREEN DEPTH (hloubka; barva{; obrazovka})	HLOUBKA OBRAZOVKY
SET SCREEB DEPTH (hloubka{; barva{; obrazovka}})	NASTAVIT HLOUBKU OBRAZOVKY
Menu bar screen → Číslo	Obrazovka záhlaví nabídek
Menu bar height → Číslo	Výška záhlaví nabídek
FONT LIST (písma)	SEZNAM PÍSEM
Font name (ČísloPísma) → Řetězec	Název písma
Font number (NázevPísma) → Číslo	Číslo písma
System folder → Řetězec	Systémová složka
Temporary folder → Řetězec	Dočasná složka
Current machine → Řetězec	Platný stroj
Current machine owner → Řetězec	Platný vlastník stroje
Gestalt (dotaz; hodnota) → Číslo	LOG UDÁLOST
LOG EVENT (zpráva{; důležitost})	

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Přerušení

ON EVENT CALL (MetodaUdálostí{; NázevProcesu})	Při události provést
FILTER EVENT	ZACHYTIT UDÁLOST
ON ERR CALL (MetodaChyby)	Při chybě provést
ABORT	PŘERUŠIT

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Přístup k struktuře

Struktura	
Count tables → Číslo	Počet tabulek
Count fields (tabulkaČíslo | ukazTabulky) → Číslo	Počet polí
Table name (tabulkaČíslo | ukazTabulky) → Řetězec	Název tabulky
Field name (tabulkaČíslo; poleČíslo) | (ukazPole) → Řetězec	Název pole
Table (tabulkaČíslo | Ukazatel) → Ukazatel | Číslo	Tabulka
Field (tabulkačíslo | UkazatelPole{; polečíslo}) -→> Číslo | Ukazatel	Pole
GET FIELD PROPERTIES (tabulkačíslo; polečíslo | ukazpole; poleTyp{; poleDel{; indexované}})	ZÍSKAT VLASTNOSTI POLE
SET INDEX (pole; index{; mod{; *}})	NASTAVIT INDEX
Get database parameter ({tabulka; }volba) → Longint	Získat parametr databáze
SET DATABASE PARAMETER ({tabulka; }volba; hodnota)	NASTAVIT PARAMETR DATABÁZE

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Rozhraní uživatele

BEEP	PÍPNOUT
PLAY (objektnázev{; kanál})	ZAHRÁT
Get platform interface → Číslo	Získat rozhraní platformy
SET PLATFORM INTERFACE (interface)	NASTAVIT ROZHRANÍ PLATFORMY
SET TABLE TITLES (tabTituly; tabČísla)	NASTAVIT NÁZVY TABULEK
SET FIELD TITLES (tabulka | podtabulka; poleTituly; poleČísla)	NASTAVIT NÁZVY POLÍ
Shift down → Logické	Stisknut shift
Caps lock down → Logické	Stisknut zámek písmen
Windows Ctrl down → Logické	Stisknut windows ctrl
Windows Alt down → Logické	Stisknut windows alt
Macintosh command down → Logické	Stiskut macintosh command
Macintosh option down → Logické	Stisknut macintosh option
Macintosh control down → Logické	Stisknut macintosh control
GET MOUSE (myšX; myšY; myšTlačítko{; *})	ZÍSKAT MYŠ
Pop up menu (obsah{; výchozí}) -→> Číslo	Rozevírací nabídka
POST KEY (kód{; modifikátor{; proces}})	POSLAT KLÁVESU
POST CLICK (myšX; myšY{; proces}{; *})	POSLAT KLEPNUTÍ
POST EVENT (co;zpráva;kdy;myšX;myšY;modifikátor{;proces})	POSLAT UDÁLOST
GET HIGHLIGHT (oblast; počvyb; konecvyb)	ZÍSKAT ZVÝRAZNĚNÉ
HIGHLIGHT TEXT (oblast; počvyb; konecvyb)	ZVÝRAZNIT TEXT
SET CURSOR {(kurzor)}	NASTAVIT KURZOR
Last object → Ukazatel	Poslední objekt
REDRAW (objekt)	PŘEKRESLIT
INVERT BACKGROUND ({*; }textProm | textPole)	INVERTOVAT POZADÍ

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Sady (kapitola)

Sady	
CREATE EMPTY SET ({tabulka ;}set)	VYTVOŘIT PRÁZDNOU SADU
CREATE SET ({tabulka ;}set)	VYTVOŘIT SADU
CREATE SET FROM ARRAY ({tabulka;}záznArray{; setNázev})	VYTVOŘIT SADU Z ARRAY
USE SET (set)	POUŽÍT SADU
ADD TO SET ({tabulka ;}set)	PŘIDAT DO SADY
REMOVE FROM SET ({tabulka ;}set)	ODSTRANIT ZE SADY
CLEAR SET (set)	ODSTRANIT SADU
Is in set (set) → Logické	Je v sadě
Records in set (set) → Číslo	Záznamů v sadě
SAVE SET (set; dokument)	ULOŽIT SADU
LOAD SET ({tabulka; }set; dokument)	NAČÍST SADU
DIFFERENCE (set; odečítSet; výslSet)	ROZDÍL
INTERSECTION (set1; set2; VýsleSet)	PRŮNIK
UNION (set1; set2; výsleSet)	SJEDNOCENÍ
COPY SET (zdrSet; cílSet)	KOPÍROVAT SADU

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Schránka

APPEND TO CLIPBOARD (DruhDat; data)	PŘIPOJIT DO SCHRÁNKY
CLEAR CLIPBOARD	VYMAZAT SCHRÁNKU
GET CLIPBOARD (drihDat; data)	ZÍSKAT ZE SCHRÁNKY
GET PICTURE FROM CLIPBOARD (obrázek)	ZÍSKAT OBRÁZEK ZE SCHRÁNKY
Get text from clipboard → Řetězec	Získat text ze schránky
SET PICTURE TO CLIPBOARD (obrázek)	VLOŽIT OBRÁZEK DO SCHRÁNKY
SET TEXT TO CLIPBOARD (text)	VLOŽIT TEXT DO SCHRÁNKY
Test clipboard(druhDat) → Číslo	Testovat schránku

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Staré příkazy

SEARCH BY INDEX	HLEDAT DLE INDEXU
SORT BY INDEX	TŘÍDIT DLE INDEXU
ON SERIAL PORT CALL (serialMetoda{; proces})	Při volání sériového portu

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Stránky formuláře (kapitola)

Stránky formuláře	
FIRST PAGE	PRVNÍ STRÁNKA
LAST PAGE	POSLEDNÍ STRÁNKA
NEXT PAGE	DALŠÍ STRÁNKA
PREVIOUS PAGE	PŘEDCHOZÍ STRÁNKA
GOTO PAGE (číslo stránky)	JÍT NA STRÁNKU
Current form page → Číslo	Platná stránka formuláře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Systémové dokumety (kapitola)

Systémové dokumenty	
Open document (dokument{;TypSouboru{;mod}}) → DocRef	Otevřít dokument
Create document (dokument{; typSouboru}) → DocRef	Vytvořit dokument
Append document (dokument{; typSouboru}) → DorRef	Připojit k dokumentu
CLOSE DOCUMENT (docRef)	ZAVŘÍT DOKUMENT
COPY DOCUMENT (zdrojNázev; cílNázev{; *})	KOPÍROVAT DOKUMENT
MOVE DOCUMENT (zdrCesta; cílCesta)	PŘESUNOUT DOKUMENT
DELETE DOCUMENT (dokument)	VYMAZAT DOKUMENT
Test path name (cesta) → Číslo	Testovat cestu
CREATE FOLDER (cestasložky)	VYTVOŘIT SLOŽKU
Select folder ({zpráva}) → Řetězec	Vybrat složku
VOLUME LIST (jednotky)	SEZNAM JEDNOTEK
VOLUME ATTRIBUTES (jenotka; velikost; užito; volno)	VLASTNOSTI JEDNOTKY
FOLDER LIST (cesta; složky)	SEZNAM SLOŽEK
DOCUMENT LIST (cesta; dokumenty)	SEZNAM DOKUMENTŮ
Document type (dokument) → Řetězec	Typ dokumentu
SET DOCUMENT TYPE (document; SouborTyp)	NASTAVIT TYP DOKUMENTU
MAP FILE TYPES (MacOS; Windows; obsah)	PŘIŘADIT TYPY SOUBORŮ
Document creator (dokument) → Řetězec	Původce dokumentu
SET DOCUMENT CREATOR (dokument; SouborCreator)	NASTAVIT PŮVODCE DOKUMENTU
GET DOCUMENT PROPERTIES (dokument; zamčen; neviděn; vytvořenD; vytvořenČ; upravenD; upravenČ)	ZÍSKAT VLASTNOSTI DOKUMENTU
SET DOCUMENT PROPERTIES (dokument; zamčen; neviděn; vytvořenD; vytvořenČ; upravenD; upravenČ)	NASTAVIT VLASTNOSTI DOKUMENTU
Get document size (dokument{; *}) → Číslo	Získat velikost dokumentu
SET DOCUMENT SIZE (dokument; velikost)	NASTAVIT VELIKOST DOKUMENTU
Get document position (docRef) → Číslo	Získat pozici v dokumentu
SET DOCUMENT POSITION (docRef; offset{; ukotvení})	NASTAVIT POZICI V DOKUMENTU

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Tabulky

DEFAULT TABLE (tabulka)	VÝCHOZÍ TABULKA
Current default table → Ukazatel	Platná výchozí tabulka
INPUT FORM ({tabulka; }formulář{; *})	VSTUPNÍ FORMULÁŘ
OUTPUT FORM ({tabulka ;}formulář)	VÝSTUPNÍ FORMULÁŘ
Current form table → Ukazatel	Tabulka platného formuláře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Táhnout a vsadit (kapitola)

Táhnout a vsadit	
Drop position → Číslo	Umístění vsazení
DRAG AND DROP PROPERTIES (zdrObjekt; zdrPrvek; zdrProces)	VLASTNOSTI TAŽENÍ A VSAZENÍ

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Transakce

Použití Transakcí	
START TRANSACTION	ZAČÍT TRANSAKCI
VALIDATE TRANSACTION	POTVRDIT TRANSAKCI
CANCEL TRANSACTION	ZRUŠIT TRANSAKCI
In transaction → Logické	V transakci

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Text řetězce

String (výraz{; formát}) → Řetězec	Řetězec
Num (výraz) → Číslo	Číslo
Position(nalézt; řetězec) → Číslo	Pozice
Substring (zdroj; prvnízn{; početzn}) → Řetězec	Podřetězec
Length (řetězec) → Číslo	Délka
Ascii (znak) → Číslo	Ascii kód
Char (asciikód) → Řetězec	Znak
Symboly odkazů na znaky	
Uppercase (řetězec) → Řetězec	Velká písmena
Lowercase (Řetězec) → Řetězec	Malá písmena
Change string (zdroj; novéznaky; kde) → Řetězec	Změnit řetězec
Insert string (zdroj; co; kam) → Řetězec	Vložit řetězec
Delete string (zdroj; kde; početZnaků) → Řetězec	Vymazat řetězec
Replace string (zdroj; původřet; novýřet{; kolik}) → Řetězec	Nahradit řetězec
Mac to Win (text) → Řetězec	Mac do Win
Win to Mac (text) → Řetězec	Win do Mac
Mac to ISO (text) → Řetězec	Mac do ISO
ISO to Mac (text) → Řetězec	ISO do Mac

Position(»Position»

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Tisk

REPORT ({tabulka; }dokument{; *})	Sestava
PRINT LABEL ({tabulka}{; dokument}{; *})	TISKNOUT ŠTÍTEK
PRINT SELECTION ({tabulka}{; }{*})	TISKNOUT VÝBĚR
Printing page → Číslo	Tištěná stránka
BREAK LEVEL (úroveň {; zlomStr})	ÚROVEŇ ZLOMU
ACCUMULATE (data{; data2; ...;dataN})	AKUMULOVAT
Subtotal (data{; zlomstr}) → Číslo	Mezisoučet
Level → Číslo	Úroveň
PRINT RECORD ({tabulka}{; }{*})	TISKNOUT ZÁZNAM
PAGE SETUP ({tabulka; }formulář)	NASTAVENÍ STRÁNKY
PRINT SETTINGS	NASTAVENÍ TISKU
SET PRINT PREVIEW (náhled)	NASTAVIT NÁHLED TISKU
PRINT FORM ({tabulka; }formulář)	TISKNOUT FORMULÁŘ
PAGE BREAK {(* | >)}	ZLOMIT STRÁNKU

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Triggery (kapitola)

Triggery	
Database event → Číslo	Událost databáze
Trigger level → Číslo	Úroveň triggeru
TRIGGER PROPERTIES (úroveňTriggeru; dbUdálost; tabČíslo; záznamČíslo)	VLASTNOSTI TRIGGERU

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Události formuláře

Form event → Číslo	Událost formuláře
Before	Před
During	Během
After	Po
In header	V záhlaví
In break	Ve zlomu
In footer	V zápatí
Activated	Aktivované
Deactivated	Deaktivované
Outside call	Vnější volání
Get edited text → Text	Získat upravovaný text
SET TIMER (početTiků)	NASTAVIT ČASOVAČ

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Uživatelé a skupiny

EDIT ACCESS	UPRAVIT PŘÍSTUP
CHANGE ACCESS	ZMĚNIT PŘÍSTUP
CHANGE PASSWORD (heslo)	ZMĚNIT HESLO
Validate password (uživID; heslo) → Logické	Potvrdit heslo
Current user → Řetězec	Platný uživatel
User in Group (uživatel; skuipna) → Logické	Uživatel ve skupině
DELETE USER (uživID)	VYMAZAT UŽIVATELE
Is user deleted (uživatelČíslo) → Logické	Je uživatel vymazán
GET USER LIST (JménaUživatelů; ČíslaUživatelů)	ZÍSKAT SEZNAM UŽIVATELŮ
GET USER PROPERTIES (uživID; jméno; startup; heslo; početPřihl; poslPřihl{; členem})	ZÍSKAT VLASTNOSTI UŽIVATELE
Set user properties (uživID; jméno; startup; heslo; početPřihl; poslPřihl{; členem}) → Číslo	Nastavit vlastnosti uživatele
GET GROUP LIST (NázvySkupin; číslaSkupin)	ZÍSKAT SEZNAM SKUPIN
GET GROUP PROPERTIES (skupinaID; název; vlastník{; členi})	ZÍSKAT VLASTNOSTI SKUPINY
Set group properties (skupinaID; název; vlastník{; členi}) → Číslo	Nastavit vlastnosti skupiny
CHANGE WEB LICENCE	ZMĚNIT WEB LICENCI

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Vlastnosti objektů (kapitola)

Vlastnosti objektů	
FONT ({*; }objekt; písmo)	PÍSMO
FONT SIZE ({*; }objekt; velikost)	VELIKOST PÍSMA
FONT STYLE ({*; }objekt; styl)	STYL PÍSMA
ENABLE BUTTON({*; }objekt)	ZPŘÍSTUPNIT TLAČÍTKO
DISABLE BUTTON({*; }objekt)	ZNEPŘÍSTUPNIT TLAČÍTKO
BUTTON TEXT ({*; }objekt; TextTlačítka)	TEXT TLAČÍTKA
SET FORMAT ({*; }objekt; formátzobr)	NASTAVIT FORMÁT
SET FILTER ({*; }objekt; vstupFiltr)	NASTAVIT FILTR
SET CHOICE LIST ({*; }objekt; seznam)	NASTAVIT VÝBĚROVÝ SEZNAM
SET ENTERABLE ({*; }vstupníOblast; dostup)	NASTAVIT DOSTUPNOST
SET VISIBLE ({*; }objekt; viditelné)	NASTAVIT VIDITELNOST
SET COLOR VISIBLE ({*; }objekt; barva)	NASTAVIT BARVU
SET RGB COLORS ({*; }objekt; popředíBarva; pozadíBarva)	NASTAVIT RGB BARVU
GET OBJECT RECT ({*; }objekt; levá; vrch; pravá; spodní)	ZÍSKAT OBDÉLNÍK OBJEKTU
MOVE OBJECT ({*; }objekt; posunH; posunV{; změnaH{; změnaV{; *}}})	POSUNOUT OBJEKT

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Vstup dat

ADD RECORD ({tabulka}{; }{*})	PŘIDAT ZÁZNAM
MODIFY RECORD ({tabulka}{; }{*})	UPRAVIT ZÁZNAM
ADD SUBRECORD (podtabulka; formulář{; *})	PŘIDAT PODZÁZNAM
MODIFY SUBRECORD (podtabulka; formulář{; *})	UPRAVIT PODZÁZNAM
DIALOG ({tabulka; }formulář)	DIALOG
Modified (pole) → Logické	Upraveno
Old (pole) → Výraz	Staré

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Výběry

ALL RECORDS ({tabulka})	VŠECHNY ZÁZNAMY
Records in selection ({tabulka}) → Číslo	Záznamů ve výběru
DELETE SELECTION ({tabulka})	VYMAZAT VÝBĚR
Selected record number ({tabulka}) → Číslo	Pořadí záznamu ve výběru
GOTO SELECTED RECORD ({tabulka; }záznam)	JÍT NA ZÁZNAM VE VÝBĚRU
FIRST RECORD ({tabulka})	PRVNÍ ZÁZNAM
NEXT RECORD ({tabulka})	DALŠÍ ZÁZNAM
LAST RECORD ({tabulka})	POSLEDNÍ ZÁZNAM
PREVIOUS RECORD ({tabulka})	PŘEDCHOZÍ ZÁZNAM
Before selection ({tabulka}) → Logické	Před výběrem
End selection ({tabulka}) → Logické	Za výběrem
DISPLAY SELECTION ({tabulka}{; *}{; *})	ZOBRAZIT VÝBĚR
MODIFY SELECTION ({tabulka}{; *}{; *})	UPRAVIT VÝBĚR
APPLY TO SELECTION ({tabulka; }tvrzení)	POUŽÍT NA VÝBĚR
REDUCE SELECTION ({tabulka ;}počet)	OMEZIT VÝBĚR
SCAN INDEX (pole; počet {; > nebo <})	HLEDAT V INDEXU
ONE RECORD SELECT ({tabulka})	VYBRAT JEDEN ZÁZNAM
HIGHLIGHT RECORDS ({NázevSady})	ZVÝRAZNIT ZÁZNAMY

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Vztahy (kapitola)

Vztahy	
AUTOMATIC RELATIONS (jedinci{; skupiny})	AUTOMATICKÉ VZTAHY
RELATE ONE (TabulkaSkupin | Pole{; VýbPole})	VZTÁHNOUT K JEDINCI
RELATE MANY (TabulkaJed | Pole)	VZTÁHNOUT K SKUPINĚ
CREATE RELATED ONE (pole)	VYTVOŘIT VZTAŽENÉHO JEDINCE
SAVE RELATED ONE (pole)	ULOŽIT VZTAŽENÉHO JEDINCE
OLD RELATED ONE (pole)	VZTÁHNOUT K PŮVODNÍMU JEDINCI
SAVE OLD RELATED ONE (pole)	ULOŽIT PŮVODNÍHO VZTAŽENÉHO JEDINCE
OLD RELATED MANY (pole)	VZTÁHNOUT K PŮVODNÍ SKUPINĚ
RELATE ONE SELECTION (TabSkupin; TabJedinců)	VZTÁHNOUT VÝBĚR K JEDINCŮM
RELATE MANY SELECTION (pole)	VZTÁHNOUT VÝBĚR K SKUPINÁM

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Web server

Web služby, Popis
Web služby, Nastavení
Web služby, Poprvé (Část I)
Web služby, Poprvé (Část II)
Web služby, Bezpečnost připojení
Metoda databáze Při ověření WEB
Web služby, Procesy Web spojení
Metoda databáze Při Web spojení
Web služby, Web server nastavení
Web služby, Nekontextní mód
Web služby, Podpora HTML
Web služby, Zahrnutí HTML a JavaScriptu
Textové parametry předávané do 4D Metod volaných přes URL
Web služby, Informace o Web site

START WEB SERVER	ZAPNOUT WEB SERVER
STOP WEB SERVER	ZASTAVIT WEB SERVER
SET WEB TIMEOUT (vyprší)	NASTAVIT WEB LIMIT
SET HTML ROOT (cestaHTML)	NASTAVIT CESTU K HTML
SET WEB DISPLAY LIMITS (početZáznamů{;početStran{; picRef}})	NASTAVIT POČET K ZOBRAZENÍ WEB
SET HOME PAGE (DomovStránka)	NASTAVIT DOMOVSKOU STRÁNKU
SEND HTML FILE (htmlSoubor)	POSLAT HTML SOUBOR
SEND HTML BLOB (blob; typ{; bezKontextu})	POSLAT HTML BLOB
Web context → Logické	Web kontext
SET HTTP HEADER (poleHTTP)	NASTAVIT ZÁHLAVÍ HTTP
SEND HTTP REDIRECT (url{; *})	POSLAT HTTP PŘESMĚROVÁNÍ
WEB CACHE STATISTICS (stránky; hitů; užití)	STATISTIKA WEB CACHE
OPEN WEB URL (url{; *})	OTEVŘÍT WEB URL

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Zamykání záznamů (kapitola)

Zamykání záznamů	
READ WRITE ({tabulka | *})	ČÍST PSÁT
READ ONLY ({tabulka | *})	POUZE ČÍST
Read only state ({tabulka}) → Logické	Stav pouze číst
LOAD RECORD ({tabulka})	ZAVÉST ZÁZNAM
UNLOAD RECORD ({tabulka})	VYVÉST ZÁZNAM
Locked ({tabulka}) → Logické	Zamčeno
LOCKED ATTRIBUTES ({tabulka; }proces; uživatel; stroj; NázevProcesu)	VLASTNOSTI UZAMČENÍ

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Záznamy

DISPLAY RECORD ({tabulka})	ZOBRAZIT ZÁZNAM
CREATE RECORD ({tabulka})	VYTVOŘIT ZÁZNAM
DUPLICATE RECORD ({tabulka})	DUPLIKOVAT ZÁZNAM
Is new record ({tabulka}) → Logické	Je novým záznamem
Modified record ({tabulka}) → Logické	Záznam upraven
Is record loaded ({tabulka}) → Logické	Je záznam zaveden
SAVE RECORD ({tabulka})	ULOŽIT ZÁZNAM
DELETE RECORD ({tabulka})	VYMAZAT ZÁZNAM
Records in table ({tabulka}) → Číslo	Záznamů v tabulce
Record number ({tabulka}) → Číslo	Číslo záznamu
GOTO RECORD ({tabulka; }záznam)	JÍT NA ZÁZNAM
Sequence number ({tabulka}) → Číslo	Pořadové číslo
O číslech záznamů	
PUSH RECORD ({tabulka})	VSUNOUT ZÁZNAM
POP RECORD ({tabulka})	VYSUNOUT ZÁZNAM
Používání paměti Stack pro záznam	

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Zdroje (kapitola)

Zdroje
Zdroje a 4D Insider: Příklad

Open resource file (resSoubor; SouborTyp) → DocRef	Otevřít zdrojový soubor
Create resource file (resSoubor; SouborTyp) → DocRef	Vytvořit zdrojový soubor
CLOSE RESOURCE FILE (resSoubor)	ZAVŘÍT ZDROJOVÝ SOUBOR
RESOURCE TYPE LIST (resTypy; resSoubor)	SEZNAM TYPŮ ZDROJŮ
RESOURCE LIST (resTyp; resID; resNázvy; resSoubor)	SEZNAM ZDROJŮ
STING LIST TO ARRAY (resID; strArray; resSoubor)	SEZNAM ŘETĚZCŮ DO ARRAY
ARRAY TO STRING LIST (řatězec; resID; resSoubor)	ARRAY DO SEZNAMU ŘETĚZCŮ
Get indexed string (resID; strID; resSoubor) → Řetězec	Získat indexovaný řetězec
Get string resource (resID; resSoubor) → Řetězec	Získat řetězec ze zdroje
SET STRING RESOURCE (resID; resData; resSoubor)	NASTAVIT ŘETĚZEC DO ZDROJE
Get text resource (resID; resSoubor) → Text	Získat text ze zdroje
SET TEXT RESOURCE (resID; resData; resSoubor)	NASTAVIT TEXT DO ZDROJE
GET PICTURE RESOURCE (resID; resData; resSoubor)	ZÍSKAT OBRÁZEK ZE ZDROJE
SET PICTURE RESOURCE (resID; resData; resSoubor)	NASTAVIT OBRÁZEK DO ZDROJE
GET ICON RESOURCE (resID; resData; resSoubor)	ZÍSKAT IKONU ZE ZDROJE
GET RESOURCE (resTyp; resID; resData; resSoubor)	ZÍSKAT OBSAH ZDROJE
SET RESOURCE (resTyp; resID; resData; resSoubor)	NASTAVIT DO ZDROJE
Get resource name (resTyp; resID; resSoubor) → Řetězec	Získat název zdroje
SET RESOURCE NAME (resTyp; resID; resNázev; resSoubor)	NASTAVIT NÁZEV ZDROJE
Get resource properties (resTyp; resID; resSoubor) → Číslo	Získat vlastnosti zdroje
SET RESOURCE PROPERTIES (resTyp; resID; resVlastn; resSoubor)	NASTAVIT VLASTNOSTI ZDROJE
DELETE RESOURCE (resTyp; resID; resSoubor)	VYMAZAT ZDROJ

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Kódy chyb

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ASCII kódy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Syntaxe příkazů

Předmluva

Příkazy a odkazy pro Úvod

Verze 6.0

4th Dimension obsahuje vlastní programovací jazyk. Tento vestavěný jazyk, obsahující přes 500 příkazů, dělá z 4th Dimension silný vývojářský nástroj pro databázové aplikace na osobních počítačích. Můžete použít 4th Dimension k mnoha rozdílným úkolům - od plnění různých výpočtů až po vlastní kompletní uživatelské rozhraní. Můžete například

- Programově zpřístupnit všechny editory dostupné v prostředí uživatele,
- Vytvořit a vytisknout úplné zprávy a štítky z informací v databázi,
- Komunikovat s jinými zařízeními,
- Řídit dokumenty,
- Importovat a exportovat data mezi databázemi 4th Dimension a jinými aplikacemi,
- Připojit procedury napsané v jiném jazyce do programovacího jazyka 4th Dimension.

Přizpůsobivost a síla programovacího jazyka 4th Dimension z něj udělala ideální nástroj pro všechny úrovně uživatelů a vývojářů k vytvoření kompletního rozsahu pro řízení informací. Základní uživatelé mohou rychle vytvářet různé výpočty. Pokročilí uživatelé bez zkušeností s programováním mohou upravovat své vlastní databáze. Zkušení vývojáři mohou použít tento silný programovací jazyk k předání různých vlastností a možností do jejich databází, obsahující převody souborů a komunikaci. Vývojáři se zkušenostmi z jiných programovacích jazyků mohou vkládat své vlastní příkazy do jazyka 4th Dimension.

Jazyk 4th Dimension se automaticky rozšíří při instalaci některého modulu. Každý z modulů obsahuje příkazy, které jsou specifické pro funkce které může provádět.

O manuálech

Tyto manuály popisují práci s 4th Dimension a 4D Server. Jediná výjimka je Příručka 4D Server, která popisuje možnosti 4D Serveru, které jsou pouze v balíku 4D Server.

Popis jazyka popisuje veškeré příkazy programovacího jazyka 4th Dimension. V této příručce se naučíte jak používat programovací jazyk 4th Dimension.

Příručka návrháře popisuje práci v Prostředí návrháře a provádí vás všemi operacemi, které můžete dělat v tomto prostředí. Tuto příručku můžete používat ve spojení s jinými částmi vaší dokumentace.

Příručka uživatele popisuje Prostředí uživatele - prostředí, ve kterém zadáváte a měníte data vaší databáze.

Objevování 4th Dimension vás provede příklady, ve kterých si vytvoříte a vyzkoušíte databáze 4th Dimension. Tyto příklady vám umožní lépe se seznámit s prostředím 4th Dimension a způsobem ovládání tohoto programu.

Příručka 4D server obsahuje popis řízení víceuživatelské databáze pomocí 4D Serveru. Tato příručka je obsažena pouze v balíku 4D Server.

O tomto manuálu

Tento manuál popisuje jazyk 4th Dimension. Záleží na tom, jestli jste seznámeni s výrazy jako je tabulka, pole, formulář, atd. Před tím, než budete číst tento manuál, by jste měli:

- Použít manuál k Příkladům 4th Dimension a trochu se s nimi seznámit.
- Začít vytvářet novou databázi s pomocí manuálu návrháře,
- Seznamte se s řízením prostředí uživatele. K tomu slouží příručka uživatele.

Kam jít nyní?

Pokud tento manuál čtete poprvé, přečtěte si část Úvod.

Příkazy a odkazy pro Úvod

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Licenční ujednání

Příkazy a odkazy pro Úvod

Verze 6.5

ACI PRODUCT LINE
LICENČNÍ DOHODA

UPOZORNĚNÍ !

Měli by jste si pozorně přečíst všechny termíny a podmínky této dohody předtím, než budete pokračovat v instalaci a používání Softwaru. Tento dokument je licenční dohoda o užití díla mezi Analyses Conseils Informations - A.C.I. (dále ACI), francouzskou veřejnou společností a Vámi (dále držitel licence). Držitel licence klepnutím na tlačítko "Ano" nebo "Yes", souhlasí s dohodou a je od té doby vázán dodržovat všechny podmínky a termíny této dohody.

Jestliže nesouhlasíte s těmito termíny a podmínkami, navraťte prosím okamžitě média obsahující Software, spolu s obaly, organizaci, v které jste je získali a obdržíte úplnou náhradu.

Tato dohoda je uzavírána pouze pro Software a dokumentaci, pro který byla licence zakoupena a pro který držitel licence obdržel autorizované sériové číslo.

ACI si přeje licenzovat Software a Dokumentaci pouze v případě, jestliže držitel licence přijímá všechny termíny a podmínky této licenční dohody.

1. VLASTNICTVÍ A LICENCE

Tato dohoda je licenční dohoda o užití díla a nikoliv kupní smlouva.

ACI dále vlastní všechny kopie počítačových souborů obsažené v balíčku a všechny další kopie, které je držitel licence oprávněn vytvořit v souladu s dohodou.

Práva držitele licence jsou určena touto dohodou a ACI si dále ponecháva všechna práva, která nejsou výslovně poskytnuta držiteli licence touto dohodou.

2. POSKYTNUTÁ PRÁVA

ACI poskytuje držiteli licence osobní a nevýhradní právo užívat počítačový program obsažený na médiu zakoupeném držitelem licence a to pouze ten, pro který držitel licence zakoupil licenci a pro který obdržel autorizované sériové číslo a odpovídající dokumentaci v elektronické formě (dále Software). Odpovídající soubory elektronické dokumentace jsou dále uváděny pouze jako Dokumentace.

A. PRÁVA K SOFTWARE

ACI poskytuje držiteli licence, pro konkrétní druh počítače (platformu) a počet současně připojených uživatelů zmíněných na štítku média nebo v přiložených materiálech, následující práva po dobu a za podmínek uvedených níže:

"Jednouživatelský" znamená, že licenzovaný Software je samostatný produkt pouze pro jeden samostatný počítač a

jednu instalaci.

"Víceuživatelský" znamená, že licenzovaný Software je produkt pro více uživatelů v síti počítačů a licenční poplatek je stanoven na základě počtu současně připojených uživatelů.

"Počet současně připojených uživatelů" znamená maximální počet uživatelů, schválených držitelem licence, používajících stejný software v jednom místě a ve stejnou dobu, a je to počet za který ACI obdržela licenční poplatek od držitele licence.

DRŽITEL LICENCE smí:

Jednouživatelský:

a) Instalovat a používat Software na libovolném stolním počítači zajistí-li, že Software je používán pouze na jednom stolním počítači v daný časový okamžik.

b) Přenášet software z jednoho počítače na jiný zajistí-li, že Software je používán pouze na jednom stolním počítači v daný časový okamžik.

Víceuživatelský:

a) Instalovat a používat Software výhradně pro jeden počítač ukládající a zpracovávající data (počítač serveru) a na počítačích klientských v jedné síti klientů a serveru.

b) Přenášet software z jednoho počítače serveru na jiný zajistí-li, že Software je používán pouze na jednom počítači serveru v daný časový okamžik.

c) Používat Software na tolika klientských počítačích kolik jich potřebuje k uspokojení svých potřeb zajistí-li, že v žádný časový okamžik nebude překročen maximální počet současně připojených uživatelů.

Jednouživatelský i víceuživatelský:

a) Přenést Software na pevný disk pouze za účelem použití popsaným v předchozím zajistí-li, že originální médium nebude používáno na jiném počítači v tutéž dobu a zadá-li okamžitě číslo původní licence.

b) Provést kopii Softwaru ve spustitelném stavu pro účely zálohování a archivace, zajistí-li že budou kopírovány všechny znaky autorství, vlastnictví, práv a obchodní značky, které jsou obsaženy na a v Softwaru.

c) Převést všechna práva držitele licence na třetí stranu zajistí-li, že tento převod bude obsahovat i tuto dohodu, software bude zahrnovat všechny kopie, vylepšené verze i předchozí verze a veškerou dokumentaci a zajistí-li, že současnému držiteli licence nezbude žádná kopie softwaru. V tomto případě převod zahrnuje ukončení poskytování licenčních práv současnému držiteli licence, současný držitel licence dále nebude mít žádné právo požívat Software a přestane být oprávněným držitelem licence.

DRŽITEL LICENCE nesmí:

Jednouživatelský:

a) Instalovat Software v síti počítačů. (Jestliže si držitel přeje používat Software na více než jednom stolním počítači, musí v tomto případě zakoupit dodatečnou licenci, v ceně odpovídající standardním licenčním poplatkům ACI v době vyřizování objednávky.)

b) Používat Software k vytvoření víceuživatelských aplikačních serverů a datových serverů, kromě případu, že byla držiteli licence poskytnuta licence 4D WEB EXTENSION nebo 4D MAEL a zaplatil patřičný licenční poplatek.

Víceuživatelský:

Přenášet Software na jiný druh počítače (platformu) nebo pod jiný druh operačního systému a nebo používat Software pro více současně připojených uživatelů než odpovídá licenci. (Držitel licence musí zaplatit licenční poplatek odpovídající standardním licenčním poplatkům ACI v době takovéhoto přenosu nebo zvýšení počtu současně připojených uživatelů.)

Jednouživatelský i víceuživatelský:

a) Pronajímat Software třetí straně nebo sdílet jej s třetí stranou nebo poskytnout jakýkoliv druh práv k Softwaru či jeho libovolné části (vyjma práv v článku 2 této dohody) v jakékoliv formě třetí straně, bez předchozího písemného souhlasu ACI, který bude-li poskytnut je vázán na souhlas třetí strany s podmínkami a termíny této dohody.

b) Přenášet Software na jinou počítačovou platformu, nebo operační systém (Držitel licence musí zaplatit licenční poplatek odpovídající standardním licenčním poplatkům ACI v době takovéhoto přenosu)

c) Upravovat, překládat, zpětně sestavovat, dekompilovat, rozkládat Software, vytvářet odvozené práce na základě částí nebo celku Software, kromě již zmíněných zákonných způsobů.

d) Odstranit, nebo nahradit jakoukoliv identifikaci Softwaru, vlastnické poznámky, štítky nebo obchodní označení, které se vyskytují na nebo v Softwaru.

e) Užívat záložní a archivní kopii (nebo dovolit komukoliv užít tuto kopii) pro jakékoliv účely odlišné od původního záměru kopie, t.j. nahradit originální kopii v případě jejího zničení nebo poškození.

f) Zveřejnit výsledky jakýchkoli porovnávacích nebo jiných testů Softwaru bez předchozího písemného souhlasu ACI.

B. VYJÍMKY

- 4D ENGINE: Držitel licence smí pouze, bez omezení, užívat Software výhradně k spouštění a zahrnutí do kompilovaných aplikací vyvinutých ve 4D (4D aplikací), určených pro vlastní potřebu, nebo k distribuci. .
V žádném případě nesmí být tento Software používán k zahrnování do jiných než 4D aplikací a používán mimo ně.

- 4D RUNTIME CLASSIC, 4D SERVER APPLICATION: Software smí být používán pouze pro účely spouštění jedné či více 4D aplikací vyvinutých ve 4D a jako jejich součást (nebo interpret, v případě 4D RUNTIME CLASSIC). V žádném případě, nesmí být Software používán k vývoji nových aplikací, nebo pro účely řízení jiných databází.

- 4D WEB EXTENSION / 4D SERVER WEB EXTENSION: V souladu s touto dohodou může držitel licence použít software jako Intranet/Internet Server, včetně ”Web” funkcí celé 4D Product Line, definovaných v dokumentaci, bez omezení počtu připojení, za předpokladu, že souhlasí, že poskytnutá licence je omezena na jeden výskyt Softwaru, to znamená, že je omezena na jednu 4D Aplikaci spuštěnou , pomocí některého z produktů 4D Product Line, používaného rovněž v souladu s touto licenční dohodou.

- 4th DIMENSION MULTI ACCESS EXTENSION LICENCE (4D MAEL): V souladu s touto dohodou může

držitel licence používat jednouživatelskou verzi k zahrnutí do 4D alikací, jako víceuživatelské aplikační servery nebo datové servery pro Intranet/Internet servery . Držitel licence musí získat tolik licencí 4D MAEL, kolik je současně spuštěných 4D Aplikací na jednom počítači.

C. PRÁVA K ELEKTRONICKÉ DOKUMENTACI

DRŽITEL LICENCE smí:

a) Tisknout elektronickou dokumentaci za účelem používání spolu se Software.

b) Přemístit soubory HTML na server za účelem používání v síti Intranet držitele licence.

c) Přemístit elektronickou dokumentaci na pevný disk držitele licence za účelem používání spolu se Software.

DRŽITEL LICENCE nesmí:

a) Distribuovat dokumentaci.

b) Přemístit dokumentaci jakýmkoliv způsobem tak, že tato bude dostupná v síti Internet.

D. DALŠÍ PRÁVA

Software může obsahovat jednu nebo více knihoven, další soubory a položky, které slouží k usnadnění užívání Software. ACI uděluje držiteli licence práva užívat tyto knihovny, soubory či položky za stejných podmínek specifikovaných v této dohodě a za dalších podmínek specifikovaných přímo pro knihovny a další soubory. Držitel licence by si měl přečíst soubor "Read me" obsažený v Software, aby se seznámil s dalšími informacemi a podmínkami.

3. TECHNICKÁ PODPORA A PRÁVA NA VYLEPŠENÉ VERZE

Jestliže si držitel licence přeje v budoucnu získat technickou podporu a právo na vylepšené verze k Software ACI, musí NEPRODLENĚ REGISTROVAT SOFTWARE prostředky, které jsou k registraci určeny.

Jestliže držitel licence zakoupí vylepšenou verzi Software je stále držitelem pouze jedné licence produktu, který je držiteli licence tímto aktem pouze vylepšen.

Jesliže tento balíček je vylepšení, držitel licence musí potvrdit všechny podmínky této dohody pro novou verzi, která nahrazuje předchozí verzi Software a tato dohoda nahrazuje předchozí dohodu.

4. OMEZENÍ ZÁRUKY A POVINNOSTÍ

ACI zaručuje výlučně pouze, že

- Software je nahrán na médium prostém vad materiálu a zaručuje používání v normálních podmínkách

- Software bude schopen provádět podstatné funkce popsané v dokumentaci

Výše uvedená záruka je platná po dobu 90 (slovy devadesáti) dnů od data dodání prokázaného dodacím listem, a to v případě, že držitel licence vrátí Software do místa kde jej získal.

V případě vad je úplná povinnost ACI a úplná náprava pro držitele licence jedna z následujících možností a to podle

volby ACI:

a) náhrada média, s podmínkou, že na nahrazený Software se vztahuje záruka po zbytek záruční doby od původního data

b) navrácení licenčního poplatku zaplaceného za software a ukončení této dohody.

Záruka se nevztahuje na případy, kdy vady vznikly nepředvídatelnou událostí, hrubým zacházením, při úpravách Softwaru nebo nesprávným užitím, v tomto případě ACI nemá povinnost nahradit médium ani vrátit licenční poplatek.

Poskytovaná záruka, definovaná v předchozím, je jediná a všechny ostatní záruky ať již vyslovené nebo předpokládané včetně, ale ne pouze, záruk na chování, obchodovatelnost, vhodnost pro jakýkoliv určitý účel a záruk o nezasahování do práv třetích stran, nejsou poskytovány.

Výslovně ACI nezaručuje, že funkce obsažené v Software budou splňovat požadavky držitele licence a že běh bude nepřerušený a bezchybný. Držitel licence nese plné riziko pokud jde o užití, kvalitu a chování programu.

Ani ACI ani kdokoliv jiný, kdo byl účasten při navrhování, vytváření, výrobě nebo distribuci softwaru neopovídá za jakékoliv náhodné, následné, přímé či nepřímé škody způsobené držiteli licence, jakémukoliv uživateli nebo třetí straně, včetně, ale ne pouze, náhrady ušlého zisku, ztráty údajů nebo dalších vynaložených nákladů vzniklých užitím Softwaru, nebo nemožnosti jej užívat. Toto omezení bude uplatněno dokonce i tehdy byla-li ACI upozorněna na možnost takovéto škody.

Rozhodně a v žádném případě odpovědnost ACI nemůže přesáhnout částku zaplacenou za poskytnutí licence již se tato dohoda týká.

Předchozí se nevztahuje a není na úkor zákonných práv držitele licence.

5. DUŠEVNÍ VLASTNICTVÍ

Software je duševní vlastnictví ACI a nebo jejích dodavatelů a je chráněn Francouzským autorským zákonem, a vzhledem k mezinárodním dohodám, odpovídajícími zákony země, kde bude software užíván.

Obchodní značky, loga a obchodní názvy citované v této dohodě, na nebo v Software jsou vlastnictvím jejich držitelů.

Jakékoliv rozmnožování částí nebo celého Software je povoleno pouze tehdy, když rozmnoženiny budou obsahovat všechny zákonné znaky vlastnictví tohoto Software.

6. DŮVĚRNOST

Struktura a organizace Software je významné obchodní tajemství ACI a jejích dodavatelů. Držitel licence nesmí toto obchodní tajemství prozradit.

Držitel licence výslovně souhlasí, že ACI má právo zveřejnit licenční vztah ACI a držitele licence.

Povinnost neprozratzovat obchodní tajemství zůstává v platnosti i po ukončení této Licenční dohody.

7. TERMÍNY A UKONČENÍ

Tato dohoda je v platnosti do jejího ukončení.

Držitel licence může ukončit platnost dohody kdykoliv, bez předchozí písemné výpovědi. Toto ukončení neosvobozuje držitele licence od jeho odpovědnosti vyplývající z dohody v době její platnosti před datem ukončení.

Jesliže držitel licence nedodrží jakýkoliv bod této dohody, tato dohoda a veškerá poskytnutá práva užívat Software jsou automaticky ukončena. Toto ukončení dohody nebrání ACI uplatnit nárok na náhradu jakékoliv škody.

Při ukončení dohody z jakéhokoliv důvodu je držitel licence povinen zničit nebo vrátit ACI Software a jakoukoliv kopii ať celého Software, nebo jeho části.

Držitel licence je povinen stvrdit písemným zápisem podepsaným zákonným zástupcem, že ustanovení tohoto odstavce byla dodržena v termínu 30 (slovy třiceti) dnů od data ukončení dohody.

8. VŠEOBECNÁ OPATŘENÍ

Držitel licence nesmí přímo či nepřímo přenést Software do jakékoli země, do které by takovýto přenos mohl být zakázán libovolným použitelným zákonem řídícím vývoz.

Žádné změny nebo úpravy této dohody nejsou platné pokud nejsou v písemné formě a stvrzeny podpisy držitele licence a určené úřední osoby zastupující ACI.

Bude-li libovolné ustanovení této dohody prohlášeno za neúčinné na základě platného zákona či soudního rozhodnutí, zbývající ustanovení této dohody zůstanou v plné právní síle a účinnosti.

Vzdání se nároku ze strany ACI při nějakém případném porušení této dohody nebo při případném nesplnění povinností vyplývajících z této dohody nezakládá žádné další vzdání se nároku pří libovolném dalším porušení dohody nebo při nesplnění vyplývajících povinností.

Toto je úplná a nezkrácená verze dohody mezi ACI a držitelem licence týkající se obsahu programového balíku. Tato dohoda nahrazuje jakákoliv jiná sdělení ohledně tohoto programového balíku vyjádřená v objednávkách, přímé komunikaci, reklamě i jiným způsobem.

Vztah mezi ACI a držitelem licence je vztahem mezi poskytovatelem licence a jejím nabyvatelem. Ve všech záležitostech týkajících se této dohody je držitel licence považován za nezávislý právní subjekt.

Tato dohoda je upravována platnými zákony země dle sídla ACI, její pobočky nebo dle sídla distributora. Jakékoliv spory a uplatňované nároky vzniklé z této dohody budou řešeny nejdříve těmito soudy před kompetentním soudem země, v které držitel licence získal software.

Ve Spojených státech, příslušný soud je Krajský soud v Santa Clara stát Kalifornie, Santa Clara County.

DRŽITEL LICENCE POTVRZUJE, ŽE DOHODU PŘEČETL, POROZUMĚL JÍ A SOUHLASÍ SE ZÁVAZKY, TERMÍNY A PODMÍNKAMI VYPLÝVAJÍCÍMI Z TÉTO DOHODY.

Máte-li libovolné dotazy týkající se této dohody, nebo požadavek na rozšíření informací, kontaktujte prosím ACI (33) 1 40 87 92 00 nebo pobočku ACI příslušnou vaší zemi.

* Upozornění pro konečné uživatele ve státní správě Spojených států:

Užívání, kopírování nebo zveřejňování ve státní správě Spojených států je upraveno paragrafem c) (1) (ii) doložky Práva v oblasti technických dat a počítačových programů (the Rights in Technical Data and Computer Software clause) v 252.227-7013.

Příkazy a odkazy pro Úvod

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Úvod

Příkazy a odkazy pro Úvod

Verze 6.0

Tato část popisuje programovací jazyk 4th Dimension. Popisuje následující části:

• Co je jazyk a co pro vás může dělat,
• Jak používat metody,
• Jak vyvíjet aplikaci pod 4th Dimension.

Tyto části jsou zde diskutovány pouze obecně. Podrobnější informace najdete v dalších částech manuálu.

Co je Jazyk?

Jazyk 4th Dimension není příliš odlišný od mluveného jazyka, který používáme každý den. Je to způsob komunikace použitý ke sdělení požadavků, informací a instrukcí. Jako mluvený jazyk, má i jazyk 4th Dimension svůj vlastní slovník, gramatiku a syntaxe. Říkáte jím 4th Dimension jak má provést jednotlivé operace s daty vaší databáze.

K tomu, aby jste mohli pracovat efektivně s jazykem 4th Dimension, jej nepotřebujete znát celý. V normální mluvě také nepotřebujete znát všechna slovíčka, ale stačí vám určitý počet a můžete s ním vystačit. Jazyk 4th Dimension je naprosto stejný - také vám stačí znát pouze část a vaše práce bude produktivní. Další části se můžete doučit, když je budete potřebovat.

Proč použít Jazyk?

Na první pohled se může zdát, že je malá potřeba programovacího jazyka ve 4th Dimension. Prostředí návrháře a uživatele obsahují mnoho nástrojů, které se obejdou bez programování. Základní operace jako vyhledávání, třídění, zadávání dat a zprávy jsou řízeny velmi jednoduše. Samozřejmě máte k dispozici i mnoho dalších vlastností jako upravování a kontrola dat, vytváření diagramů a štítků, atd.

Pak tedy proč potřebujeme Jazyk 4th Dimension? Zde jsou některé důvody:

• Automatické opakující se úlohy: mohou to být úpravy dat, vytváření komplexních zpráv a různé další operace s daty,
• Kontrola uživatelského rozhraní: můžete řídit okna a nabídky a řídit vzhled formulářů a jednotlivých objektů,
• Plnit složitější řízení dat: Tyto úlohy obsahují transakce procesů, kompletní kontrola dat, řízení více uživatelů, nastavení a výběrové operace,
• Řídit počítač: můžete řídit komunikace sériového vstupu, řízení dokumentů a chyb,
• Vytváření aplikací: můžete vytvořit snadno ovladatelné databáze, které budou využívat prostředí Vlastních nabídek,
• Vložit funkce do vestavěných Web služeb: vytvořit dynamické HTML stránky které budou automaticky převedeny z formulářů 4th Dimension.

Jazyk vám umožní kompletní kontrolu nad návrhem a operacemi vaší databáze. Zatím co prostředí uživatele vám nabízí výkonné obecné nástroje, jazyk vám umožní přizpůsobit databázi vašim potřebám

Řízení vašich dat

Jazyk 4th Dimension vám umožňuje úplně řídit vaše data způsobem, který je současně účinný i elegantní. Jazyk je dostatečně snadný pro začátečníky a dostatečně promyšlený pro vývojáře aplikací. Umožňuje snadný přechod od

obecného řízení databáze ke zcela vlastnímu ovládání.

Příkazy jazyka 4th Dimension zpřístupňují editory prostředí uživatele, které již znáte. Pokud například použijete příkaz QUERY (dotaz), bude zobrazen Editor vyhledávání. Použití tohoto příkazu jazyka je stejně jednoduché jako volby položky Dotaz v nabídce Dotazy, ale příkaz QUERY je užitečnější. Pokud chcete, můžete v příkazu QUERY určit, co chcete hledat. Například příkaz QUERY([Lidé]Příjmení="Kovářová") najde všechny osoby s příjmením Kovářová.

Jazyk 4th Dimension je velmi výkonný - jeden příkaz často nahrazuje stovky nebo dokonce tisíce řádků programu napsaného v tradičním programovacím jazyku. Výkon je však překvapivě doprovázen i jednoduchostí. Příkazy mají jednoduché názvy. Například pro splnění dotazu můžete použít příkaz QUERY a pro přidání nového záznamu příkaz ADD RECORD (přidat záznam).

Jazyk je navržen tak, aby v něm šly snadno splnit veškeré úlohy. Přidávání záznamů, třídění, vyhledávání a podobné operace jsou značeny jednoduchým a jasným příkazem. Jazyk ale může řídit i výstupy počítače, číst dokumenty z disku, řídit složité transakce a daleko více.

Jazyk 4th Dimension dokáže plnit složité úlohy pomocí jednoduchého programování. Plnění těchto úloh bez použití jazyka může být pro většinu nepředstavitelné.

Stejně jako výkonné příkazy jazyka mohou být i některé úlohy komplexní a složité. Nástroj sám o sobě neprovede úlohu, ale může zjednodušit její provedení. Například textový procesor vám usnadní psaní knížky, ale nenapíše ji za vás.

Je to tradiční počítačový jazyk?

Pokud jste seznámeni s tradičními počítačovými jazyky, nemusí pro vás tato část být zajímavá. Můžete ji přeskočit.

Jazyk 4th Dimension není tradiční počítačový jazyk. Je jedním z nejpřínosnějších a nejvariabilnějších jazyků dostupných na současných počítačích. Jazyk byl navržen tak, abyste mohli pracovat vašim obvyklým způsobem.

Při použití tradičních jazyků musíte pečlivě plánovat. Obvykle je plánování nejvýznamější částí vývoje. 4th Dimension vám umožňuje použít jazyk v kterémkoli okamžiku a v kterékoli části vaší databáze. Můžete začít doplněním metody objektu do formuláře a později přidat několik metod. Když se vaše databáze stane složitější, můžete doplnit metody projektu ovládané z nabídek. Můžete použít právě tolik jazyka, kolik potřebujete. Neznamená to "všechno nebo nic", jako u mnoha jiných databází.

Tradiční jazyky vyžadují, abyste předem formálně definovali použité objekty. Ve 4th Dimension objekt, například tlačítko, jednoduše vytvoříte a použijete. 4th Dimension za vás objekt automaticky zpracuje. Například při použití tlačítka jej pouze nakreslíte ve formuláři a pojmenujete. Pokud uživatel klepne na tlačítko, jazyk to automaticky oznámí vašim metodám.

Tradiční jazyky jsou často velmi nepružné a vyžadují, aby příkazy byly uváděny podle přesných formálních omezení. Jazyk 4th Dimension tuto tradici porušuje k vašemu prospěchu.

Metoda - brána jazyka

Metoda je řada instrukcí, podle kterých 4th Dimension provádí daný úkol. Jednotlivým řádkům v metodě říkáme příkazy. Příkazy jsou tvořeny prvky jazyka.

Protože jste prošli Učební lekce 4th Dimension (nebo snad ne?), již jste napsali a použili metody.

Ve 4th Dimension můžete vytvořit pět typů metod:

- **Metody objektu**: Obvykle krátké metody použité k řízení objektů ve formulářích,
- **Metody objektu**: Řídí zobrazení a tisk formuláře,

• **Metody tabulky/Triggery**: Používají se k vnucení pravidel do vaší databáze,
• **Metody objektu:** Metody použitelné ve všech částech databáze. Například mohou být přiřazeny k položkám nabídek,
• **Metody databáze:** Provádí se při spuštění nebo zavírání databáze, nebo při spojení a odpojení Web prohlížeče pokud je vaše databáze publikována jako Web server na internetu nebo intranetu.

Následující část popisuje každý z těchto typů metod a dá vám příklady jak je použít.

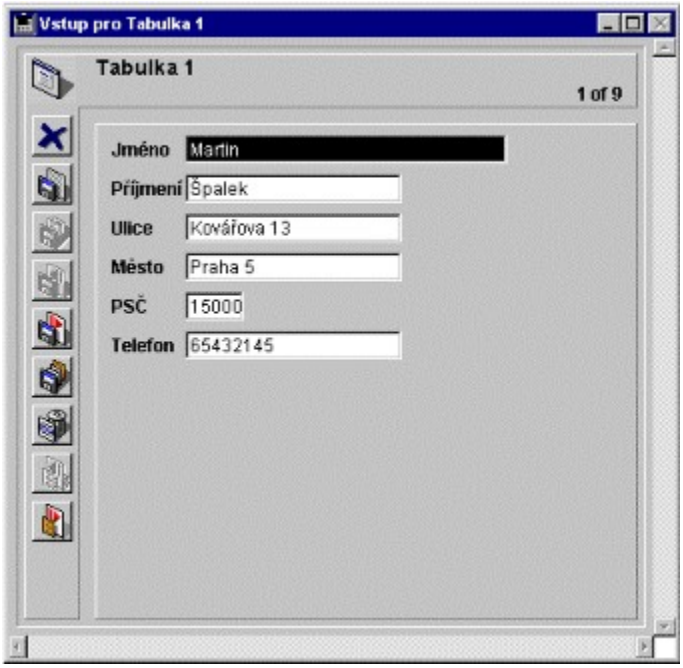
Začínáme s metodou objektu

Každý objekt, který plní určitou akci (to znamená všechny aktivní objekty) mohou mít přiřazenu metodu. Metody objektu mohou kontrolovat a řídit aktivní objekty během vstupu a tisku dat. Metoda objektu se bude s objektem přesunovat i během kopírování a duplikování. Toto vám umožňuje vytvářet objekty použitelné v síti. Metodu objektu můžete kontrolovat kdykoli potřebujete.

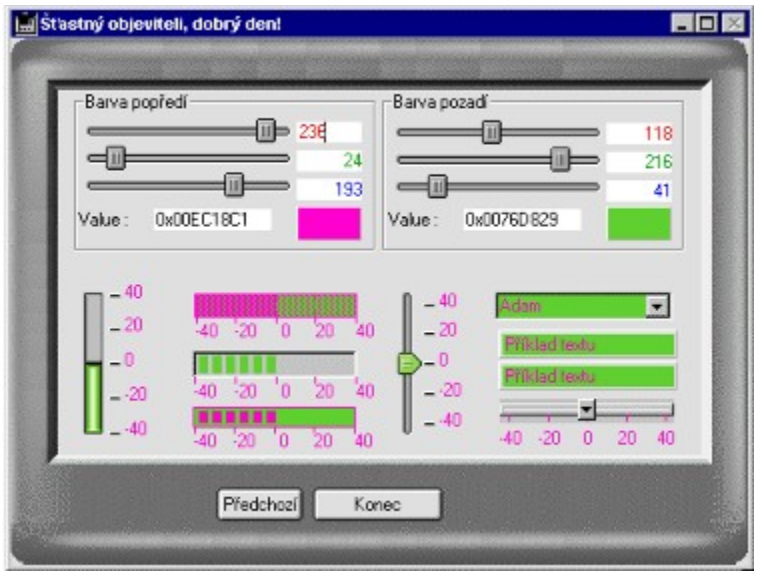
Metody objektu jsou hlavní nástroj pro řízení uživatelského rozhraní, což je brána do vaší databáze. Rozhraní uživatele obsahuje veškeré nástroje pomocí kterých komunikuje počítač s uživatelem. Dobré je vytvořit rozhraní uživatele tak jednoduché a pochopitelné, jak je to možné. Rozhraní uživatele může zpříjemnit práci s počítačem a vaši uživatelé si nebudou stěžovat.

Toto jsou dva základní typy aktivních objektů ve formuláři:
• Pro vkládání, zobrazování a ukládání dat jako pole a podpole
• Pro řízení: dostupné oblasti, tlačítka, posuvné oblasti, hierarchické seznamy, pravítka

4th Dimension vám umožňuje vytvořit klasické formuláře jako je následující ukázka:



Můžete také vytvořit formuláře s více grafickými ovladači, jak je ukázáno dále:



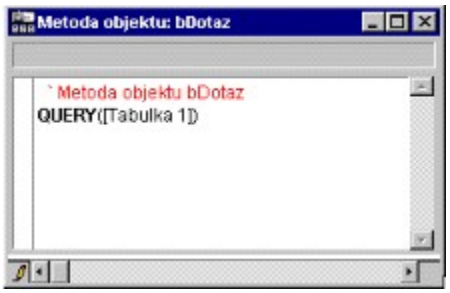
Můžete vytvářet také formuláře, které jsou vzhledem omezeny pouze vaší fantazií:

Bez rozdílu stylu při vytváření formulářů, mají objekty vestavěné vlastnosti jako je kontrola rozsahu a vstupní filtry pro dostupné objekty a automatické akce pro ovladače, nabídky a tlačítka. Tyto vlastnosti vždy použijte při vytváření metod objektu. Vestavěné vlastnosti jsou stejné jako u metod které jsou k nim přiřazeny a aktivují se pouze při použití aktivního objektu. Obvykle se používá kombinace vestavěných vlastností a metod objektu k řízení rozhraní uživatele.

Metody objektu přiřazené k aktivním objektům pro vstup dat obvykle plní různé úlohy specifické pro pole nebo proměnnou. Metoda může provádět úpravu dat, formátování dat nebo výpočty. Může používat i informace ze vztažených tabulek. Některé z těchto úloh mohou samozřejmě být plněny s vestavěnými vlastnostmi pro objekty vstupu dat. Metodu objektu použijte pokud je úloha příliš složitá pro vestavěné vlastnosti. Jestli chcete vědět více informací o vestavěných vlastnostech, přečtěte si *Příručku návrháře 4th Dimension*.

Metody objektu se přiřazují také k ovladačům jako jsou tlačítka. Aktivní objekty použité pro řízení jsou nejdůležitější pro práci s databází. Tlačítka umožňují posunování mezi záznamy, změnu formuláře a předání a vymazání dat. Tyto aktivní objekty zjednodušují ovládání databáze a zkracují čas potřebný k naučení této databáze. Tlačítka také obsahují vestavěné vlastnosti a stejně jako u objektů pro vstup dat, je můžete použít před tím, než použijete metody objektu. Metody objektu vám umožňují přiřadit akce, které nejsou standardní.

Například následující okno ukazuje metodu objektu, která zobrazí Editor vyhledávání.



Jakmile budete mít větší zkušenosti, přijdete na to, že si můžete vyvořit knihovnu objektů s přiřazenými metodami. Tyto objekty můžete i s jejich metodami kopírovat mezi formuláři, tabulkami nebo databázemi. Nakopírujete je do schránky a můžete je vložit kamkoliv jinam.

Řízení formuláře pomocí metody formuláře

Stejně jako jsou metody objektu spojeny s aktivními objekty, jsou i metody formuláře spojeny s formulářem. Každý formulář může mít jednu metodu formuláře. Ve formuláři můžete zobrazovat, vkládat a tisknout data. Formuláře umožní uživateli zobrazit základní data různými způsoby. Pomocí formulářů můžete vytvářet působivé a snadno ovladatelné vstupní formuláře a tištěné zprávy. Metoda formuláře sleduje a řídí použití formuláře pro vstup dat i pro tisk.

Metoda formuláře řídí formulář na vyšší úrovni než metody objektu. Metody objektu jsou spuštěny pouze při použití objektu, zatím co metoda formuláře je provedena při použití jakéhokoli prvku formuláře. Metoda formuláře obvykle slouží k řízení komunikace mezi různými objekty a formulářem jako celkem.

Protože formuláře jsou používány mnoha různými způsoby, je vhodné sledovat, co se děje při použití daného formuláře. K tomu slouží **události formuláře**. Ty oznamují, co se právě děje ve formuláři. Každá událost (klepnutí, poklepání, aktivace...) spustí nebo zastaví metodu formuláře, stejně jako metody objektu u každého objektu ve formuláři.

Jestli chcete vědět více informací o formulářích, objektech, událostech a metodách přečtěte si část Události formuláře.

Vnucení pravidel vaší databázi s použitím metod tabulek/triggerů

Trigger je přiřazen k tabulce. Z tohoto důvodu se nazývají metodou tabulky. Triggery jsou automaticky prováděny kdykoli pracujete se záznamy tabulky (vkládání, mazání, upravení nebo načtení). Triggery jsou metody, které zabraňují "nezákonným" operacím se záznamy vaší databáze. Například v systému faktur můžete každému zabránit ve vytvoření faktury bez definování zákazníka pro kterého je faktura. Trigger je velmi silný nástroj k odlišení operací v tabulce stejně jako k předejití ztráty nebo porušení dat. Můžete zpočátku napsat jednoduchý trigger a později jej upravovat a dodělávat.

Jestli chcete vědět více informací o triggerech, přečtěte si část Triggery.

Použití metod projektu v celé databázi

Na rozdíl od metod objektu, formuláře nebo tabulky, které jsou vázány k příslušnému objektu, formuláři nebo tabulce, jsou metody projektu použitelné v celé databázi. Metody projektu jsou použitelné vícekrát a dostupné ze všech ostatních metod. Pokud potřebujete zopakovat některé úlohy, nemusíte metodu psát pokaždé znovu. Můžete se odkazovat na metodu projektu, kdekoli potřebujete - z jiné metody projektu nebo z metody objektu, formuláře nebo tabulky. Pokud použijete metodu projektu v určitém místě jiné metody, je to jako by jste tuto metodu napsali do místa odkud ji voláte. Metody projektu, které jsou volané jinými metodami jsou nazývány "podprogramy".

Je ještě jeden způsob použití metod projektu - jejich přiřazení k položkám nabídek. Pokud metodu přiřadíte k položce nabídky, je tato metoda spuštěna při použití této položky. Položky nabídek můžete považovat za odkazy k metodám projektu.

Řízení databáze pomocí metod databáze

Stejně jako metody objektu nebo formuláře které se provádějí pouze při určitých událostech ve formuláři, existují metody databáze které jsou přiřazeny k událostem databáze. Jsou to **metody databáze**. Například při otevření databáze můžete potřebovat naplnit určité proměnné, které budete používat v databázi. K tomu můžete použít metodu databáze Při spuštění, která se automaticky provádí při otevření databáze.

Jestli chcete vědět více informací o metodách databáze, přečtěte si část Metody databáze.

Vývoj vaší databáze

Vývoj je proces přizpůsobení databáze pomocí jazyka a ostatních vestavěných nástrojů.

Vytvořením databáze jste provedli první kroky k použití jazyka. Všechny části vaší databáze - tabulky a pole, formuláře a jejich objekty a nabídky - jsou spojeny s jazykem. Jazyk 4th Dimension "ví" o všech těchto částech databáze.

Jazyk patrně začnete používat při vytváření metod pro objekty formuláře pro řízení vstupu dat. Později můžete přidat metodu formuláře k řízení vzhledu vašich formulářů. V dalším vývoji databáze můžete přidat záhlaví nabídek s metodami projektu, které dokončí přizpůsobení databáze vašim potřebám.

Stejně jako ostatní rysy 4th Dimension je i vývoj velmi pružný proces. V průběhu vývoje neplatí žádná formální pravidla - můžete vyvíjet způsobem, který vám vyhovuje. V celém procesu samozřejmě existují některá obecná pravidla.

- Tvorba: V prostředí návrháře vyvíjíte svoji databázi,
- Testování: Návrh zkoušejte v prostředí uživatele,
- Použití: Po úplném přizpůsobení databáze ji používáte v prostředí vlastních nabídek,
- Opravy: Pokud najdete chyby, vrátíte se do prostředí návrháře a opravíte je.

4th Dimension obsahuje zvláštní podpůrné prostředky pro vývoj, které jsou před uživatelem skryty, dokud je nepotřebuje. Při častějším používání jazyka zjistíte, že tyto pomůcky usnadňují proces vývoje. Editor metod například zjistí překlepy a formátuje vaši práci, překladač (jádro, které zpracovává jazyk) zjistí chyby v syntaxi a ukáže, kde jsou; prostředí pro Ladění a odstranění chyb (debugger) vám umožní kontrolovat provádění metod a zjistit chyby v návrhu.

Vytváření aplikace

Doposud jste se seznámili s obecným použitím databáze - vstup dat, hledání, třídění a zprávy. Tyto úlohy jste prováděli v prostředí uživatele pomocí vestavěných nabídek a editorů.

Při používání databáze některé posloupnosti úkonů provádíte opakovaně. Například v databázi osobních kontaktů můžete vyhledávat své obchodní partnery, třídit je podle příjmení a tisknout určitou zprávu pokaždé, když se změní jejich údaje. Tyto úkony se mohou zdát jednoduché, ale když je budete dělat po dvacáté, budou vám určitě zabírat mnoho času. Pokud navíc databázi nebudete několik týdnů používat, zjistíte, že kroky, pomocí kterých jste vytvářeli zprávu si již zcela přesně nepamatujete. Kroky v metodách jsou zřetězeny, takže jediná položka nabídky automaticky provede všechny úkony, které jsou s ní spojeny. Znamená to, že se nemusíte zabývat jednotlivými kroky.

Prostředí vlastních nabídek dovede automatizaci databáze k vrcholu. Aplikace mají vlastní nabídky a provádějí úkony podle potřeb uživatele databáze. Aplikace je tvořena všemi prvky vaší databáze: strukturou, formuláři, metodami, nabídkami a hesly.

Můžete použít 4D Compiler ke zkompilování vaší databáze a vytvoření samostatné aplikace pro Windows nebo Macintosh. Kompilace podstatně urychlí provádění příkazů jazyka, zvýší bezpečnost dat a umožní vám vytvořit

aplikace, které budou úplně nezávislé. Samozřejmě se také prověří správnost syntaxe a typy proměnných v metodách.

Aplikace může být jednoduše pouhá nabídka, která vám umožní vložit jména osob a vytisknout zprávu nebo složitý úplný fakturační skladový a řídící systém. Použití vlastních nabídek nemá žádné omezení. Obvykle aplikace vzniká z databáze používané v prostředí uživatele a rozroste se do databáze úplně řízené vlastními nabídkami.

Kam jít nyní?

• Vývoj databáze může být podle vašich potřeb jednoduchý nebo složitý. Pokud chcete vědět jak rychle vytvořit aplikaci ve 4th Dimension, přečtěte si část Vytváření aplikace 4D.
• Pokud jste nováček ve 4D, přečtěte si část Definice jazyka aby jste se dozvěděli základy o jazyku 4th Dimension. Začněte s částí Úvod do jazyka 4th Dimension.

Příkazy a odkazy pro Úvod

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Vytváření aplikace 4D

Příkazy a odkazy pro Úvod

Verze 6.0

Aplikace je databáze navržená k naplnění určitých potřeb. Obsahuje rozhraní uživatele, které je specifické pro jednoduchost obsluhy. Úkoly, které aplikace plní jsou omezeny podle jejích potřeb. Vytváření aplikací pomocí 4th Dimension je jednodušší a rychlejší než u většiny ostatních programů. 4th Dimension může být použita k vytvoření mnoha typů aplikací obsahujících:

- Systém fakturace,
- Evidence zboží,
- Peněžní systém,
- Účetní systém,
- Personalistika,
- Evidence zákazníků
- Databáze publikované na Internetu a Intranetu.

Je samozřejmě možné, aby jedna aplikace obsahovala všechny tyto systémy. Aplikace jako tyto, jsou klasickým příkladem databází. Nástroje ve 4th Dimension vám umožňují vytvořit aplikace s novějšími rysy:

- Evidence dokumentů,
- Evidence obrázků,
- Aplikace rozesílání katalogů
- Systém elektronické pošty
- Víceuživatelský tabulkový systém
- Seznam jako nabídkový seznam, video kolekce, obrázková kolekce

Aplikace obvykle začínají jako databáze v prostředí uživatele. Databáze se stane aplikací, během přizpůsobování. Co odlišuje aplikaci je to, že systém řízení databáze je skryt před uživatelem. Řízení databáze je automatizováno a uživatel používá nabídky k různým úkonům.

Pokud používáte databázi 4th Dimension v prostředí uživatele, musíte znát kroky k uchování výsledků. V aplikaci používáte prostředí Vlastních nabídek, ve kterém musíte řídit všechny nástroje které jsou v prostředí uživatele automatické.

Jsou to:

- Vybírání tabulky: Dialogová okna **Vybrat tabulku/formulář** a **Seznam tabulek** nejsou pro uživatele dostupné. Musíte použít položky nabídek a metody k přesunu mezi tabulkami.
- Nabídky: V prostředí Vlastních nabídek máte pouze základní nabídku Soubor s položkou Konec, nabídku Upravit a nabídku Nápověda (pouze ve Windows) nebo nabídku Jablko (pouze na Macintoshi). Pokud aplikace vyžaduje více nabídek, musíte je vytvořit a řídit pomocí metod 4th Dimension.
- Editory: Editory jako Dotazů a Třídění nejsou automaticky přístupné v prostředí Vlastních nabídek. Pokud je chcete použít v tomto prostředí, musíte je vyvolat pomocí metod 4th Dimension.

Následující část obsahuje příklady, které ukazují jak může Jazyk automatizovat použití databáze.

Vlastní nabídky: Příklad

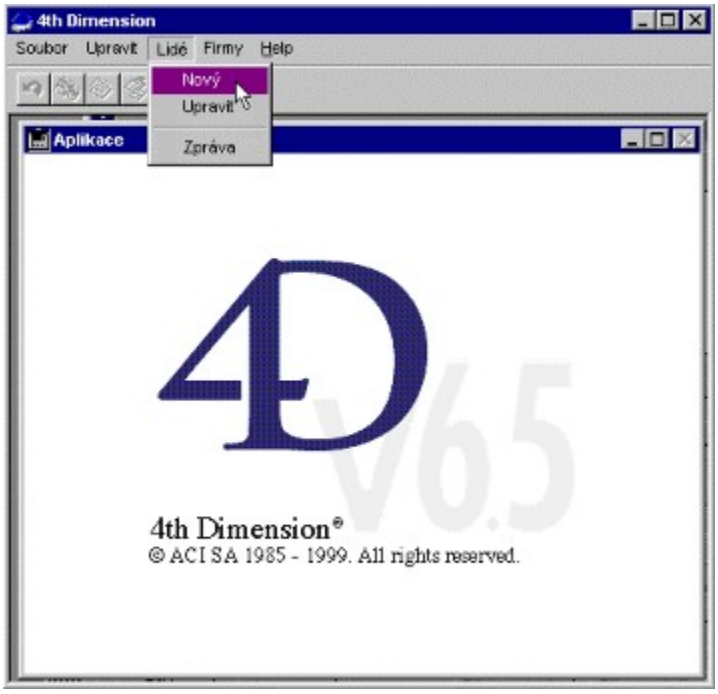
Vlastní nabídky jsou primární rozhraní aplikace. Usnadňují uživateli ovládání a práci s databází. Vytvoření těchto nabídek je velmi jednoduché - přiřadíte metodu k příkazu nabídky (také nazývané položka nabídky) v Editoru nabídek.

Část "Perspektiva uživatele" popisuje, co se stane když uživatel vybere položku nabídky. Další "Co je v pozadí" popisuje práci návrhu, který to způsobí. Ačkoli je příklad jednoduchý, může ukázat, jak vlastní nabídky zjednoduší

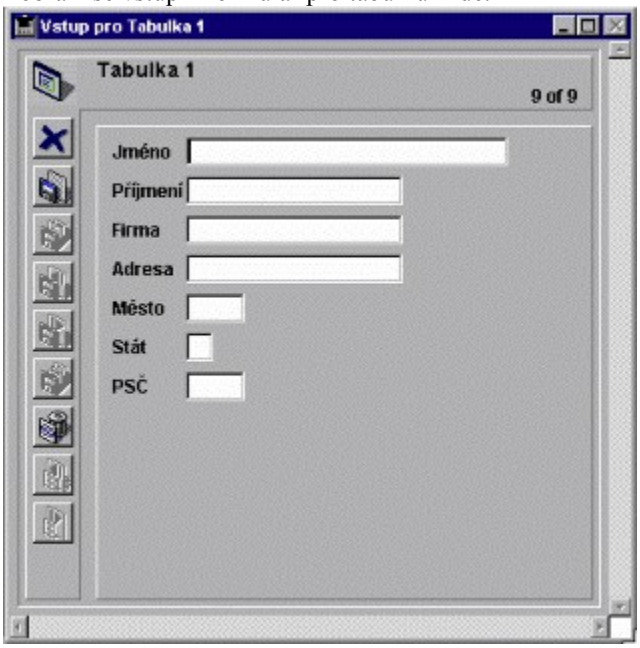
učení a práci s databází. Na rozdíl od nástrojů a nabídek v prostředí uživatele, jsou zde použity pouze nástroje, které budete potřebovat k práci.

Perspektiva uživatele

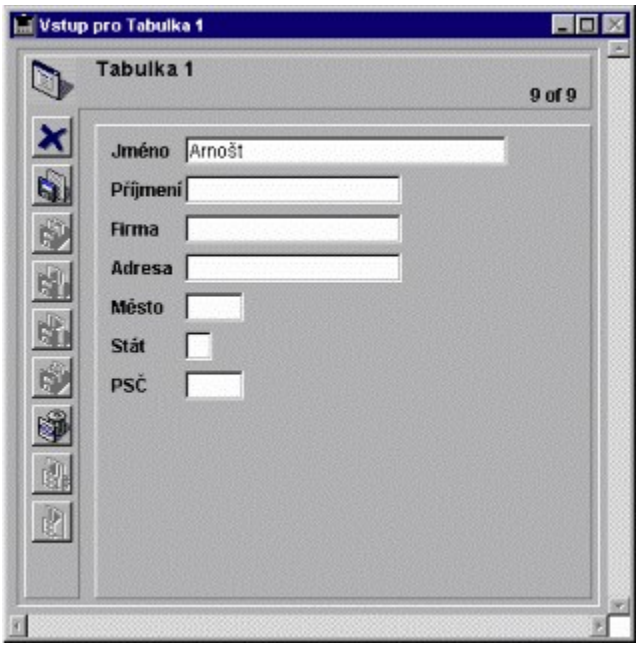
Uživatel použije položku **Nový** z nabídky **Lidé** k předání další osoby do databáze.



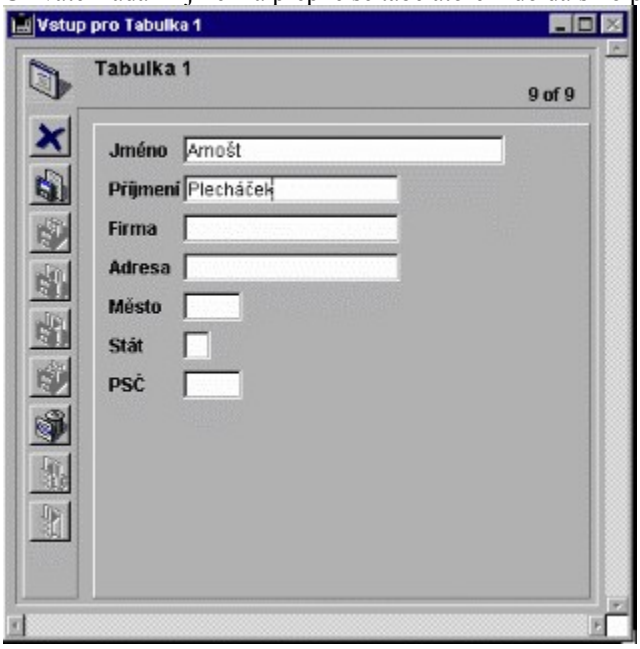
Zobrazí se vstupní formulář pro tabulku Lidé.



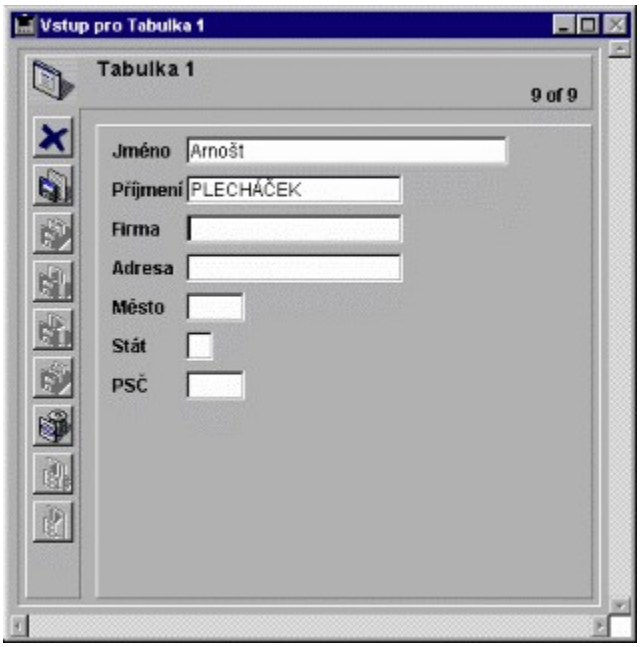
Uživatel napíše Jméno a přepne se tabelátoram do dalšího pole.



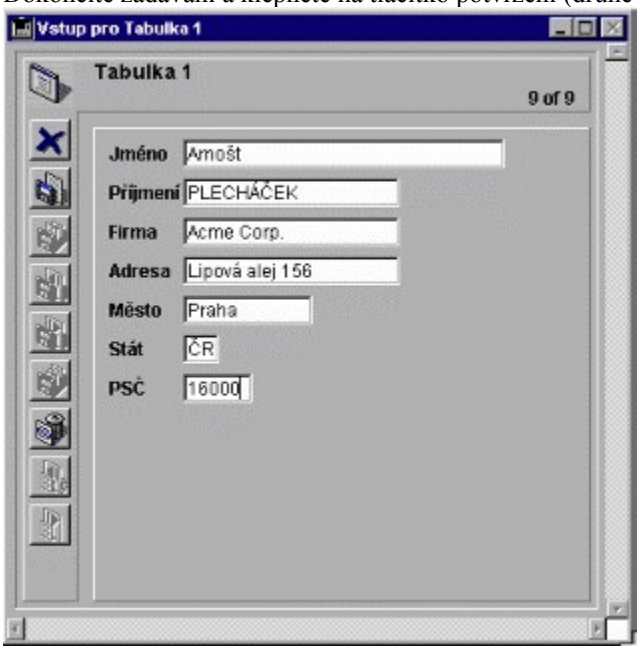
Uživatel zadá Příjmení a přepne se tabelátorem do dalšího pole.



Vidíte, že Příjmení bylo převedeno do velkých písmen.



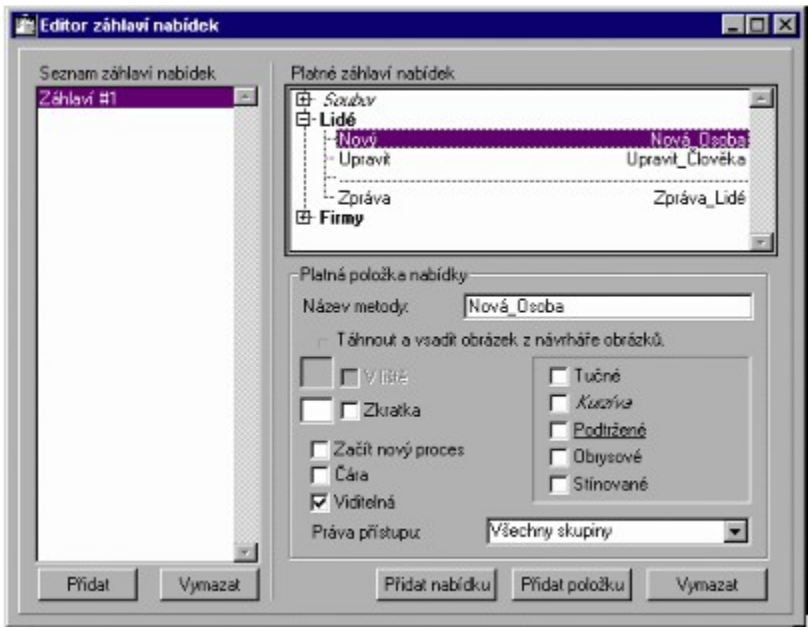
Dokončíte zadávání a klepnete na tlačítko potvrzení (druhé tlačítko ve svislé řadě).



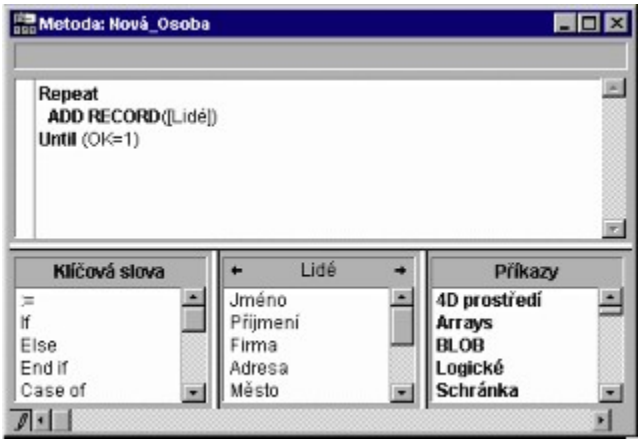
Objeví se další prázdný záznam a uživatel klepne na tlačítko Zrušit (tlačítko s X) k přerušení smyčky pro zadávání záznamů. Uživatel je vrácen do záhlaví nabídek.

Co je v pozadí

Záhlaví nabídek bylo vytvořeno v prostředí návrháře v Editoru nabídek.



Položka nabídky Nový má přiřazenu metodu Nová_Osoba. Tato metoda byla vytvořena v prostředí návrháře v Editoru metod.



Jakmile uživatel vybere tuto položku nabídky, je spuštěna tato metoda:

Repeat

ADD RECORD ([Lidé])

Until (OK=1)

Smyčka Repeat ... Until s příkazem ADD RECORD (přidat záznam) je stejná jako položka nabídky **Nový záznam** v prostředí uživatele. Zobrazí vstupní formulář, a uživatel může zadávat nové záznamy. Jakmile je záznam uložen, zobrazí další prázdný záznam. Tato smyčka bude pokračovat tak dlouho, dokud uživatel neklepne na tlačítko Zrušit.

Jakmile je záznam vkládán, tak probíhá následující:

• K poli Jméno není přiřazena žádná metoda a tak se neděje nic.

• K poli Příjmení je přiřazena metoda. Tato metoda objektu byla vytvořena v prostředí návrháře s použitím Editoru formulářů a Editoru metod.Metoda vypadá takto:

Příjmení:=**Uppercase**(Příjmení)

Tento řádek převede Příjmení na velká písmena.

Jakmile uživatel dokončí zadávání a klepne na tlačítko Zrušit (proměnná OK je nastavena na 0) skončí smyčka s příkazem ADD RECORD.

Pokud nejsou již žádné řádky v metodě, metoda Nová_Osoba skončí a navrátí uživatele do Záhlaví nabídek

Porovnání automatizovaných úloh a akcí prováděných v prostředí uživatele

Pojďme porovnat způsob jakým se úlohy plní v prostředí uživatele a způsob splnění stejných úloh pomocí Jazyka. Budou to následující úlohy:

- Najít skupinu záznamů
- Třídit
- Tisknout zprávu

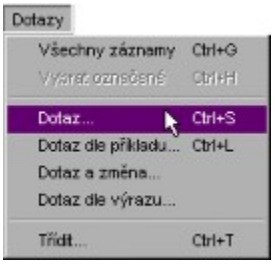
Další část "Použití databáze v prostředí uživatele" zobrazuje úlohy prováděné v prostředí uživatele.

Následující část "Použití vestavěných editorů v prostředí Vlastních nabídek" zobrazuje ty samé úlohy ale ve vlastní aplikaci.

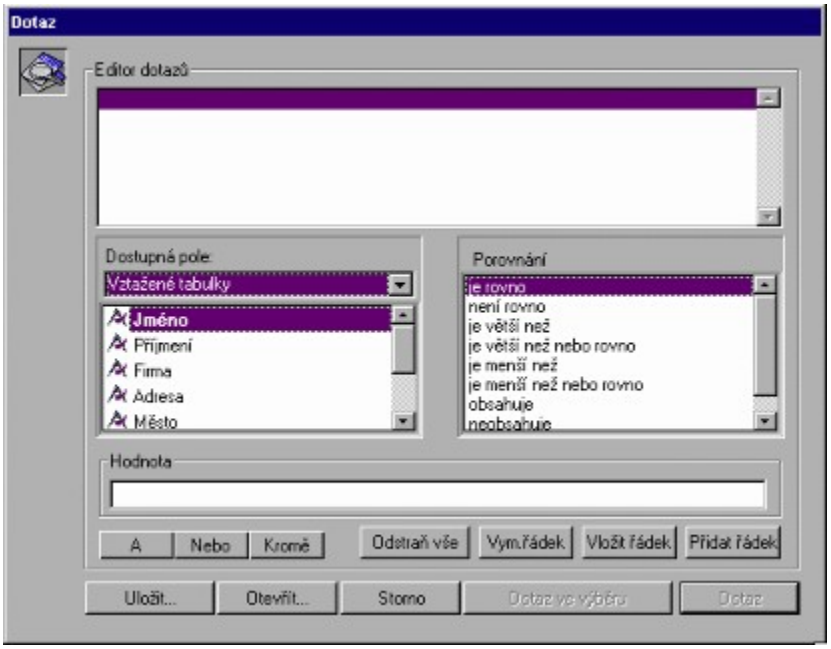
Všiměnte si, že přesto ,že oba způsoby plní jednu akci, v druhé části jsou kroky automatizovány pomocí metody.

Použití databáze v prostředí uživatele

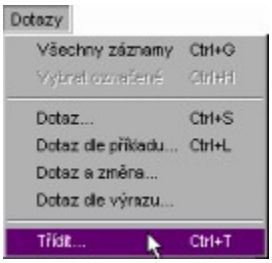
Uživatel vybere položku **Dotaz** z nabídky **Dotazy**.



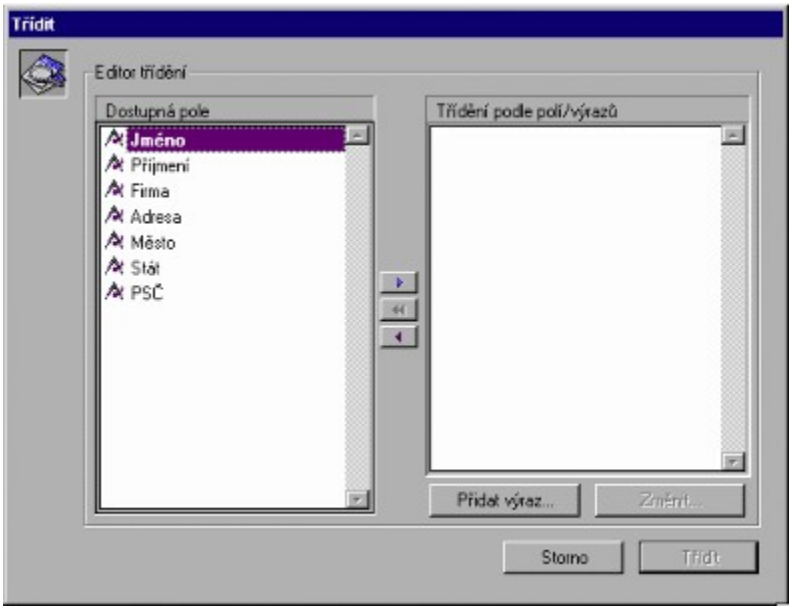
Zobrazí se **Editor dotazů**.



Uživatel zadá kriteria a klepne na tlačítko **Dotaz**. Hledání je dokončeno.
Uživatel vybere položku **Třídit** z nabídky **Dotazy**.



Zobrazí se **Editor třídění**.



Uživatel vloží kriteria a klepne na tlačítko **Třídit**. Třídění je provedeno.
Pokud chcete vytisknout záznam, musíte udělat tyto kroky:
• Vybrat položku **Tisk** z nabídky **Soubor**
• Protože je potřeba vědět, ve kterém formuláři se bude tisknout, musíte **Vybrat tiskový formulář** v okně, které se zobrazí.
• Zobrazí se okno tiskárny, kde nastavíte vaše požadavky a zpráva je vytištěna.

Použití vestavěných editorů v prostředí Vlastních nabídek

Pojďme se podívat, jak probíhají tyto operace v prostředí Vlastních nabídek.

Uživatel vybere položku Zpráva z nabídky Lidé.

V této části je to jednodušší pro uživatele - nepotřebují vědět, že vyhledání je první krok.

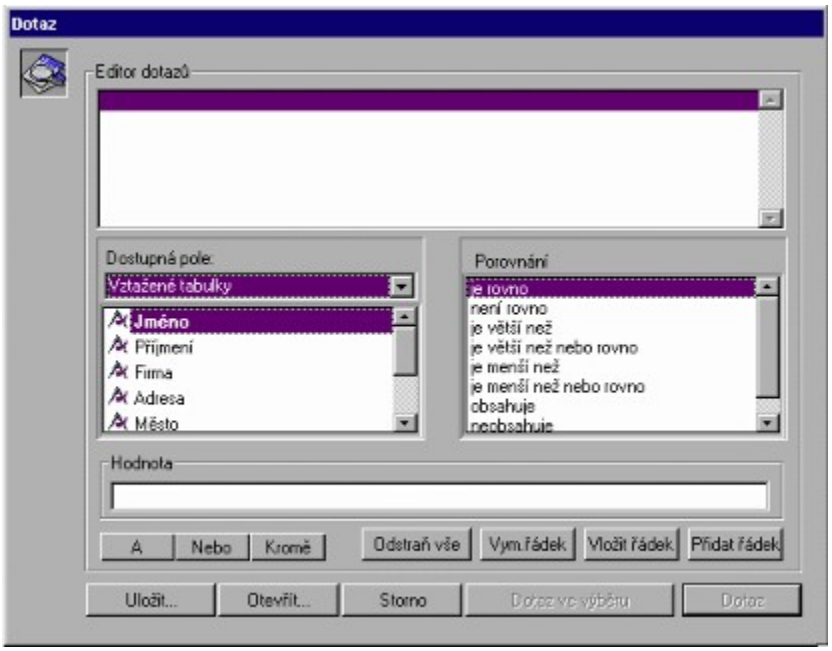
Metoda s názvem Moje_Zpráva přiřazená k položce nabídky vypadá takto:

```
QUERY ([Lidé])
ORDER BY ([Lidé])
OUTPUT FORM ([Lidé];"Zpráva")
PRINT SELECTION ([Lidé])
```

První řádek je proveden:

```
QUERY ([Lidé])
```

Zobrazí se Editor dotazů.

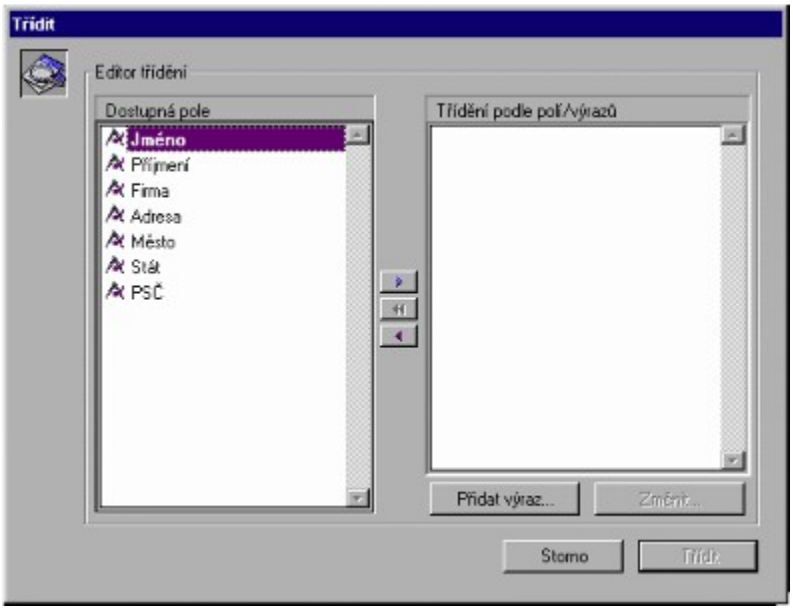


Uživatel zadá kriteria a klepne na tlačítko **Dotaz.** Dotaz je proveden.
Provede se druhá řádka metody Moje_Zpráva:

ORDER BY ([Lidé])

Všimněte si, že uživatel nemusí vědět, že třídění je druhý krok.

Zobrazí se okno Editoru třídění.



Uživatel zadá kriteria a klepne na tlačítko **Třídit**. Třídění se provede.
Třetí řádka metody Moje_Zpráva se provede:

OUTPUT FORM ([Lidé];"Zpráva")

Opět uživatel nemusí vědět, že má nyní nastavit výstupní formulář, protože to udělá metoda sama.

Spustí se poslední řádek metody:

PRINT SELECTION ([Lidé])

Objeví se dialogová okna tisku. Uživatel vybere volby a vytiskne zprávu.

Další automatizace aplikace

Stejné příkazy jako v předchozím příkladu mohou být použity pro další automatizaci aplikace.

Podívejme se na novou verzi metody Moje_Zpráva.

Uživatel vybere položku Zprávy z nabídky Lidé. Metoda Moje_Zpráva, která je přiřazena k položce vypadá takto:

```
QUERY([Lidé];[Lidé]Firma="Acme")
ORDER BY([Lidé]; [Lidé]Příjmení;>;[Lidé]Jméno;>)
OUTPUT FORM([Lidé];"Zpráva")
PRINT SELECTION([Lidé];*)
```

Spustí se první řádek:

```
QUERY([Lidé];[Lidé]Firma="Acme")
```

Editor dotazů se nebude zobrazovat, protože dotaz je již zadán pomocí příkazu QUERY (dotaz). Uživatel již nemusí nic dělat.

Spustí se druhý řádek metody:

```
ORDER BY([Lidé]; [Lidé]Příjmení;>;[Lidé]Jméno;>)
```

Opět se nebude zobrazovat okno Editoru třídění a provede se třídění zadané v metodě.

Spustí se poslední řádky metody:

```
OUTPUT FORM([Lidé];"Zpráva")
PRINT SELECTION([Lidé];*)
```

Nezobrazí se okna tisku. Příkaz PRINT SELECTION (tisknout výběr) přijme volbu hvězdičky (*) a ta znamená, že příkaz vezme nastavení tiskárny, které bylo při vytváření zprávy. Zpráva je vytištěna.

Tato dodatečná automatizace zabrání uživateli zadávat kritéria do třech dialogových oken. Zde jsou výhody:

- Dotaz je automaticky proveden: uživatel může zadat špatné údaje při hledání,
- Třídění je provedeno automaticky: uživatel může zadat špatné údaje při definici třídění,
- Tisk je proveden automaticky: uživatel může zadat špatné údaje při tisku.

Nástroje pro vývoj aplikací 4D

Při vývoji aplikace 4D objevíte spoustu vlastností, které jste neznali když jste začínali. Standardní verzi 4D můžete rozšířit přidáním dalších nástrojů a zásuvných modulů (plug-in) do vašeho vývojářského prostředí 4th Dimension.

Vývojářské nástroje

Firma ACI vytvořila mnoho nástrojů, které mohou být použity při vývoji aplikací. Tyto nástroje vám pomohou přenášet objekty mezi databázemi, kompilace databáze, zkoušení databáze syntakticky. Jsou to tyto nástroje:

- **4D Insider**: Můžete jej použít k zobrazení a tisku metod, proměnných, příkazů, externích metod, struktury, seznamů a formulářů. Nástroje 4D Insideru vám umožní zjistit kde je každý z těchto objektů použit v databázi. Umožňuje vám přesunovat objekty jako jsou tabulky, formuláře, metody, záhlaví nabídek, seznamy a styly z jedné

databáze do druhé.

• **4D Compiler:** přeloží vaše metody a skripty do strojového kódu. Urychlí to práci databáze, zkontroluje metody a objeví logické a syntaktické konflikty. Zabrání také prohlížení a upravování databáze ať už úmyslně nebo omylem.

Plug-iny 4D

Můžete zvýšit výkon vašich aplikací tím, že do vývojového prostředí 4th Dimension přidáte profesionální zásuvné moduly (Plug-in).

ACI distribuuje následující moduly produktivity:

• **4D Write**: Textový procesor
• **4D Calc**: Tabulkový procesor
• **4D Draw**: objektový grafický modul

ACI také distribuuje následující moduly propojení:

• **4D ODBC**: Spojení přes ODBC
• **4D ORACLE**: Spojení s databázemi ORACLE
• **4D SQL SERVER**: Spojení se SYBASE SQL Server a Microsoft SQL Server
• **4D Open**: Spojení (z 4D do 4D) pro vytvoření informačních systémů 4D

Jestli chcete vědět více informací, kontaktujete firmu ACI nebo jejich partnery. Navštivte naše Web stránky:

USA a International	http://www.acius.com
Francie a International	http://www.aci.fr
Japonsko a Asie	http://www.aci.co.jp
Česká republika	http://www.inforce.cz

Komunita 4D a third party nástroje

Ve světě existuje velmi silná komunita 4D složená ze skupin uživatelů, elektronických fór a partnerů ACI. Partneři ACI vytvářejí **third party** nástroje jako Area List Pro od Foresight Technology, Inc. (http://www.fsti.com).

Pokud si prohlédnete CD s 4th Dimension najdete tam informace a dema od partnerů ACI.

Komunita 4th Dimension má přístup k tipům a trikům, výuce, informacím a dodatečným nástrojům které ušetří váš čas a zvýší vaši produktivitu.

Příkazy a odkazy pro Úvod

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Úvod do jazyka 4D

Příkazy a odkazy pro Definice jazyka

Verze 6.0

Jazyk 4th Dimension je vytvořen z mnoha rozdílných komponentů které vám pomohou plnit různé úlohy a řídit práci s vašimi daty.

• **Typy dat:** Zařazení dat v databázi. Přečtěte si tuto část stejně jako podrobný popis v části Typy dat..
• **Proměnné**: Volitelné uložení dat do paměti. Přečtěte si část Proměnné..
• **Operátory**: Znaky které provádějí výpočty dvou hodnot. . Přečtěte si tuto část stejně jako podrobný popis v části Operátory a jejich podčásti.
• **Výrazy**: Kombinace různých komponentů které vytvářejí hodnoty. Přečtěte si popis v této části.
• **Příkazy:** Vestavěné instrukce k plnění různých akcí. Všechny příkazy 4th Dimension jsou popsány v tomto manuálu po tématických částech. Pokud je to nutné, téma je probráno v úvodu. Pokud chcete, můžete vložit nové příkazy do vašeho vývojového prostředí pomocí plug-inů 4th Dimension. Pokud například přidáte k databázi 4D Write, přidají se příkazy 4D Write k vytváření dokumentů.
• **Metody**: Instrukce které napíšete mohou obsahovat všechny zde popsané části. Přečtěte si popis v části Metody a jejích pododílech.

Tato část vám poskytne úvod k Typům dat, Operátorům a Výrazům. Pro další části jazyka si přečtěte části zmíněné níže.

Navíc:

• Jednotlivé komponenty jazyka jako jsou proměnné, mají své názvy, kterým se říká identifikátory. Jestli chcete vědět více informací o identikátorech a pravidlech pro pojmenování objektů, přečtěte si část Identifikátory..
• Jestli chcete vědět více informací o proměnných array, přečtěte se část Array.
• Jestli chcete vědět více informací o BLOB proměnných, přečtěte si část Příkazy BLOB._.
• Pokud budete vaši databázi kompilovat, přečtěte si část Příkazy kompilátoru a *Příručku ke 4D Compiler.*

Příkazy a odkazy pro Definice jazyka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Typy dat

Příkazy a odkazy pro Definice jazyka

Verze 6.0

Pole, proměnné a výrazy mohou mít následující typ dat:

Typ dat	***Pole***	***Proměnná / Výraz***	
Řetězec (string)	*Ano*	*Ano*	*Ano*
Číselné	*Ano*	*Ano*	*Ano*
Datum	*Ano*	*Ano*	*Ano*
Čas	*Ano*	*Ano*	*Ano*
Logické	*Ano*	*Ano*	*Ano*
Obrázkové	*Ano*	*Ano*	*Ano*
Ukazatel	*Ne*	*Ano*	*Ano*
BLOB	*Ano*	*Ano*	*Ne*
Aray	*Ne*	*Ano*	*Ne*
Podtabulka	*Ano*	*Ne*	*Ne*
Nedefinované	*Ne*	*Ano*	*Ano*

Poznámky

1. Řetězce mohou obsahovat alfanumerická pole, proměnné s pevnou délkou a textová pole nebo proměnné,

2. Čísla obsahují pole nebo proměnné Real, Integer a Long Integer,

3. BLOB je zkratka pro Binary Large Object. Jestli chcete vědět více informací o BLOBech, přečtěte Příkazy BLOB.

4. Array obsahují všechny typy array. Jestli chcete vědět více informací, přečtěte si část Array.

Řetězec (String)

Řetězec je generický člen, kterým může být:

- Alfanumerické pole
- Proměnná s pevnou délkou
- Textové pole nebo textová proměná
- Jakýkoli řetězec nebo textový výraz

Řetězec je série znaků. Každý znak může být jeden z 256 ASCII znaků. Jestli chcete vědět více informací o ASCII znacích a jejich použití mezi platformami ve 4D, přečtěte si část ASCII kódy.

- Alfanumerické pole může obsahovat od 2 do 80 znaků (tento limit je dán vlastnostmi (nastavením) pole)
- Proměnná s pevnou délkou může obsahovat od 0 do 255 znaků (tato délka je určena definicí proměnné)
- Textové pole nebo proměnná může obsahovat od 0 do 32.000 znaků.

Řetězec nemůžete přiřadit k textovému poli: 4D převede hodnotu. Ve výrazu můžete slučovat text a řetězce.

Poznámka

V této příručce jsou, jak řetězce tak i textové parametry v příkazech, popisovány jako řetězce, pokud není zmíněno jinak.

Číselné

Číslo je generický člen, který může být:

- Real pole, proměnná nebo výraz
- Integer pole, proměnná nebo výraz
- Long Integer pole, proměnná nebo výraz

Rozsah typu Real je ±1.7e±308 (15 znaků)

Rozsah typu Integer je -32.768 ... 32.766

Rozsah typu Long Integer je -2^31..(2^31)-1

Jakýkoli typ číselných dat můžete přiřadit k jinému. Pokud to bude potřeba, provede 4D převod nebo zaokrouhlení. Pokud je hodnota jiná než povolený rozsah, převod nevrátí správnou hodnotu. Typy dat můžete spojovat ve výrazu.

Poznámka: V této příručce jsou všechny typy číselných dat, Real, Integer a Long Integer, zmiňovány jako čísla, pokud není popsáno jinak.

Datum

- Datumové pole, proměnná nebo výraz mohou být v rozmezí 1.1.100 do 31.12.32767.
- V anglické verzi jsou datumy v pořadí Měsíc/den/rok, v české verzi podle nastavení systému.
- Pokud století zadáte jenom dvěma znaky, bude pochopeno jako 19xx. (pokud není upraveno století příkazem SET DEFAULT CENTURY (nastavit výchozí století))

Poznámka: V této příručce jsou datumové parametry příkazu zmiňovány jako Datum, pokud není popsáno jinak.

Čas

- Časové pole, proměnné nebo výraz mohou být v rozmezí od 00:00:00 do 596000:00:00.
- V anglické i české verzi je čas v pořadí hodina:datum:sekunda
- Čas je ve 24 hodinovém formátu
- Časová hodnota může být konvertována do čísla. Číslo, které je vraceno z času je počet vteřin, který čas udává. Jestli chcete vědět více informací, přečtěte si část Časové operátory.

Poznámka: V této příručce jsou časové parametry příkazů popisovány jako Čas, pokud není popsáno jinak.

Logické

Logické pole, proměnná nebo výraz mohou mít pouze hodnotu TRUE (pravda) nebo FALSE (nepravda).

Poznámka: V této příručce jsou logické parametry popisovány jako Logické, pokud není popsáno jinak.

Obrázková

Obrázková pole, proměnná nebo výraz mohou mít jakýkoli obrázkový formát Windows nebo Macintosh. To znamená obrázky, které můžete načíst do Schránky můžete přečíst 4D nebo dodatečnými Plug-in.

Poznámka: V této příručce jsou obrázkové parametry příkazů popisovány jako Obrázky, pokud není popsáno jinak.

Ukazatel

Proměnná ukazatele nebo výraz typu ukazatel je odkaz na jinou proměnnou (obsahující i array či prvky array), tabulce, nebo poli. Pole struktury typu ukazatel neexistuje.

Jestli chcete vědět více informací o ukazatelích přečtěte si část Ukazatele,.

Poznámka: V této příručce jsou parametry ukazatele v popisu příkazu popisovány jako Ukazatel, pokud není popsáno jinak.

BLOB

BLOB pole nebo proměnná je série bytů (od 0 do 2 GB) které můžete vkládat s použitím příkazů BLOB. Výraz typu BLOB neexistuje.

Poznámka: V této příručce jsou BLOB parametry v popisech příkazů zmiňovány jako BLOB.

Array

Array není samostatný typ dat. Různé existující typy array (jako Long Integer Array, Textový Array, atd.) jsou tímto názvem odkazovány. Array jsou proměnné řazené za sebou - neexistuje pole typu array ani výraz typu array. Jestli chcete vědět více informací, přečtěte si část Array.

Poznámka: V této příručce jsou parametry array v příkazech popisovány jako Array pokud není popsáno jinak.

Podtabulka

Podtabulka není ve skutečnosti typ dat. Pouze pole mohou být typu podatbulka. Jestli chcete vědět více informací o podtabulkách, přečtěte si *Příručku návrháře 4th Dimension*.

Nedefinováno

Nedefinováno není typ dat. Značí pouze proměnné, které ještě nebyly definovány. Funkce (metoda projektu, která vrací výsledek) může vracet nedefinovanou hodnotu jestliže je v metodě výsledek funkce ($0) přiřazen nedefinovanému výrazu (výraz je počítán z nejméně jedné nedefinované proměnné). Pole nemůže být nedefinované.

Převádění typů dat

Jazyk 4D obsahuje operátory a příkazy k převodu mezi typy dat, kde je převod možný. Jazyk 4D provede kontrolu typu dat. Například nemůžete napsat: "abc" +0.5+!25.12.98!-?700:30:45? Tento zápis způsobí chybu syntaxe.

Následující tabulka ukazuje základní typy dat, typy na které může být daný typ převeden a příkaz, kterým to je možné:

Typ dat	***Převést na Řetězec***	***Převést na Číslo***	***Převést na Datum***	***Převést na Čas***
Řetězec		**Num**	**Date**	**Time**

Číslo (*)	**String**	
Datum	**String**	
Čas	**String**	
Logické		**Num**

(*) Časové hodnoty mohou být zobrazeny jako čísla.

Poznámka Mimo převodů v této tabulce mohou být provedena složitější data s použitím různých operátorů a příkazů.

Dále si přečtěte

Array, Konstanty, Řízení průběhu, Identifikátory, Metody, Operátory, Ukazatele, Typy dat, Proměnné.

Příkazy a odkazy pro Definice jazyka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Konstanty

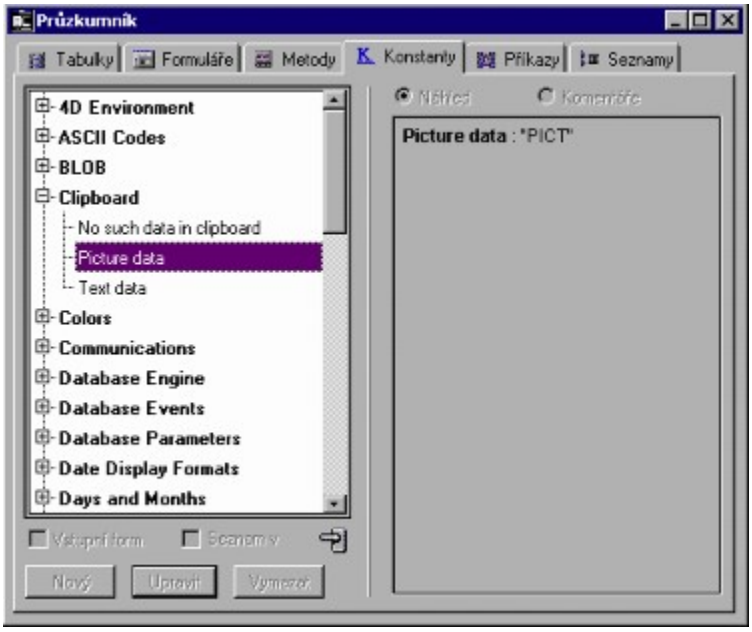
Příkazy a odkazy pro Definice jazyka

Verze 6.0

Konstanta je výraz, který má pevně danou hodnotu. Existují dva typy konstant: **Předdefinované konstanty** které vybíráte podle názvu a **přesné konstanty** pro které píšete aktuální hodnotu.

Předdefinované konstanty

Ve 4th Dimension V6 jsou předdefinované konstanty, které jsou zobrazeny na straně Konstanty v Průzkumníku.



Předdefinované konstanty jsou zobrazeny podle tématu. K použití předdefinovaných konstant v Editoru Metod:

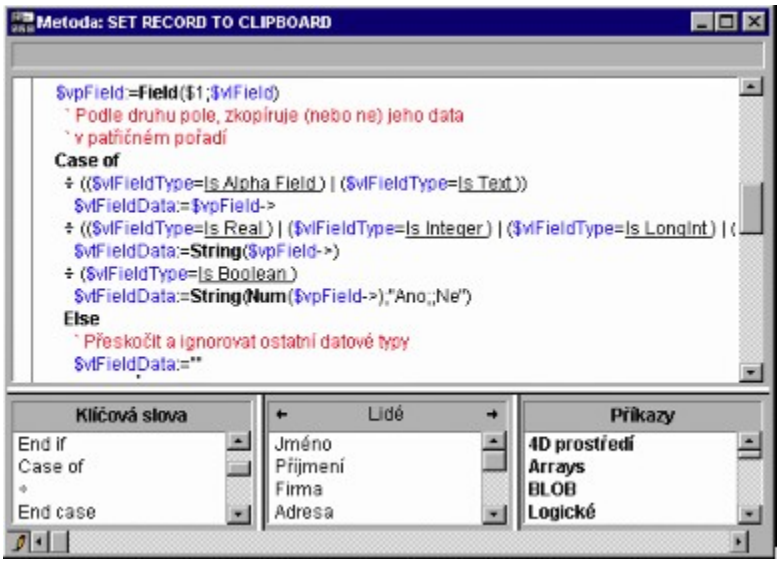
- Přetáhněte konstantu z okna Průzkumníka do okna Editoru metod.
- Napište název konstanty v okně Editoru metod

Název předdefinované konstanty může mít až 31 znaků.

Tip: Pokud název konstanty zadáváte ručně, můžete použít znak @ k doplnění názvu, pokud jej neznáte celý. Například pokud napíšete "No such da@", 4th Dimension doplní tento řádek na "No such data in clipboard" ve chvíli kdy stisknete Return nebo Enter.

Poznámka: Předdefinované konstanty (kolem 500) jsou zobrazeny podle tématu v tomto manuálu. Přečtěte si část O tomto manuálu pro podrobnější informace. Pokud je to potřeba, je zobrazen seznam konstant v popisu příkazu.

Předdefinované konstanty se v Editoru metod a v Debuggeru (Ladění) objevují podtržené.



V tomto okně je použita předdefinovaná konstanta is Alpha Field.

Přesné konstanty

Tyto konstanty mohou mít jeden ze čtyř typů dat:

- Řetězec
- Číselné
- Datumové
- Časové

Řetězcové konstanty

Řetězové konstanty jsou uzavřeny ve dvou uvozovkách ("..."). Následují nějaké příklady:

"Přidat záznamy"

"Nenalezeny žádné záznamy"

"Faktura"

Prázdný řetězec je zadán pomocí uvozovek, ale nic v nich není ("").

Číselné konstanty

Číslené konstanty jsou psané jako reálné číslo. Následují nějaké příklady:

27

123.76

0.0076

Záporné konstanty jsou značeny znaménkem mínus (-). Například:

-27

-123.76

-0.0076

Datumové konstanty

Datumové konstanty jsou uzavírány do vykřičníků (!...!). V České verzi 4D jsou datumy zadávány v pořadí Den.Měsíc.rok. Následují nějaké příklady:

!1.1.76!

!4.4.04!

!12.6.98!

Nulové datum je zadáno !00.00.00!

Tip: Pro zadání nulového datumu, můžete v Editoru Metod použít zkratku. Stačí napsat vykřičník (!) a editor sám doplní nulové datum.

Poznámka: Pokud neupravíte století pomocí příkazu SET DEFAULT CENTURY, bude dvoumístný rok chápán jako v 1900.

Časové konstanty

Časové konstanty se zadávají do dvou otazníků (?...?).

Poznámka: Tato syntaxe je použitelná jak na Windows tak na Macintoshi. Na Macintoshi můžete použít ještě znak kříže (†) Option T na US klávesnici.

Na české klávesnici je čas zadáván v pořadí hodina:minuta:sekunda s dvojtečkou (:) jako oddělovačem mezi částmi. Čas je zadáván ve 24 hodinovém formátu.

Následují nějaké příklady časových konstant:

?00:00:00? - půlnoc

?09:30:00? - 9:30 ráno

?13:01:59? - 13 hodin, 1 minuta a 59 sekund

Nulový čas je ?00:00:00?.

Tip: Pro zadání nulového času můžete v Editoru metod použít znak (?) a editor sám doplní nulový čas.

Dále si přečtěte

Řízení průběhu, Typy dat, Identifikátory, Metody, Operátory, Ukazatele, Proměnné.

Příkazy a odkazy pro Definice jazyka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Proměnné

Příkazy a odkazy pro Definice jazyka

Verze 6.0

Ve 4th Dimension jsou možné dva typy ukládání dat. Pole ukládají data trvale na disk a proměnné je ukládají dočasně do paměti.

Když vytváříte databázi, musíte definovat názvy a typy polí, které chcete používat. Proměnné jsou v tomto stejné - také musíte nastavit názvy a typy.

Následující typy proměnných komunikují s každým typem dat:

- Řetězec: Pevný alfanumerický řetězec s délkou maximálně 255 znaků
- Text: Alfanumerický řetězec s maximální délkou 32000 znaků.
- Integer: Číslo od -32768 do 32767
- Long Integer: Číslo od -2^31 do (2^31)-1
- Real: Číslo od ±1.7e±308 (15 znaků)
- Datum: 1.1.100 do 31.12.3767
- Čas: 00:00:00 do 696000:00:00 (sekund od půlnoci)
- Logické: Pravda a Nepravda
- Obrázkové: Jakýkoli obrázek Windows nebo Macintosh
- BLOB (Binary Large Object): Řada bitů do 2GB
- Ukazatel: Ukazatel na tabulce, poli, proměnné, array nebo položce array

Proměnné (až na ukazatele a BLOB) můžete zobrazit na obrazovce, vložit do nich data a vytisknout je ve zprávě. V tomto případě fungují dostupné a nedostupné proměnné pouze jako pole a můžete na ně použít stejné vestavěné prostředky:

- Formát zobrazení
- Kontrola dat jako vstupní filtr a výchozí hodnota
- Znaky filtru
- Výběrový seznam (hierarchický seznam)
- Dostupné nebo nedostupné hodnoty

Proměnné mohou dělat také následující:

- Řídit tlačítka (tlačítka, přepínače, zaškrtávací políčka, 3D tlačítka, atd.)
- Řídit ukazatele (pravítka, výseče, měřítka)
- Řídit posuvné oblasti, rozevírací nabídky a rozevírací seznamy
- Řídit hierarchické seznamy a nabídky
- Řídit tlačítka na síti, ovládací karty, obrázková tlačítka, atd.
- Zobrazit výsledky výpočtů, které se nemusí ukládat.

Vytváření proměnných

Proměnnou vytvoříte tím, že jí použijete: nemusíte ji nijak definovat, jako u polí. Pokud například potřebujete proměnnou, která bude obsahovat platné datum plus 30 dní, napíšete:

```
MojeDatum:=Current date+30
```

4th Dimension vytvoří MojeDatum a bude zde držet hodnotu, kterou potřebujete. Význam této řádky je "do proměnné MojeDatum přiřadit platné datum plus 30 dní (není-li proměnná definována, definovat)". Proměnnou

MojeDatum můžete nyní použít kdekoli v databázi. Můžete například přiřadit tuto proměnou k poli stejného typu:

[MojeTabulka]MojePole:=MojeDatum

Někdy můžete potřebovat proměnnou, která bude mít přesně určený typ. Jestli chcete vědět více informací o psaní proměnných pro kompilaci databáze, přečtěte si část Příkazy kompilátoru.

Přiřazení dat do proměnných

Data mohou být předána do proměnných nebo z nich zkopírována. Předání dat do proměnné se nazývá **přiřazení dat do proměnné** a provádí se operátorem přiřazení (:=). Tento operátor se používá i k přiřazení hodnot do polí nebo proměnných.

Operátor přiřazení je základní způsob vytvoření proměnné a předání dat. Název proměnné, kterou chcete vytvořit napíšete nalevo operátoru. Například řádek:

MojeČíslo:=3

vytvoří proměnnou MojeČíslo a přiřadí k ní hodnotu 3. Pokud by tato proměnná již existovala, pouze se do ní přiřadí nová hodnota 3.

Proměnné by samozřejmě nebyly moc užitečné, pokud by jste z nich nemohli data použít. Ještě jednou použijete operátor přiřazení. Pokud potřebujete přiřadit proměnnou MojeČíslo (hodnotu v ní) do pole [Produkty]Velikost, napíšete MojeČíslo na pravou stránku operátoru:

[Produkty]Velikost:=MojeČíslo

V tomto případě bude hodnota pole 3. Tento příklad je jednoduchý, ale dobře ukazuje,jak se dají data přesunovat pomocí Jazyka.

Důležité: Dejte si pozor, aby se vám nepletl operátor přiřazení (:=) a porovnání (=). Každý z těchto znaků, má úplně jinou funkci. Jestli chcete vědět více informací, přečtěte si část Operátory.

Místní, procesní a meziprocesové proměnné

Můžete vytvářet tři typy proměnných: Místní proměnné, proměnné procesu a meziprocesové proměnné. Rozdíl mezi těmito třemi typy proměnných je jejich rozsah platnosti a rozdíl je v dostupnosti těchto typů proměnných pro jednotlivé objekty.

Místní proměnné

Místní proměnná, jak její název napovídá, je místní pro metodu ve které byla vytvořena a nelze ji použít mimo tuto metodu. Být místní v metodě znamená že je "místní v rozsahu". Místní proměnné se používají k omezení života proměnné, která je potřebná pouze pro danou metodu.

Místní proměnné můžete použít z důvodů:

- Vyhnout se konfliktům s názvy jiných proměnných, jiných metod
- Dočasně použít data
- Omezit počet proměnných procesu

Název místní proměnné vždy musí začínat znakem dolar ($) a může být až 31 znaků dlouhý. Pokud použijete delší název, 4th Dimension jej ořízne na potřebnou velikost.

Pokud pracujete v databázi, která má mnoho metod a proměnných zjistíte, že potřebujete používat některé proměnné

pouze v metodách na kterých pracujete. Pak můžete vytvořit místní proměnnou v metodě bez toho, že jste se museli bát, že její název je již použit v jiné metodě.

Velmi často v databázi potřebujete drobné informace od uživatelů. Na to je určen příkaz Reqeust (žádost). Zobrazí dialogové okno s vámi zadaným textem na žádost. Jakmile uživatel zadá hodnotu, příkaz vrátí tu informaci, která byla zadána. Obvykle nepotřebujete tuto informaci držet příliš dlouho. Následuje klasický příklad použití tohoto příkazu:

```
$vsID:=Request("Zadejte prosím vaše ID:")
If (OK=1)
    QUERY ([Lidé];[Lidé]ID =$vsID)
End if
```

Tato metoda vyžaduje na uživateli zadání ID. Zadaná hodnota se vloží do místní proměnné $vsID a následuje vyhledání tohoto ID v tabulce [Lidé]. Jakmile je příkaz dokončen, vymaže se tato proměnná z paměti. Je to velmi užitečné, protože již není potřeba a zbytečně by zabírala paměť.

Proměnné procesu

Proměnná procesu (procesová) je přístupná pouze pro proces. Je přístupná pro metodu procesu a všechny další metody, které jsou volány z tohoto procesu.

Proměnná procesu nemusí začínat nějakým určitým znakem a může být dlouhá až 31 znaků.

V nezkompilované verzi jsou proměnné zpracovávány dynamicky, t.j. jsou vytvářeny a mazány z paměti během chodu. Ve zkompilované verzi nesou procesy každý stejnou definici procesových proměnných, ale každý z procesů má svůj vlastní výskyt proměnných. Například proměná pProm může mít jednu hodnotu v procesu P_1 a jinou v procesu P_2.

Nově ve verzi 6 může proces použít proměnou z jiného procesu pomocí příkazu GET PROCESS VARIABLE a SET PROCESS VARIABLE. Je dobrou praxí omezit použití těchto příkazů na situace, pro které byly ve 4D zamýšleny.

- Meziprocesová komunikace na určitém místě vaší metody
- Použití vlastností uchopit a vsadit mezi procesy
- V architektuře klient/server komunikace mezi procesyklienta a procesyna serveru

Jestli chcete vědět více informací, přečtěte si část Procesy a popis těchto příkazů.

Meziprocesové proměnné

Meziprocesové proměné jsou dostupné celé databázi a jsou sdíleny každým procesem. Nejčastěji se používají pro přenos dat mezi procesy.

Názvy meziprocesových proměnných vždy začínají znaky (<>) - menší než a větší než - následované až 31 znaků dlouhým názvem.

Poznámka : Tento styl zápisu může být použit jak na Macintoshi tak i na Windows. Pouze na Macintoshi můžete použít ještě znak diamant (Option-Shift-V na US klávesnici).

V architektuře klient/server je na každém počítači jeden výskyt meziprocesových proměnných, ale může mít jinou hodnotu na každém ze strojů.

Proměnné objektů formuláře

V Editoru formulářů při vytvoření aktivního objektu - tlačítko, přepínač, zaškrtávací políčko, posuvná oblast, pravítko, atd. - se automaticky vytvoří proměnná, která má stejný název jako objekt. Pokud například vytvoříte

tlačítko pojmenované MojeTlačítko, vytvoří se proměná MojeTlačítko. Povšiměte si, že tento název není text tlačítka, ale pouze jeho název.

Proměnné objektů formuláře vám umožňují řídit a hlídat objekty. Například pokud na tlačítko klepnete, je jeho proměnná nastavena na 1; jinak je stále nastavena na 0. Proměnné přiřazené k měřítkům nebo výsečím vám umožňují číst a měnit aktuální hodnotu. Pokud například potáhnete měřítko na jiné nastavení, hodnota proměnné se změní na platné nové nastavení. Stejně tak pokud změníte hodnotu proměné pomocí metody, překreslí se i měřítko.

Jestli chcete vědět více informací o proměných a formulářích, přečtěte si *Příručku návrháře 4th Dimension.*

Systémové proměnné

4th Dimension pracuje s mnoha proměnými, které se nazývají Systémové proměné. Tyto proměné vám umožňují konrolovat mnoho operací. Systémové proměné jsou proměné procesu, které jsou přístupné pouze z procesu.

Nejdůležitější systémovou proměnnou je proměnná **OK**. Jak její název napovídá, říká vám, že je všechno OK v tomto procesu. Byl záznam uložen? Byla dokončena operace importu? Klepnul uživatel na tlačítko OK? Proměnná OK je nastavena na 1 pokud je operace úspěšně dokončena, pokud ne, je nastavena na 0.

Jestli chcete vědět více informací o systémových proměnných, přečtěte si část Systémové proměnné.

Dále si přečtěte

Array, Konstanty, Řízení průběhu, Typy dat, Identifikátory, Metody, Operátory, Ukazatele.

Příkazy a odkazy pro Definice jazyka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Systémové proměnné

Příkazy a odkazy pro Definice jazyka

Verze 6.0 (Upraveno)

4th Dimension ovládá **systémové proměnné** pomocí kterých můžete řídit provádění různých operací. Systémové proměnné jsou všechny proměnné procesu, které jsou přístupné pouze z jednoho procesu. Tato část popisuje systémové proměnné 4th Dimension.

OK

Toto je nejpoužívanější systémová proměnná. Většinou je nastavena na 1 pokud je operace úspěšně dokončena a na 0 pokud není dokončena. Následující příkazy ovlivňují nastavení systémové proměnné OK:

Append document
ADD RECORD
APPLY TO SELECTION
CHANGE ACCESS
QUERY
QUERY BY EXAMPLE
QUERY SELECTION BY FORMULA
COMPRESS BLOB
Create document
EXPAND BLOB
DIALOG
SET PICTURE TO CLIPBOARD
SET RESOURCE PROPERTIES
SET STRING RESOURCE
SET TEXT RESOURCE
SAVE VARIABLES
EXPORT DIF
SEND RECORD
SEND VARIABLE
SELECT LOG FILE
PRINT LABEL
RELATE ONE SELECTION
START WEB SERVER
IMPORT DIF
Get indexed string
GET RESOURCE
GET ICON RESOURCE
Get text resource
LOAD VARIABLES
MODIFY RECORD
CANCEL
Open resource file
RECEIVE RECORD
RECEIVE VARIABLE
RELATE MANY SELECTION
DELETE DOCUMENT
ARRAY TO LIST
ARRAY TO SELECTION
APPEND TO CLIPBOARD
ADD SUBRECORD
BLOB TO DOCUMENT
LOAD SET
QUERY SELECTION
QUERY BY FORMULA
SEARCH BY INDEX
CONFIRM
Create resource file
Request
DOCUMENT TO BLOB
SET RESOURCE NAME
SET RESOURCE
SET PICTURE RESOURCE
SET TEXT TO CLIPBOARD
EXPORT TEXT
EXPORT SYLK
SEND PACKET
REPORT
SET TIMEOUT
PRINT SELECTION
PLAY
IMPORT TEXT
IMPORT SYLK
GET PICTURE FROM LIBRARY
Get string resource
GET PICTURE RESOURCE
Get text from clipboard
STRING LIST TO ARRAY
MODIFY SUBRECORD
Open document
PRINT SETTINGS
RECEIVE PACKET
SET CHANNEL
SAVE SET
DELETE RESOURCE
ARRAY TO STRING LIST
ORDER BY

ORDER BY FORMULA
DISTINCT VALUES
VALIDATE TRANSACTION
USE ASCII MAP
ACCEPT

Document

Document obsahuje dlouhé názvy (cesta+název) posledního souboru otevřeného pomocí těchto příkazů:

Append document
Create document
SAVE VARIABLES
EXPORT DIF
REPORT
PRINT LABEL
IMPORT DIF
LOAD VARIABLES
Open resource file
SET CHANNEL
LOAD SET
Create resource file
EXPORT TEXT
EXPORT SYLK
SELECT LOG FILE
IMPORT TEXT
IMPORT SYLK
Open document
SAVE SET
USE ASCII MAP

FldDelimit

FldDelimit obsahuje ASCII kód, který bude použit jako oddělovač polí během importu nebo exportu dat. Jako výchozí je nastavena hodnota 9, což je ASCII kód Tabelátoru. Pokud chcete tuto hodnotu změnit, přiřaďte novou hodnotu k proměnné FldDelimit.

RecDelimit

RecDelimit obsahuje ASCII kód, který bude použit jako oddělovač záznamů při importu nebo exportu dat. Jako výchozí je tato hodnota nastavena na 13, což je ASCII kód klávesy Return. K použití jiného oddělovače záznamů, zadejte jinou hodnotu pro FldDelimit.

Error

Error může být použit pouze v metodě kde je použit příkaz ON ERR CALL. Tato proměnná obsahuje kód chyby. Seznam kódů chyb 4th Dimension a systémových chyb je v části Kódy chyb..

MouseDown, MouseX, MouseY, KeyCode, Modifiers a MouseProc

Tyto systémové proměnné mohou být použity pouze v metodě instalované příkazem ON EVENT CALL.

• *MouseDown*: Je nastaven na 1 pokud je tlačítko myši stisknuté, jinak je nastaven na 0.
• Pokud je událost *MouseDown*, jsou systémové proměnné *MouseX* a *MouseY* nastaveny na vodorovné a svislé souřadnice místa, kde bylo klepnutí provedeno. Obě hodnoty jsou zobrazeny v bodech a používají místní nastavení okna.
• *KeyCode* je číslo ASCII kódu klávesy, která byla naposledy stisknuta. Pokud je tato klávesa funkce, je KeyCode nastaven na specielní kód. ASCII kódy a kódy kláves funkcí najdete v částech ASCII kódy a Kódy kláves funkcí.
• *Modifiers*: je nastaven na klávesové modifikátory (**Ctrl/Command, Alt/Option, Shift, Caps Lock**). Tato proměnná je při příkazu ON EVENT CALL.
• *MouseProc*: je nastaven na číslo procesu, ve kterém byla poslední událost.

Dále si přečtěte

Sady, Proměnné.

Příkazy a odkazy pro Definice jazyka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Ukazatele

Příkazy a odkazy pro Definice jazyka

Verze 6.0

Ukazatele umožňují rozšířenou cestu (programovou) k přístupu k datům.

Když používáte jazyk, přistupujete k různým objektům - například tabulkám, polím, proměnným a array - pouhým použitím jejich názvu. Je také možné odkazovat se na tyto položky (objekty) a přistupovat k nim bez znalosti jejich názvu. To vám umožňují ukazatele.

Koncept ukazatelů není o moc odlišnější než běžný život. Odkazujete se na různé věci bez znalosti jejich podrobného popisu. Například můžete svému příteli říct "Pojedeme tvým autem" místo "Pojedeme autem se značkou ABC-12-34". V tomto případě se odkazujete na auto se značkou ABC-12-34 použitím "tvoje auto". Výraz "auto se značkou ABC-12-34" je jako název objektu a výraz "tvoje auto" je jako použití ukazatele k odkazu na objekt.

Použití odkazu na objekt, u kterého neznáte přesný název je velmi používané. Váš přítel si může koupit nové auto a výraz "tvoje auto" bude stále platný - stále je to auto a vy s ním můžete dále jezdit. Ukazatele pracují úplně stejně. Ukazatel může například jednou odkazovat na číselné pole s názevem Věk a později na číselné pole s názvem Starý věk. V obou případech se ukazatel odkazuje na číselná data, která můžete použít ve výpočtu.

Ukazatele můžete použít k odkazům na tabulky, pole, array a položky array. Následující tabulka ukazuje příklady ke každému typu dat:

Objekt	**Odkaz na**	**K použití**	**K přiřazení**
Tabulka	vpTabulka:=→[Tabulka]	DEFAULT TABLE(vpTabulka→)	není
Pole	vpPole:=→[Tabulka]Pole	ALERT(vpPole→)	vpPole→:="Jan"
Proměnná	vpProm:=→Proměnná	ALERT(vpProm→)	vpProm→:="Jan"
Array	vpArr:=→Array	SORT ARRAY(vpArr→;>)	COPY ARRAY(Arr;vpArr→)
Položka Array	vpPol:=→Array{1}	ALERT (vpPol→)	vpPol→:="Jan"

Použití ukazatele: Příklad

Nejjednoduší vysvětlení ukazatelů je na příkladu. Tento příklad popisuje jak použít proměnnou přes ukazatel. Začneme vytvoření proměnné:

```
MojeProm:="Ahoj"
```

MojeProm je nyní proměnná, která obsahuje řetězec "Ahoj". Nyní můžeme vytvořit ukazatel na MojeProm:

```
MůjUkazatel:=→MojeProm
```

Symbol → znamená "urči ukazatel na". Tento symbol se skládá ze znaků pomlčky a "menší než". V tomto případě je vytvořen ukazatel na proměnnou MojeProm. Tento ukazatel je přiřazen k MojeProm pomocí operátoru přiřazení.

MůjUkazatel je nyní proměnná která obsahuje ukazatel na MojeProm. MůjUkazatel sice neobsahuje řetězec "Ahoj", který je obsahem MojeProm, ale můžete jej použít k dosazení této hodnoty. Následující výraz vrací hodnotu proměnné MojeProm:

```
MůjUkazatel→
```

V tomto případě vrací hodnotu řetězce "Ahoj". Tento symbol →, pokud následuje za ukazatelem, odkazuje na objekt, pro který je ukazatel zadán. Nazývá se to **DEREFERENCOVÁNÍ**.

Je důležité, aby jste si pamatovali, že ukazatel následovaný symbolem → můžete použít kdekoli potřebujete objekt

(jeho hodnotu), ke kterému je ukazatel určen. To znamená, že můžete použít výraz MůjUkazatel→ kdekoli potřebujete použít hodnotu proměnné MojeProm.

Následující příklad ukazuje upozornění s textem Ahoj:

```
ALERT(MůjUkazatel→)
```

Ukazatel můžete použít také ke změně hodnoty v proměnné MojeProm. Například následující řádek naplní hodnotu "Nashledanou" do proměnné MojeProm:

```
MůjUkazatel→:="Nashledanou"
```

Pokud použijete víckrát výraz MůjUkazatel→, uvidíte, že výsledek je stejný jako by jste použili proměnnou MojeProm. Následující dva řádky provedou stejnou akci - oba zobrazí výstražné okno obsahující hodnotu proměnné MojeProm:

```
ALERT(MůjUkazatel→)
ALERT(MojeProm)
```

Následující dva řádky provedou také stejnou akci - přiřadí do proměnné MojeProm hodnotu "Nashledanou":

```
MůjUkazatel→:="Nahledanou"
MojeProm:="Nashledanou"
```

Použití ukazatelů na tlačítka

Tato část popisuje jak použít ukazatele k odkazům na tlačítka. Tlačítko není nic jiného než proměnná. Ačkoli příklad v této části používá ukazatel na odkazu na tlačítko, je toto způsob použitelný pro jakýmukoliv typ objektů, na který se může ukazatel odkazovat.

Řekněme, že máte více tlačítek ve formuláři a tyto je potřeba aktivovat nebo deaktivovat. Ke každému tlačítku je přiřazena podmínka stavu, která je buď PRAVDA nebo NEPRAVDA. Podmínka říká kdy je tlačítko aktivní a kdy neaktivní. Můžete použít následující test kdykoli potřebujete aktivovat nebo potlačit tlačítko:

```
If (Stav) `Jestliže je podmínka pravda...
   ENABLE BUTTON (MojeTlačítko) `aktivuje tlačítko
ELSE `Jinak ...
   DISABLE BUTTON (MojeTlačítko) `potlačí tlačítko
End if
```

Změnou názvu tlačítka můžete použít tento test pro každé tlačítko, které potřebujete nastavit. Aby jste dosáhli větší efektivity, můžete použít ukazatele pro odkazy na tlačítka a pro vlastní test použít podmetodu.

Pokud použijete podmetodu, musíte použít ukazatel, protože se nemůžete odkazovat k proměnné tlačítka jiným způsobem, než parametrem s ukazatelem. Například tato metoda projektu nazvaná NASTAVIT TLAČÍTKO se odkazuje na tlačítko pomocí ukazatele:

```
   ` NASTAVIT TLAČÍTKO Metoda projektu
   ` NASTAVIT TLAČÍTKO ( Ukazatel ; Logické )
   ` NASTAVIT TLAČÍTKO ( → Tlačítko ; Aktivovat nebo Potlačit )
   ` $1 _ Ukazatel na tlačítku
   ` $2 _ Logické. Jestliže TRUE, aktivovat tlačítko. Jestliže FALSE, potlačit tlačítko
If ($2) ` Jestliže podmínka je TRUE…
```

```
   ENABLE BUTTON($1→) ` aktivovat tlačítko
Else ` Jinak…
   DISABLE BUTTON($1→) ` potlačit tlačítko
End if
```

Metodu NASTAVIT TLAČÍTKO můžete volat takto:

```
   ` ...
NASTAVIT TLAČÍTKO (→bValidate;True)
   ` ...
NASTAVIT TLAČÍTKO (→bValidate;False)
   ` ...
NASTAVIT TLAČÍTKO (→bValidate;([Zákazníci]Příjmení#"")
   ` ...
For ($vlPřepínač;1;20)
   $vpPřepínač:=Get pointer("r"+String($vlPřepínač))
   NASTAVIT TLAČÍTKO ($vpPřepínač;False)
End for
```

Použití ukazatelů na tabulky

Kdekoli jazyk používá tabulku, můžete použít ukazatel na tabulku. Vytvoříte jej následovně:

```
UkzTabulka:=→[Tabulka]
```

Ukazatel můžete vytvořit také pomocí příkazu Table (tabulka):

```
UkzTabulka:=Table(20)
```

Nyní tento ukazatel můžete použít v jiných příkazech:

```
DEFAULT TABLE(UkzTabulka→)
```

Použití ukazatelů na pole

Kdekoli jazyk používá pole, můžete se k němu odkazovat pomocí ukazatele. Ukazatel na pole vytvoříte následovně:

```
UkzPole:=→[Tabulka]Pole
```

Ukazatel na pole můžete vytvořit také pomocí příkazu Field:

```
UkzPole:=Field(1; 2)
```

Ukazatel můžete nyní použít v příkazech místo pole:

```
FONT(UkzPole→;"Arial")
```

Použití ukazatelů na položky (prvky) Array

Ukazatel můžete vytvořit i na prvky Array. Následující příklad ukazuje vytvoření array a přiřazení ukazatele na jeho první prvek do proměnné PolArr:

```
ARRAY REAL(anArray;10) `vytvoření array
PolArr:=→anArray{1} `Vytvoří ukazatel na první položku array
```

Nyní můžete použít tento ukazatel na přiřazení hodnoty k položce array:

```
PolArr→:=8
```

Použití ukazatelů na Array

Ukazatel můžete vytvořit i na array. Například následující řádky vytvoří array a přiřadí ukazatel na něj do proměnné:

```
ARRAY REAL(anArray; 10) ` Vytvoření array
UkzArr := →anArray ` Vytvoření ukazatele na array
```

Je důležité, aby jste pochopili, že tento ukazatel se odkazuje na array a ne na jednotlivé prvky array. Ukazatel na array můžete použít třeba následovně:

```
SORT ARRAY(UkzArr→; >) `Třídí array
```

Pokud se potřebujete odkázat třeba na čtvrtou položku array, můžete to udělat následovně:

```
UkzArr→{4} :=  84
```

Použití array ukazatelů

Velmi používané je také vytvoření array z ukazatelů, které jsou nastavené k různým skupinám objektů s nějakým vztahem.

Dobrý příklad takové skupiny objektů je síť proměnných ve formuláři. Každá proměnná v síti je postupně očíslovaná například Prom1, Prom2, ... Prom10. Většinou potřebujete odkazy k těmto proměnným podle čísel. Jestliže vytvoříte array ukazatelů a každý z ukazatelů nastavíte na jednu proměnnou, můžete potom jednoduše používat jednotlivé proměnné. Pro vytvoření array a nastavení každé položky můžete použít následující kód:

```
ARRAY POINTER(apUkazatel;10)   `Vytvoří array s 10 prvky
For ($i; 1; 10) ` Smyčka pro každou proměnnou
   apUkazatel{$i}:=Get pointer("Prom"+String($i)) ` Nastavení prvků array
End for
```

Funkce Get pointer vrací ukazatel pro každý název objektu.

K odkazování k jednotlivým proměnným používáte položky array. Například k naplnění následujících deseti datumů do proměnných (proměnné musí mít typ datum), můžete použít tento kód:

```
For ($i;1;10) ` Smyčka pro každou proměnnou
   apUkazatel{$i}→:=Current date+$i `Přiřadí datumy
End for
```

Nastavení tlačítka pomocí ukazatele

Pokud máte ve formuláři skupinu přepínačů, obvykle je potřebujete nastavovat rychle. Nejrychlejší je se k nim odkazovat pomocí názvu. řekněme, že máte skupinu přepínačů, které se jmenují Tlačítko1, Tlačítko2,... Tlačítko5.

Ve skupině přepínačů je zapnutý pouze jeden. Číslo přepínače, který je zapnutý může být uloženo v číselném poli. Například pokud pole [Předvolby]Nastavení obsahuje 3, pak je označeno Tlačítko3. Ve vaší metodě formuláře můžete použít následující kód:

```
Case of
   :(Form event=On Load)
      ` ...
      Case of
         ÷ ([Preferences]Setting = 1)
            Button1:=1
         ÷ ([Preferences]Setting = 2)
            Button2:=1
         ÷ ([Preferences]Setting = 3)
            Button3:=1
         ÷ ([Preferences]Setting = 4)
            Button4:=1
         ÷ ([Preferences]Setting = 5)
            Button5:=1
      End case
         ` ...
End case
```

Jednotlivé případy musí být testovány pro každé tlačítko. To může znamenat velmi dlouhou metodu, pokud máte více přepínačů. K vyřešení tohoto problému můžete použít ukazatel. Můžete použít funkci Get pointer k navrácení ukazatele na přepínač. Následující příklad ukazuje použití ukazatele k odkazu na přepínač, který má být nastaven:

```
Case of
      :(Form event=On Load)
            `...
      $vpPřepínač:=Get pointer ("Tlačítko"+String([Předvolby]Nastavení))
      $vpPřepínač→:=1
            `...
End case
```

Které přepínače byly nastaveny musí být uloženo v poli *[Předvolby]Nastavení*. Můžete to udělat v metodě formuláře pro událost *On Clicked (Při klepnutí):*

```
[Předvolby]Nastavení:=Tlačítko1+(Tlačítko2*2)+ (Tlačítko3*3)+ (Tlačítko4*4)+ (Tlačítko5*5)
```

Předání ukazatelů do metody

Ukazatel můžete předat do metody jako parametr. Uvnitř metody můžete upravit objekt který je odkazován ukazatelem. Například následující metoda projektu ZadatDva, obsahuje dva parametry, které jsou ukazatele. Mění objekt, na který odkazuje první ukazatel, na velká písmena a objekt, na který odkazuje druhý ukazatel, na malá písmena. Zde je metoda:

```
` ZadatDva metoda projektu
```

` $1 _ Ukazatel na pole nebo proměnnou. Mění jej na velká písmena.
` $2 _ Ukazatel na pole nebo proměnnou. Mění jej na malá písmena.
$1→:=**Uppercase**($1→)
$2→:=**Lowercase**($2→)

Následující řádek používá metodu ZadatDva k změně pole Pole na velká písmena a k změně proměnné Proměnná na malá písmena:

ZadatDva (→[Tabulka]Pole; →Proměnná)

Pokud pole [Tabulka]Pole obsahuje řetězec "jarda", změní se na "JARDA". Jestliže proměnná Proměnná obsahuje řetězec"AHOJ", změní se na "ahoj".

V metodě ZadatDva, a ve skutečnosti i kdekoliv jinde, když použijete ukazatele, je důležité aby typ dat byl správný. V předchozím příkladu je důležité, aby pole i proměnná obsahovali řetězec nebo text.

Ukazatele na ukazatele

Pokud máte rádi komplikované věci, můžete použít ukazatele, které odkazují na jiné ukazatele. Uvažujme tento příklad:

MojeProm := "Ahoj"
UkazJedna := →MojeProm
UkazDva := →UkazJedna
(UkazDva→)→ := "Nashledanou"
ALERT((UkazDva→)→)

Zobrazí se výstražné okno s textem "*Nashledanou*".

Následuje popis každého řádku v metodě:

MojeProm:="Ahoj"

Zde se vloží řetězec "Ahoj" do proměnné MojeProm.

UkazJedna:=→MojeProm

UkazJedna nyní obsahuje odkaz na MojeProm.

UkazDva:=→UkazJedna

UkazDva (nová proměnná) nyní obsahuje odkaz na ukazatel UkazJedna, který obsahuje odkaz na MojeProm.

(UkazDva→)→:="Nashledanou"

UkazDva→ se odkáže na UkazJedna, který se odkáže na MojeProm. To znamená, že (UkazDva→)→ se odkazuje na MojeProm. Hodnota MojeProm se změní na "Nashledanou".

ALERT ((UkazDva→)→)

Totéž jak v předchozím: UkazDva→ se odkazuje na UkazJedna, která se odkazuje na MojeProm. Proto (UkazDva→)→ je přímý odkaz na Moje prom a ve výstražném okně se objeví obsah MojeProm.

Následující řádek doplní do proměnné MojeProm řetězec "Ahoj":

(UkazDva→)→:="Ahoj"

Následující řádek doplní hodnotu z MojeProm do NováProm:

NováProm:=(UkazDva→)→

Důležité: Vícenásobné odkazování vyžaduje závorky.

Dále si přečtěte

Array, Array a Ukazatele, Konstanty, Řízení průběhu, Typy dat, Identifikátory, Metody, Operátory, Proměnné

Příkazy a odkazy pro Definice jazyka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Identifikátory

Příkazy a odkazy pro Definice jazyka

Verze 6.0

Tato část popisuje konvence pro vytváření názvů objektů v jazyku 4th Dimension. Názvy všech objektů se řídí těmito pravidly:

• Název musí začít písmenem,

• Název může obsahovat písmena, čísla, mezery a podtržítka (_),

• Tečky, lomítka a dvojtečky nejsou povoleny,

• Nejsou povoleny znaky, které se používají jako operátory například * a +,

• 4th Dimension ignoruje poslední mezery

Tabulky

Tabulku značíte uzavřením jejího názvu do hranatých závorek [...]. Název tabulky může být až 31 znaků dlouhý.

Příklady

DEFAULT TABLE ([Objednávky])
INPUT FORM ([Klienti];"Vstup")
ADD RECORD ([Dopisy])

Pole

Pole se použije nejprve zadáním tabulky, ke které pole patří a název pole těsně následuje za názevem tabulky. Název pole může být až 31 znaků dlouhý.

Název pole nezačínejte podtržítkem (_). Tento znak je vyhrazen pro plug-iny. Jakmile 4th Dimension narazí na tento znak na začátku pole v Editoru metod, bude podtržítko odstraněno.

Příklady

[Objednávky]Celkem:=**Sum**([Položky]Celkem)
QUERY([Klienti];[Klienti]Příjmení="Kovář")
[Dopisy]Text:=*Velká písmena* ([Dopisy]Text)

Definovat tabulku před názvem pole je dobrá programovací technika, i když u metod formulářů a metod objektu to není nutné pro pole patřící do hlavní tabulky.

Podtabulky

U podtabulky musíte také nejdříve definovat tabulku ve které je podtabulka. Název podtabulky těsně následuje název tabulky a může být dlouhý až 31 znaků.

Příklady

ALL SUBRECORDS ([Lidé]Děti)
ADD SUBRECORD ([Klienti]Telefony;"Přidat jeden")
NEXT SUBRECORD ([Dopisy]KlíčSlova)

Podtabulka je považována za typ pole: proto se na ní při použití ve formuláři vztahují stejná pravidla jako na pole. Pokud definujete podtabulku v tabulce, formuláři nebo metodě objektu "rodičovské" tabulky, nepotřebujete zadat název tabulky ve které je podtabulka. Nicméně je dobrá programovací technika definovat názvev tabulky před názvem podtabulky.

Podpole

Podpole se označuje stejným způsobem jako pole. U podpole musíte nejdříve definovat podtabulku, ke které podpole patří. Podpole následuje za podtabulkou a jsou odděleny apostrofem (). Název podpole může být až 31 znaků dlouhý.

Příklady

[Lidé]Děti'Jméno:=**Uppercase**([Lidé]Děti'Jméno)
[Klienti]Telefony'Číslo:="34658723"
[Dopisy]KlíčSlova'Slovo:VelkáPísmena ([Dopisy]KlíčSlova'Slovo)

Pokud definujete podpole v podtabulce, formuláři nebo metodě objektu podtabulky, nepotřebujete definovat název podtabulky. Ačkoli je dobrá programovací technika definovat tabulku a podtabulku před názvem podpole.

Meziprocesové proměnné

Meziprocesové proměnné se použijí znaky <> - "menší než" následované "větší než" - a názvem proměnné.

Poznámka: Tento zápis funguje jak na Windows tak na Macintoshi, ale na Macintoshi můžete použít i diamant (Option-Shift-V na US klávesnici).

Meziprocesové proměnné mohou být až 31 znaků dlouhé, nepočítaje znaky <>.

Příklady

<>vProcID:=**Curent process**
<>vsKlíč:=**Char**(KeyCode)
If (<>vtJméno#"")

Procesové proměnné

Procesové proměnné se použují svým názvem (nesmí začínat znaky <> nebo $). Název proměnné může být až 31 znaků dlouhý.

Příklady

If (bVal=1)
vsPlJméno:=""

Místní proměnné

Místní proměnná se označuje znakem dolar ($) následovaným názvem proměnné. Název místní proměnné může být až 31 znaků dlouhý, bez znaku dolar.

Příklady

```
For ($vlZázn; 1; 100)
If ($vProm="Ne")
$vMůjŘet:="Ahoj tam"
```

Array

Array označíte napsáním jeho názvu, který jste zadali při vytváření (jako ARRAY LONGINT). Array jsou proměnné a jako u normálních proměnných, mají několik typů:

- Meziprocesové array
- Procesové array
- Místní array

Meziprocesové array

Název meziprocesové array musí začínat znaky <> - "menší než" následované "větší než".

Poznámka: Tento zápis funguje jak na Windows tak na Macintoshi, ale na Macintoshi můžete použít i diamant (Option-Shift-V na US klávesnici).

Název meziprocesového array může být až 31 znaků dlouhý, bez znaků <>.

Příklady

```
ARRAY TEXT (<>atSubjekt;Records in table([Hlavní]))
SORT ARRAY (<>asKlíčSlov; >)
ARRAY INTEGER (<>aiBigArray;10000)
```

Procesové array

Procesové array se použije napsáním názvu (nesmí začínat znaky <> nebo $). Název procesové proměnné může být až 31 znaků dlouhý.

Příklady

```
ARRAY TEXT (atSubjekt;Records in table([Hlavní]))
SORT ARRAY (asKlíčSlov; >)
ARRAY INTEGER (aiBigArray;10000)
```

Místní array

Název místní array musí začínat zankem dolar ($) a může být dlouhý až 31 znaků.

Příklady

```
ARRAY TEXT ($atSubjekt;Records in table([Hlavní]))
SORT ARRAY ($asKlíčSlov; >)
ARRAY INTEGER ($aiBigArray;10000)
```

Prvek array

Prvky všech typů array se musí zadávat do složených závorek {}. Prvek je definován číslem pořadí.

Příklady

```
    ` Adresování prvku meziprocesového array
If (<>asKeySlova{1}="Stop")
    <>atSubjects{$vlElem}:=[Topics]Subject
    $viNextValue:=<>aiBigArray{Size of array(<>aiBigArray)}

    ` Adresování prvku procesového array
If (asKeySlova{1}="Stop")
    atSubjects{$vlElem}:=[Topics]Subject
    $viNextValue:=aiBigArray{ Size of array(aiBigArray)}

    ` Adresování prvku místního array
If ($asKeySlova{1}="Stop")
    $atSubjects{$vlElem}:=[Topics]Subject
    $viNextValue:=$aiBigArray{ Size of array($aiBigArray)}
```

Prvky dvourozměrných array

Na prvky dvourozměrných array se odkazujete pomocí dvou složených závorek {}{}. Položka je označena dvěma číselnými výrazy ve dvou složených závorkách.

Příklady

```
    ` Adresování dvourozměrného meziprocesového array
If (<>asKeySlova {$vlNextRow}{1} ="Stop")
    <>atSubjects{10}{$vlElem}:=[Topics]Subject
    $viNextValue:=<>aiBigArray {$vlSet}{ Size of array (<>aiBigArray$vlSet)}

    ` Adresování dvourozměrného procesového array
If (asKeySlova {$vlNextRow}{1} ="Stop")
    atSubjects {10}{$vlElem}:=[Topics]Subject
    $viNextValue:=aiBigArray {$vlSet}{ Size of array (<>aiBigArray$vlSet)}

    ` Adresování dvourozměrného místního array
If ($asKeySlova {$vlNextRow}{1} ="Stop")
    $atSubjects {10}{$vlElem}:=[Topics]Subject
    $viNextValue:=$aiBigArray {$vlSet}{ Size of array (<>aiBigArray$vlSet)}
```

Formuláře

Formulář použijete pomocí řetězce který reprezentuje jeho název. Název formuláře může být až 31 znaků dlouhý.

Příklady

```
INPUT FORM([Lidé];"Vstup")
OUTPUT FORM([Lidé]; "Výstup")
DIALOG([Uložení];"Poznámkový blok"+String($vlStage))
```

Metody

Metodu (proceduru nebo funkci) použijete pomocí jejího názvu. Název metody může být dlouhý až 31 znaků.

Poznámka: Metoda která nevrací hodnotu se nazývá **procedura** a metoda která vrací hodnotu se nazývá **funkce.**

Příklady

```
If (Nový klient)
   VYMAZAT DVOJITÉ HODNOTY
   APPLY TO SELECTION ([Zaměstnanci];ZVÝŠIT PLATY)
```

Tip: Dobrá programovací technika je použít stejný způsob zápisu jako u příkazů 4th Dimension. Používejte všechna velká písmena pro názvy vašich metod; pokud je metoda funkce, zvětšte pouze první písmeno. Pokud tak uděláte, a až databázi otevřete po několika měsících, budete vědět která z vašich metod vrací hodnotu pouze podle jejího názvu v Průzkumníku.

Příklady

```
   ` Tento příkaz očekává metodu (funkci) nebo výraz
QUERY BY FORMULA ([aTabulka];Special query)
   ` Tento příkaz vyžaduje metodu (proceduru) nebo tvrzení
APPLY TO SELECTION ([Zaměstnanci];INCREASE SALARIES)
   ` Tento výraz vyžaduje název metody
ON EVENT CALL ("HANDLE EVENTS")
   ` Tento příkaz plug-inu vyžaduje název metody
WR ON ERROR ("WR HANDLE ERRORS")
```

Metody mohou přijímat parametry (argumenty). Parametry jsou předávány do metody v závorkách, následující název metody. Každý parametr je od druhého oddělený středníkem (;). Parametry jsou přístupné ve volané metodě jako postupné místní proměnné: $1, $2, .$n Více za sebou jdoucích parametrů může být adresováno se syntaxí *${n}* kde n, číselný výraz, je číslo parametru.

Pokud je uvnitř metody definována místní proměnná *$0* a tato obsahuje hodnotu, bude toto hodnota metodou (a tímto funkcí) navrácena.

Příklady

```
   ` Uvnitř DROP SPACES je $1 ukazatel na poli [Lidé]Příjmení
DROP SPACES (→[Lidé]Příjmení)
   ` Uvnitř Calc creator:
   ` - $1 je číselný a rovná se 1
   ` - $2 je číselný a rovná se 5
   ` - $3 je text nebo řetězec a je "Nice"
   ` - Výsledná hodnota je přiřazena do $0
$vsResult:=Calc creator (1; 5; "Nice")
   ` Uvnitř Dump:
   ` - Tři parametry jsou text nebo řetězec
   ` - Mohou být použity jako $1, $2 or $3
   ` - Mohou být také použity jako, Například, ${$vlParam} kde $vlParam je 1, 2 nebo 3
   ` - Výsledná hondota je přiřazena k $0
vtClone:=Dump ("je"; "před"; "to")
```

Příkazy Plug-in (Externí procedury, funkce a oblasti)

Příkaz plug-in (zásuvných modulů) použijete napsáním jejich názvu definovaného v plug-in. Tento příkaz může být

až 31 znaků dlouhý.

Příklady

WR BACKSPACE (wrArea; 0)
$spNováOblast:=***SP New affscreen area***

Sady (set)

Z pohledu rozsahu platnosti jsou dva typy sad:

• Meziprocesové sady

• Procesové sady

Na 4D Server také existují:

• Sady klienta

Meziprocesové sady

Sada je meziprocesová, pokud je před jejím názvem znak (<>) - "menší než" následované "větší než".

Poznámka: Tento typ zápisu může být použit jak na Windows tak na Macintoshi, ale na Macintoshi můžete ještě použít diamant (Option-Shift-V na US klávesnici).

Název meziprocesové sady může být až 80 znaků dlouhý, nepočítaje znaky (<>).

Procesové sady

Procesové sady použijete napsáním jejího názvu (nemůže začínat znaky <> nebo $). Název může být až 80 znaků dlouhý.

Sada klienta

Název sady klienta musí začínat znakem dolaru ($) a může být až 80 znaků dlouhý.

Poznámka: Ve 4D Client/Server do verze 6. byly sady udržovány na počítači klienta, kde byly vytvořeny. Od verze 6. jsou sady udržovány na počítači serveru. V mnoha případech, pro výkonnost nebo zvláštní důvody, můžete potřebovat pracovat se sadou pouze na stroji klienta. Aby jste toho dosáhli, použijte sadu klienta.

Příklady

```
` Meziprocesová sada
USE SET("<>Mazat")
CREATE SET([Zákazníci];"<>Objednávky")
If (Records in set("<>Selection"+String($i))>0)

` Procesová sada
USE SET(" Mazat ")
CREATE SET([Zákazníci];" Objednávky")
If (Records in set("<>Selection"+String($i))>0)

` Sada klienta
USE SET("$Mazat ")
CREATE SET([Zákazníci];"$ Objednávky ")
If (Records in set("$Selection"+String($i))>0)
```

Pojmenovaný výběr

Z pohledu rozsahu platnosti jsou dva typy pojmenovaných výběrů:

• Meziprocesový pojmenovaný výběr

• Procesový pojmenovaný výběr

Meziprocesový pojmenovaný výběr

Pojmenovaný výběr je meziprocesový pokud jsou před jeho názvem napsané symboly (<>) - "menší než" následované "větší než".

Poznámka: Tento zápis může být použit jak na Macintoshi tak na Windows, ale na Macintoshi můžete použít i diamant (Option-Shift-V na US klávesnici).

Název meziprocesového pojmenovaného výběru může být až 80 znaků dlouhý, bez symbolů <>.

Procesový pojmenovaný výběr

Procesový pojmenovaný výběr použijete napsáním řetězce, kterým bude jeho název (nesmí začínat symboly <> a $). Název může být dlouhý až 80 znaků.

Příklady

```
` Meziprocesový pojmenovaný výběr
USE NAMED SELECTION([Zákazníci];"<>ByZipcode")

` Procesový pojmenovaný výběr
USE NAMED SELECTION([Zákazníci];"ByZipcode")
```

Procesy

Jak v jednouživatelské verzi i verzi klient/server na počítači klienta existují dva typy procesů:

• Globální procesy

• Místní procesy

Globální procesy

Globální procesy označíte napsáním jejich názvu (nesmí začínat znakem $). Název procesu může být až 31 znaků dlouhý.

Místní procesy

Místní procesy označíte napsáním jejich názvu, který musí začínat znakem dolar ($). Název může být až 31 znaků dlouhý.

Příklady

```
` Zapnutí globálního procesu "Přidat Zákazníka"
$vlProcessID:=New process("P_PRIDAT_ZAKAZNIKA";48*1024;" Přidat Zákazníka ")

` Zapnutí místního procesu "$Sleduj Myš"
$vlProcessID:=New process("P_SLEDUJ_MYS";16*1024;"$Sleduj Myš ")
```

Shrnutí konvencí názvů

Následující tabulka shrnuje konvence názvů ve 4th Dimension:

Typ	Max. délka	Příklad
Tabulka	*31*	*[Faktury]*
Pole	*31*	*[Faktury]Datum*
Podtabulka	*31*	*[Přátelé]Děti*
Podpole	*31*	*[Zákazníci]Děti'Jméno*
Meziprocesová proměnná	*<> + 31*	*<>mpIDProces*
Procesová proměnná	*31*	*ppPlatnéJméno*
Místní proměnná	*$ + 31*	*$pMísto*
Formulář	*31*	*"Web vstup"*
Meziprocesový array	*<> + 31*	*<>maTabulky*
Procesový array	*31*	*paDatum*
Místní array	*$ + 31*	*$aHodnoty*
Metoda	*31*	*M_PŘIDAT_ZÁK*
Rutina Plug-in	*31*	*WR INSERT TEXT*
Meziprocesová sada	*<> + 80*	*"<>Záznamy k uchování"*
Procesová sada	*80*	*"Označené záznamy"*
Sada klienta	*$ + 80*	*"$Předchozí"*
Pojmenovaný výběr	*80*	*"Zaměstnanci A do Z"*
Meziprocesový pojmenovaný výběr	*<> + 80*	*"<>Zaměstnanci A do Z"*
Místní proces	*$ + 31*	*"$Události"*
Globální proces	*31*	*"P_FAKTURY"*

Zabránění konfliktu názvů

Pokud nějaký objekt má stejný název jako jiný objekt jiného typu (například pole s názvem Osoba a proměnná s názvem Osoba), 4th Dimension použije systém priorit k určení objektu. Proto si musíte být jisti, že každý název který použijete je jedinečný.

4th Dimension identifikuje názvy použité v metodě v následujícím pořadí:

1. Pole
2. Příkazy
3. Metody
4. Rutiny Plug-in
5. Předdefinované konstanty
6. Proměnné

4th Dimension Například obsahuje příkaz Date. Pokud nazvete metodu Date, 4th Dimension bude předpokládat, že používáte příkaz a ne metodu. Toto vám zabrání ve volání metody. Pokud nazvete pole Date, bude 4th Dimension předpokládat, že chcete použít pole a ne příkaz nebo metodu.

Dále si přečtěte

Array, Konstanty, Typy dat, Metody, Operátory, Ukazatele, Proměnné.

Příkazy a odkazy pro Definice jazyka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Řízení průběhu

Příkazy a odkazy pro Definice jazyka

Verze 6.0

Bez ohledu na jednoduchosti nebo složitost metody budete muset použít jeden nebo více ze třech typů strukturování programování. Struktura programu (metody) řídí průběh metody, co a v jakém pořadí se bude provádět. Existují tyto tři typy:

• Postupná

• Větvená

• Opakující se

Programovací jazyk 4th Dimension obsahuje nástroje pro řízení všech těchto struktur.

Postupná struktura

Postupná struktura je jednoduchá lineární struktura. Pořadí je určeno jednotlivými řádky, které se budou provádět jeden za druhým postupně. Například:

```
OUTPUT FORM([Lidé]; "Seznam")
ALL RECORDS([Lidé])
DISPLAY SELECTION([Lidé])
```

V jednořádkových rutinách, které jsou obvyklé v metodách objektu, je výborný příklad postupné struktury. Například:

```
[Lidé]Příjmení:=Uppercase([Lidé]Příjmení)
```

Větvená struktura

Tato struktura umožňuje 4D testovat různé podmínky a vynechávat určité části metod podle výsledku. Podmínka je logický výraz a může mít hodnotu buď TRUE (pravda) nebo FALSE (nepravda). Jednou z těchto struktur je i If...Else...End if. If struktura, která umožňuje dvě větve. Další je Case of....Else....End. Case struktura, která umožňuje programu dát se jednou z mnoha větví. Přečtěte si If...Else...End if, Case of...Else...End case.

Opakující se struktura

Při psaní metod je velmi důležité najít části které budete potřebovat opakovat. K splnění této potřeby jsou v jazyku tři nástroje:

• While...End while

• Repeat...Until

• For...End for

Smyčky jsou kontrolovány dvěma způsoby: Smyčka se bude opakovat dokud není splněna určitá podmínka nebo se opakuje v určeném počtu krát. Každá ze smyček může být použita oběma způsoby, ale While a Repeat jsou vhodnější pro ukončení při splnění určité podmínky a smyčky For jsou vhodnější pro určitý počet opakování. Přečtěte si For...End for, Repeat...Until, While...End while.

Dále si přečtěte

Logické operátory, Metody.

Příkazy a odkazy pro Definice jazyka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

If...Else...End if

Příkazy a odkazy pro Definice jazyka

Verze 6.0

Formální zápis pro If...Else...End if je:

```
If (Logický výraz)
   tvrzení
Else
   tvrzení
End if
```

Všiměte si, že část Else je volitelná:

```
If (logický výraz)
   tvrzení
End if
```

Struktura If...Else...End if umožní vaší metodě dát se jednou ze dvou cest. Záleží na tom, jestli je podmínka TRUE nebo FALSE.

Pokud je podmínka TRUE, spustí se tvrzení hned za příkazem a pokud je podmínka FALSE, spustí se tvrzení následované za Else. Vzhledem k tomu, že je Else volitelné, jestli jej vynecháte, bude průběh metody pokračovat ze řádkem End if a provedení trvrzení bude vynecháno.

Příklad

```
   ` Zeptat se uživatele na příjmení
$Find:=Request("Napište příjmení:")
If (OK=1)
   QUERY([Lidé]; [Lidé]Příjmení=$Find)
Else
   ALERT("Nenapsal jste žádné příjmení.")
End if
```

Tip: Přeskakování si nevyžaduje nutně tvrzení pro každou variantu. Pokud potřebujete, můžete napsat i:

```
If (Logický výraz)
Else
   Tvrzení
End if
```

Nebo:

```
If (Logický výraz)
   Tvrzení
Else
End if
```

Dále si přečtěte

Case of...Else...End case, Řízení průběhu, For...End for, Repeat...Until, While...End while.

Příkazy a odkazy pro Definice jazyka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Case of...Else...End case

Příkazy a odkazy pro Definice jazyka

Verze 6.0

Formální zápis struktury Case of...Else...End case je:

```
Case of
   ÷ (Logický výraz)
      tvrzení
   ÷ (Logický výraz)
      tvrzení
.
.
   ÷ (Logický výraz)
      tvrzení
Else
   tvrzení
End case
```

Všiměte si, že části Else je volitelná:

```
Case of
   ÷ (Logický výraz)
      tvrzení
   ÷ (Logický výraz)
      tvrzení
.
.
   ÷ (Logický výraz)
      tvrzení
End case
```

Stejně jako u struktury If...Else...End if vám i tato struktura dává možnosti vybrat z několika možností kudy metoda půjde. Na rozdíl od struktury If...Else...End if můžete u Case of...Else...End case zadat více podmínek a bude otestováno, která z nich je pravdivá a metoda bude pokračovat za ní.

Před každým logickým výrazem musí být znak (÷). Kombinace tohoto znaku a výrazu se nazývá případ (case). Například následující řádek je případ:

```
÷ (bPotvrdit=1)
```

Pouze tvrzení, které je za prvním pravdivým případem se spoustí. Pokud nebude splněna žádná z podmínek, nespustí se žádná část metody mezi Case of a End case (pokud není příkaz Else).

Tvrzení Else můžete vložit za poslední případ. Pokud jsou všechny případy Nepravda, bude spuštěno tvrzení za Else.

Příklad
Tento příklad testuje číselnou proměnnou a zobrazuje výstražné okno s textem:

```
Case of
   ÷ (vResult = 1) ` Testuje jestli je číslo 1
      ALERT("Jedna.") `Pokud je číslo 1, zobrazí se okno
```

```
    ÷ (vResult = 2) ` Testuje jestli je číslo 2
        ALERT("Dva.") ` Pokud je číslo 2, zobrazí se okno
    ÷ (vResult = 3) ` Testuje jestli je číslo 3
        ALERT("Tři.") ` Pokud je číslo 3, zobrazí se okno
Else ` Pokud není číslo 1, 2 nebo 3, zobrazí se okno
    ALERT("Číslo nebylo jedna, dva ani tři.")
End case
```

Následující příklad ukazuje stejnou věc, ale s použitím If...Else...End if:

```
If (vResult = 1) ` Testuje jestli je číslo 1
    ALERT("Jedna.") ` Pokud je číslo 1, zobrazí se okno
Else
    If (vResult = 2) ` Testuje jestli je číslo 2
        ALERT("Dva.") ` Pokud je číslo 2, zobrazí se okno
    Else
        If (vResult = 3) ` Testuje jestli je číslo 3
            ALERT("Tři.") ` Pokud je číslo 3, zobrazí se okno
        Else ` Pokud není číslo 1, 2 nebo 3, zobrazí se okno
            ALERT("Číslo nebylo jedna, dva ani tři.")
        End if
    End if
End if
```

Zapamatujte si, že se strukturou Case of...Else...End case, je spuštěn pouze první případ, který je pravda. Pokud bude pravda dva a více případů, bude spuštěno tvrzení které je za prvním případem.

Tip: Pokud potřebujete, nemusíte psát za případ tvrzení. Můžete napsat třeba toto:

```
Case of
    ÷ (logický výraz)
    ÷ (logický výraz)
    .
    .
    ÷ (logický výraz)
        tvrzení
Else
    tvrzení
End case
```

nebo:

```
Case of
Else
    tvrzení
End case
```

Dále si přečtěte

Řízení průběhu, For...End for, If...Else...End if, Repeat...Until, While...End while.

Příkazy a odkazy pro Definice jazyka

While...End while

Verze 6.0

Formální zápis struktury While...End while je:

While (Logický výraz)
    tvrzení
End while

Smyčka While...End while se opakuje dokud je logický výraz pravda. Testuje logický výraz na začátku smyčky a pokud je Nepravda, tak jí nespustí.

Je důležité zadat hodnotu testovanou v logickém výrazu smyčkou ještě před tím, než předáte samotnou smyčku. Zadání hodnoty znamená její nastavení na nějakou hodnotu. Obvykle když bude Logický výraz Pravda, bude spuštěna smyčka While...End while.

Logický výraz musí být také nastavovaný uvnitř smyčky, jinak vytváříte věčnou smyčku. Následující příklad je věčná smyčka, protože hodnota proměnné NikdyStop je stále Pravda:

```
NikdyStop:=True
While (NikdyStop)
End while
```

Pokud se vám stane, že vytvoříte takovouto smyčku omylem, můžete použít Debugger a zastavit její průběh. Pokud chcete vědět více informací o Ladění, přečtěte si část Ladění.

Příklad

```
CONFIRM ("Přidat nový záznam?") ` Chce uživatel zadat nový záznam?
While (OK = 1) ` Točit dokud uživatel chce vkládat nový záznam
    ADD RECORD([aTabulka]) ` Přidat nový záznam
End while ` Smyčka musí vždy končit End while
```

V tomto případě je systémová proměnná OK nastavována pomocí příkazu CONFIRM (potvrdit) ještě před započetím smyčky. Pokud uživatel klepne na tlačítko OK, je tato proměnná nastavena na 1 a smyčka se spustí. Pokud uživatel neklepne na OK, je proměnná nastavena na 0 a smyčka neproběhne. Jakmile je smyčka spuštěna, je používán příkaz ADD RECORD. Když uživatel klepne na Uložit záznam, je proměnná nastavena opět na 1 a smyčka pokračuje. Když klepne na Zrušit (neuložit záznam), je proměnná OK nastavena na 0 a smyčka se zruší.

Dále si přečtěte

Case of...Else...End case, Řízení průběhu, For...End for, If...Else...End if, Repeat...Until.

Příkazy a odkazy pro Definice jazyka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Repeat...Until

Verze 6.0

Formální zápis struktury Repeat...Until je:

Repeat
tvrzení
Until (logický výraz)

Struktura Repeat...Until je stejná jako While...End while pouze s tím rozdílem, že testuje logický výraz na konci a ne na začátku. Smyčka Repeat...Until se vždy spustí alespoň jednou, While...End while se nespustí, pokud je hodnota logického výrazu Nepravda.

Další rozdíl se smyčkou Repeat...Until je ten, že se bude provádět stále dokola, dokud bude Logický výraz Pravda.

Příklad

Porovnejte tento příklad s příkladem na While...End while. Všiměte si, že zde nemusí být logický výraz definován - není zde žádný příkaz CONFIRM k nastavení proměnné OK:

Repeat
ADD RECORD ([aTabulka])
Until (OK=0)

Dále si přečtěte

Case of...Else...End case, Řízení průběhu, For...End for, If...Else...End if, While...End while.

Příkazy a odkazy pro Definice jazyka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

For...End for

Příkazy a odkazy pro Definice jazyka

Verze 6.0

Formální zápis struktury For...End for je:

```
For (Počítací proměnná; Počáteční hodnota; Koncová hodnota{ ; Hodnota připočtení})
   tvrzení
End for
```

Smyčka For...End for je kontrolovaná počítací proměnnou:

• Počítací proměnná je číselná (Real, Integer, Long Integer), která je nastavená a hodnotu Počáteční hodnota.

• Pokaždé když smyčka proběhne, je hodnota Počítací proměnné zvýšená o hodnotu zadanou v hodnotě připočtení. Pokud není tato hodnota definovaná, je automaticky nastavena na 1.

• Jakmile dosáhne hodnota Počítací proměnné Koncová hodnota, je smyčka zastavena.

Důležité: Proměnné Počáteční hodnota, Koncová hodnota, Hodnota připočtení jsou nastaveny pouze jednou při prvním spuštění smyčky. Pokud by jste je měli zadané jako proměnné a **během** provádění smyčky jste změnili jejich hodnotu, nebude to mít **žádný** vliv.

Tip: Pokud potřebujete můžete změnit hodnotu Počítací proměnné. Tato změna je možná i **během** spuštění smyčky.

• Většinou je hodnota Počáteční hodnota menší než Koncová hodnota,

• Pokud je jejich hodnota shodná, bude smyčka spuštěna pouze jednou,

• Pokud je hodnota Počáteční hodnota větší než Koncová hodnota, nespustí se smyčka ani jednou, pokud nenastavíte Hodnotu připočtení zápornou. Podívejte se na příklady:

Základní Příklady

1) Následující příklad bude spuštěn 100x:

```
For (vPočet;1;100)
   Udělat něco
End for
```

2) Následující příklad se spustí tolikrát, kolik je položek v Array:

```
For ($vlElem;1;Size of array(anArray))
      ` Udělat něco s položkami
   anArray{$vlElem}:=...
End for
```

3) Následující příklad proběhne tolikrát, kolik je znaků v textu vtNejakyText:

```
For ($vlChar;1;Length(vtNejakyText))
      ` Udělat něco se znakem, pokud je to Tabelátor
   If (Ascii(vtNejakyText[[$vlChar]])=Char(Tab))
      ` ...
   End if
End for
```

4) Následující příklad proběhne jednou pro každý záznam ve výběru tabulky [aTabulka]:

```
FIRST RECORD([aTabulka])
For ($vlZaznam;1;Records in selection([aTabulka]))
      ` Udělat něco se záznamem
   SEND RECORD([aTabulka])
      ` ...
      ` Jít na další záznam
   NEXT RECORD([aTabulka])
End for
```

Většina smyček For...End for které napíšete ve vaší databázi bude vypadat jako jeden z těchto příkladů.

Snižování počítací proměnné

V některých případech nemusíte chtít Počítací proměnnou zvětšovat, ale zmenšovat. Aby jste toho docílili, musíte zadat Počáteční hodnotu větší než koncovou hodnotu a zápornou Hodnotu připočtení. Následující příklady dělají to samé jako předchozí, ale v opačném, pořadí:

5) Následující příklad bude spuštěn 100x:

```
For (vPočet;100;1;-1)
   Udělat něco
End for
```

6) Následující příklad se spustí tolikrát, kolik je položek v Array:

```
For ($vlElem;Size of array(anArray) ;1;-1)
      ` Udělat něco s položkami
   anArray{$vlElem}:=...
End for
```

7) Následující příklad proběhne tolikrát, kolik je znaků v textu vtNejakyText:

```
For ($vlChar;Length(vtNejakyText); 1;-1)
      ` Udělat něco se znakem, pokud je to Tabelátor
   If (Ascii(vtNejakyText[[$vlChar]])=Char(Tab))
      ` ...
   End if
End for
```

8) Následující příklad proběhne jednou pro každý záznam ve výběru tabulky [aTabulka]:

```
LAST RECORD([aTabulka])
For ($vlZaznam;Records in selection([aTabulka]);1;-1)
      ` Udělat něco se záznamem
   SEND RECORD([aTabulka])
      ` ...
      ` Jít na další záznam
   PREVIOUS RECORD([aTabulka])
End for
```

Zvětšení Počítací proměnné o více než jedna

Pokud potřebujete můžete použít hodnotu připočtení (kladnou nebo zápornou) které absolutní hondota bude větší než 1.

9) Následující příklad pracuje pouze se sudými položkami array anArray:

```
For ($vlElem;2;((Size of array(anArray)+1))*2;2)
      ` Udělat něco s položkami #2,#4...#2n
   anArray{$vlElem}:=...
End for
```

Všiměte si, že konec výrazu ((**Size of array**(anArray)+1)\2)*2 se stará o sudé a liché prvky

Zastavení smyčky upravením počítací proměnné

V některých případech potřebujete spustit smyčku v určitém počtu, ale tento počet přerušit, jakmile nějaká jiná podmínka bude Pravda. Aby jste toho docílili musíte tuto podmínku testovat uvnitř smyčky a jakmile bude splněna, tak doplnit počítací proměnnou na hodnotu, která smyčku zastaví.

V následujícím příkladu se bude procházet výběr záznamů tak dlouho dokud se nedokončí a nebo pokud nebude proměnná <>vbZastav, nastavená na FALSE, mít hodnotu TRUE. Tato proměnná je ovládána metodou projektu ON EVENT CALL která vám umožní zastavit operaci:

```
<>vbZastav:=False
ON EVENT CALL ("HANDLE STOP")
      ` HANDLE STOP nastaví <>vbZastav na True pokud stisknete Ctrl-tečka
(Windows) nebo Cmd-tečka (Macintosh)
$vlPocZazn:=Records in selection([aTabulka])
FIRST RECORD([aTabulka])
For ($vlZaznam;1;$vlPocZazn)
      ` Udělat něco se záznamem
   SEND RECORD([aTabulka])
   ` ...
   ` Jít na další záznam
   If (<>vbZastav)
      $vlZaznam:=$vlPocZazn+1 ` Nastavit počítací proměnnou na hodnotu k zastavení smyčky
   Else
      NEXT RECORD([aTabulka])
   End if
End for
ON EVENT CALL("")
If (<>vbZastav)
   ALERT("Operace byla přerušena.")
Else
   ALERT("Operace byla dokončena.")
End if
```

Porovnání opakujících se struktur

Pojďme zpět na první příklad For...End for:

Následující příklad se spustí 100x:

```
For (vPočet;1;100)
   Udělat něco
End for
```

Je zajímavé se podívat, jak struktury , Repeat...Until, While...End whileprovedou stejnou akci:

```
$i := 1 ` Nastavit počítání
While ($i<=100) ` Otočit 100x
      `Něco udělat
   $i := $i + 1 ` Potřebuje zvětšit počítání
End while
```

Zde je to samé se strukturou Repeat...Until:

```
$i := 1 ` Nastavit počítání
Repeat
      ` Něco udělat
   $i := $i + 1 ` Potřebuje zvětšit počítání
Until ($i=100) ` Otočit 100x
```

Tip: Struktura For...End for je v těchto případech rychlejší než Repeat...Until nebo While...End while, protože 4th Dimension testuje počet smyček vnitřně a stejně i přidává jednotlivé. Proto používejte strukturu For...End for kdykoli to bude možné.

Optimalizace provádění smyčky For...End for

Můžete použít Real, Integer nebo Long Integer stejně jako meziprocesní, procesní nebo místní proměnné jako počítadlo. Pro mnohokrát opakované smyčky je nejlepší použít místní proměnmnou Long Integer.

11) Zde je příklad:

```
C_LONGINT($vlCounter) ` použít místní proměnnou Long integer
For ($vlCounter;1;10000)
   ` Do something
End for
```

Vnořené struktury For...End for

Můžete do sebe vnořit tolik řídících struktur kolik (rozumně) potřebujete. Aby jste se vyhnuli problémům, ujistěte se, že používáte jinou počítací proměnnou pro každý případ vnořené smyčky.

Následují dva příklady:

12) Následující příklad jde přes všechny položky dvou-rozměrného array:

```
For ($vlElem;1;Size of array(anArray))
   ` ...
   ` Udělat něco s řadou
   ` ...
   For ($vlSubElem;1;Size of array(anArray{$vlElem}))
         ` Udělat něco s položkou
      anArray{$vlElem}{$vlSubElem}:=...
   End for
End for
```

13. Následující příklad vytvoří array ukazatelů ke všem datumovým polím zastoupeným v databázi:

```
ARRAY POINTER($apDateFields;0)
$vlElem:=0
For ($vlTabulka;1;Count tables)
```

```
      For($vlField;1;Count fields($vlTabulka))
         $vpPole:=Field($vlTabulka;$vlField)
         If (Type($vpPole→)=Is Date)
            $vlElem:=$vlElem+1
            INSERT ELEMENT($apDateFields;$vlElem)
            $apDateFields{$vlElem}:=$vpPole
         End if
      End for
   End for
```

Dále si přečtěte

Case of...Else...End case, Řízení průběhu, If...Else...End if, Repeat...Until, While...End while.

Příkazy a odkazy pro Definice jazyka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Metody

Příkazy a odkazy pro Definice jazyka

Verze 6.0

Aby všechny části jazyka, jako příkazy, operátory a další části, spolu pracovaly vkládáte je do metod. Existuje mnoho typů metod: Metody objektu, Metody formuláře, Metody tabulky (Triggery), Metody projektu a Metody databáze. Tato část popisuje všechny typy metod.

Všechny metody jsou složeny z jednotlivých **tvrzení**: každé tvrzení je dlouhé jeden řádek. Tvrzení provádí různé akce a může být jednoduché nebo složené. Tvrzení je vždy jeden řádek a může být dlouhé jak potřebujete (až 32000 znaků, což je provděpodobně dost pro každé tvrzení).

Například následující tvrzení přidá nový záznam do tabulky [Lidé]:

```
ADD RECORD([Lidé])
```

Metoda obsahuje také testy a smyčky, které kontrolují průběh a směr metody. Jestli chcete vědět více informací o řízení průběhu, přečtěte si část Řízení průběhu.

Typy metod

Existuje pět typů metod:

• **Metody objektu**: Metoda objektu je vlastnost objektu. Je to obvykle krátká metoda připojená k aktivnímu objektu ve formuláři. Může řídit objekt při jeho zobrazení nebo tištění. Metodu objektu nemůžete přímo volat - dělá to sama 4D při splnění určité události přiřazené k tomuto objektu.

• **Metoda formuláře**: Metoda formuláře je vlastnost formuláře. Pomocí této metody můžete řídit data a objekty, ale většinou je efektivnější použít k tomuto metody objektu. Metodu formuláře nemůžete přímo volat - dělá to sama 4D při splnění určité události přiřazené k tomuto formuláři.

Jestli chcete vědět více informací o metodách objektu a formuláře, přečtěte si *Příručku návrháře 4th Dimension* část Události formuláře.

• **Metoda tabulky (trigger)**: Trigger je vlastnost tabulky. Trigger nemůžete volat - dělá to jádro 4D při každé manipulaci se záznamy tabulky (Přidat, Vymazat, Upravit a Načíst). Triggery jsou metody, pomocí kterých můžete zabránit nesprávným operacím se záznamy tabulky. Například ve fakturačním systému můžete použít triggery k tomu, abyste zabránili uživateli zadat fakturu bez toho aby zadal zákazníka, pro kterého je faktura určena. Trigger je silný nástroj k řízení operací s tabulkou, stejně jako k zabránění ztráty dat nebo neoprávněnému zásahu. Můžete začít s napsáním jednoduchého triggeru a postupně k němu přidávat části.

Pro podrobnější informace o Triggerech si přečtěte část Triggery.

• **Metody projektu**: Na rozdíl od metod objektu, formuláře a tabulky, které jsou svázány s jedním objektem, formulářem nebo tabulkou, jsou metody projektu přístupné z celé databáze. Můžete je používat několikrát a použít je i v jiných metodách. Pokud potřebujete několikrát provést jednu úlohu, nemusíte psát pokaždé novou metodu. Stačí zavolat metodu projektu - z jiné metody projektu, objektu nebo formuláře. Tím že metodu projektu zavoláte, je to jako by jste jí celou napsali na místo, kam předáte její název. Metody projektu, které jsou volány jinými metodami se nazývají "podprogramy". Metoda, která vrací nějakou hodnotu se také nazývá **funkce**.

Existuje ještě jedno využití metod projektu - jejich přiřazení k položkám nabídek. Jakmile přiřadíte metodu projektu k položce nabídky, je tato metoda spuštěna vždy při použití této položky nabídky. Položky nabídek jsou v podstatě jenom odkazy na metody projektu.

Jestli chcete vědět více informací o metodách projektu si přečtěte část Metody projektu.

• **Metody databáze**: Stejně jako metody objektu a formuláře jsou metody databáze prováděny při určité události. Jsou to metody přiřazené k databázi a provádí se při určitých akcích. To jsou **metody databáze**. Například pokaždé když provádíte samotnou databázi, potřebujete nastavit určité proměnné, které budou použity během práce s databází. Aby jste toho docílili, můžete použít metodu databáze *Při spuštění*, která se automaticky spustí při otevření databáze.

Jestli chcete vědět více informací o metodách databáze, přečtěte si část Metody databáze.

Kompatibilita s předchozí verzí 4D

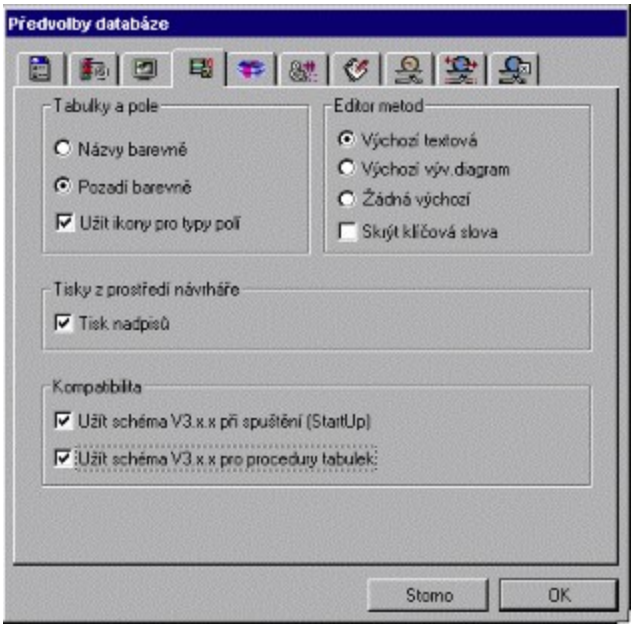
Tuto část můžete přeskočit, pokud pracujete s databází, která byla vytvořena novou 4th Dimension verze 6.

1. Verze 6 obsahuje spoustu nových objektů a událostí formuláře (jako Při poklepání, Při získání focusu, atd.) které nahradí prováděcí kruhy z předchozí verze. Pokud převedete databázi z verze 3 na verzi 6, vaše formuláře se převedou se stejnými vlastnostmi.

Pokud chcete použít nové události formulářů a objektů, musíte je označit v oknech **Vlastnosti formuláře** a **Vlastnosti objektu**.

2. Metoda tabulky, trigger, je nový typ metod uvedený ve verzi 6. V předchozí verzi 4D byly metody tabulky (procedury souboru) prováděny pouze 4D, když byl použit formulář tabulky pro vstup, zobrazení nebo tisk dat. Byly používány velmi vyjímečně. Všiměte si, že triggery jsou prováděny na nižší úrovni než staré procesury souboru. Nezáleží na tom, jestli jsou data upravována přímo, nebo programově (např. při použití příkazu SAVE RECORD), 4D bude automaticky provádět trigger. Tímto jsou triggery naprosto odlišné od starých procedur souboru. Pokud máte databázi, která je překonvertovaná z verze 3 a nižší a chcete používat nové vlastnosti triggerů, musíte odškrtnout tlačítko **Užít staré schéma pro procedury tabulek** v **Předvolbách databáze**.

3. Metody databáze jsou nové ve verzi 6. V předchozí verzi 4th Dimension byla pouze jedna metoda (procedura) která se prováděla při zapnutí databáze a nazývala se **STARTUP** (Intl, US verze) a **DEBUT** (francouzká verze). Pokud jste překonvertovali vaší databázi z verze 3 do verze 6 a chcete použít nové schéma metod databáze, musíte odškrtnout tlačítko **Užít staré schéma při spuštění (STARTUP)** v **Předvolbách databáze**. Tato vlastnost ovlivňuje pouze metodu databáte STARTUP/Při spuštění. Pokud například i volbou zpětné kompatibility verzí předáte kód do metody databáze Při ukončení, bude tato metoda spuštěna před vypnutím databáze.



Příklad metody projektu

Všechny metody jsou stejné v tom, že se spustí na prvním řádku a pokračují s případným určitým cyklením a větvením až na poslední (probíhají postupně). Zde je příklad metody projektu:

```
QUERY ([Lidé]) ` Zobrazí editor vyhledávání
If (OK=1) ` Uživatel klepne na Dotaz ne na Storno
   If (Records in selection([Lidé])=0) ` Pokud nebyl nalezen žádný záznam…
      ADD RECORD([Lidé]) ` Umožní uživateli zadat nový záznam
   End if
End if ` Konec metody
```

Každý řádek v příkladu se nazývá tvrzení nebo řádek kódu. Cokoli napíšete pomocí jazyka je volně považováno za kód. Kód je spuštěn, to znamená že 4th Dimension splní úkol, který je zadán pomocí kódu.

První řádek popíšeme podrobněji a ostatní pak probereme o něco rychleji.

```
QUERY ([Lidé]) ` Zobrazí editor vyhledávání
```

První položka na řádku je QUERY což je příkaz. Příkaz je součástí jazyka 4th Dimension - plní určitou funkci. V tomto případě zobrazí příkaz QEURY, Editor vyhledávání (dotazů). Má stejnou funkci jako položka Dotaz v nabídce Dotazy v prostředí uživatele.

Další položka v tomto řádku je umístěna do závorek a je to argument k příkazu QUERY. Argument (**parametr**) je žádost na data k doplnění dotazu. V tomto případě, [Lidé] je název tabulky. Název tabulek je vždy psán do hranatých závorek ([...]). V našem příkladu je tabulka Lidé argumet k příkazu QUERY. Příkaz může přijmout více parametrů.

Třetí položka je komentář na konci řádku. Komentář vám řekne (nebo komukoli kdo bude číst váš kód) co se stane na tomto řádku. Komentáře se píšou za obrácený apostrof (`). Cokoli je napsáno za tímto znakem (na tomto řádku) bude při provádění kódu ignorováno. Komentář může být na samostatném řádku nebo jej můžete napsat za kód, jak je v tomto řádku. Komentáře je důležité používat aby jste se vy sami, nebo někdo jiný, vyznali ve vašich kódech.

Poznámka: Komentář může být dlouhý 80 znaků.

Další řádek metody testuje jestli byli nalezeny nějaké záznamy.

```
If (Records in selection([Lidé])=0) ` Pokud nebyl nalezen žádný záznam…
```

Tvrzení If je **tvrzení řízení průběhu** - tvrzení které řídí další kroky provádění metody. **If** testuje určitou podmínku a pokud je tato podmínka pravda, metoda pokračuje na následujícím řádku. Records in selection je funkce - příkaz který vrací hodnotu. Zde vrací funkce Records in selection počet záznamů v aktuálním výběru pro vybranou tabulku.

Poznámka: Všiměte si, že je zvětšeno pouze první písmeno funkce. To je standardní zápis ve 4th Dimension.

Nejprve by jste měli vědět co to je aktuální (platný) výběr - je to výběr záznamů určité tabulky se kterým právě pracujete. Pokud je počet záznamů v tomto výběru rovný 0 (jinými slovy nebyl nalezen žádný záznam) pak se spustí další řádek metody.

```
ADD RECORD([Lidé]) ` Umožní uživateli zadat nový záznam
```

Příkaz ADD RECORD zobrazí vstupní formulář do nějž může uživatel zadat data pro nový záznam. 4th Dimension formátuje automaticky váš kód: všiměte si, že na tomto řádku je vidět, že jeho spuštění je závislé na tvrzení řízení průběhu (If).

```
End if ` Konec metody
```

End if je součást if tvrzení řízení průběhu. Pokud někam napíšete první část (If, Case of, While, atd.) musíte také jazyku 4th Dimension říci, kde má tato část skončit.

Ujistěte se, že je vám tato část naprosto jasná. Pokud ne tak si tuto část raději přečtěte ještě jednou dokud vám nebude jasná.

Kam jít nyní?

Přečtěte si více o:

• Metodách objektu a formuláře v Události formuláře

• Triggerech v části Triggery

• Metodách projektu v části Metody projektu

• Metodách databáze v části Metody databáze

Dále si přečtěte

Array, Konstanty, Řízení průběhu, Typy dat, Metody databáze, Identifikátory, Operátory, Ukazatele, Triggery, Proměnné.

Příkazy a odkazy pro Definice jazyka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Metody projektu

Příkazy a odkazy pro Definice jazyka

Verze 6.0

Metody projektu jsou vždy pojmenované. Narozdíl od metod objektu a formuláře, které jsou svázané s objektem nebo formulářem jsou metody projektu dostupné pro celou databázi - nejsou přiřazeny k žádnému objektu v databázi. Metody projektu mohou mít následující úlohu:

- Metoda nabídky
- Podprogramy a funkce
- Metody procesu
- Metody zachycující události
- Metody zachycující chyby

Toto způsob neodlišuje metody projektu podle toho co jsou, ale podle toho co dělají.

Metoda nabídky je metoda projektu přiřazená k položce nabídky. Řídí průběh vaší aplikace. Metoda nabídky může řídit – větvení programu kde je potřeba, otevírání formulářů, generování zpráv a řízení databáze.

Podprogram je metoda projektu která může být přiřazena jako "sluha". Plní ty akce které po nich chtějí jiné metody. Funkce je podprogram, který vrací hodnotu do metody kterou je volán.

Metoda procesu je metoda projektu, která je volána při spuštění procesu. Proces funguje pouze tak dlouho, dokud funguje tato metoda. Jestli chcete vědět více informací o procesech, přečtěte si část Procesy. Všimněte si, že metoda nabídky, která spustí nový proces zatržením políčka **Začít nový proces** v položce nabídky, je také metodou procesu pro tento nový proces.

Metoda zachycující události je spuštěna ve zvláštním procesu pomocí ON EVENT CALLjako metoda procesu pro zachycování událostí. Obvykle necháte 4D dělat najednou více událostí. Například během vstupu dat 4D kontroluje úhozy na klávesy a kliknutí myší, pak může zavolat správnou metodu objektu nebo formuláře ve kterých budete mít nastaveno odchytávání událostí. V jiných případech můžete události ovládat přímo. Například pokud provádíte dlouhou operaci (jako For...End for pro prozkoumávání záznamů), můžete chtít zrušit tuto akci klepnutím na Ctrl - Tečka (Windows) nebo Cmd - Tečka (Macintosh). V tomto případě můžete použít metodu pro zachytávání událostí. Jestli chcete vědět více informací, přečtěte si popis k příkazu ON EVENT CALL.

Metody zachycující chyby jsou metody projektu založené na přerušení. Pokaždé když se objeví nějaká výjimka nebo chyba, spustí se tato metoda v procesu ve kterém byla instalovaná. Jestli chcete vědět více informací, přečtěte si popis k příkazu ON ERR CALL.

Metody nabídky

Metody nabídky se provádějí v prostředí Vlastních nabídek při vybrání položky nabídky ke které jsou přiřazeny. Metody se přiřazují k nabídkám v Editoru metod. Nabídka se spustí pokud je vybrán příkaz nabídky. Toto je jeden ze základů vytváření vlastní aplikace. Vytvořením vlastních nabídek a přiřazení metod k jednotlivým položkám si vytvoříte vlastní aplikaci. Jestli chcete vědět více informací o vytváření nabídek, tak si přečtěte *Příručku návrháře 4th Dimension.*

Příkazy vlastních nabídek mohou plnit více úkolů najednou. Například položka nabídky pro vkládání dat může vyvolat metodu, která provede dva úkoly: zobrazí platný vstupní formulář a zavolá příkaz ADD RECORD dokud uživatel neklepne na Zrušit.

Automatické provádění akcí je velmi silná možnost programnovacího jazyka. S použitím vlastních nabídek můžete

zautomatizovat spoustu akcí které by jste museli dělat ručně v prostředí uživatele. S vlastními nabídkami umožníte uživateli lepší vedení databáze.

Podprogramy

Jakmile vytvoříte metodu projektu, stane se částí jazyka pro databázi ve které je vytvořena. Metodu projektu můžete volat stejně jako kterýkoli vestavěný příkaz 4th Dimension. Metoda projektu použitá tímto způsobem se nazývá podprogram.

Podprogramy můžete použít k tomuto:

• Omezit opakování kódu

• Zkrátit vaše metody

• Urychlit změny ve vašich metodách

• Vytvoření modulárního kódu

Například řekněme, že máte databázi zákazníků. Jak databázi upravujete, příjdete na to, že tam je spousta částí, které se opakují, jako je vyhledávání zákazníků a upravování jejich záznamů. Kód může vypadat takto:

```
    ` Najít zákazníka
QUERY BY EXAMPLE([Zákazníci])
    ` Vybrat vstupní formulář
INPUT FORM([Zákazníci];"Vstup")
    ` Upravit záznam zákazníka
 MODIFY RECORD([Zákazníci])
```

Pokud nepoužijete podprogramy, budete muset tuto psát znovu pokaždé, když budete chtít upravit záznam zákazníka. Pokud máte v databázi deset míst, kde toto použijete, budete to muset napsat desetkrát. Ale pokud použijete podprogram, stačí vám tento kód napsat jenom jednou. Toto je první výhoda podprogramů - omezí velikost kódu.

Pokud se tato metoda nazývá UPRAVIT ZÁKAZNÍKA, můžete jí použít v jiné metodě pouhým napsáním jejího názvu do této jiné metody. Například k upravení a vytištění zákazníka stačí napsat toto:

```
UPRAVIT ZÁKAZNÍKA

PRINT SELECTION([Zákazníci])
```

Tato možnost velmi sníží složitost metod. V tomto případě nemusíte vědět jak metoda UPRAVIT ZÁKAZNÍKA vypadá, ale jenom co dělá. Toto je druhý důvod proč použít podprogramy - zkrácení metod. Tímto způsobem metody rozšiřují jazyk 4th Dimension.

Pokud potřebujete v tomto příkladu upravit metodu vyhledávání zákazníka, stačí upravit jednu metodu a ne deset. Toto je další důvod pro podprogramy - urychlit změny ve vaší databázi.

S použitím podprogramů se stane váš kód modulární. To znamená rozdělení kódu do modulů (podprogramů) a každý z nich splní logický úkol. Uvažte následující kód pro zkoušení účtů databáze:

```
NAJÍT PRÁZDNÉ ÚČTY ` Najít prázdné účty
NASTAVIT ÚČTY ` Nastavit účty
TISK ZPRÁVY ` Vytisknout zprávu
```

I pro někoho kdo nezná tuto databázi, je jednoduché poznat co tento kód dělá. Není nutné znát všechny

podprogramy, protože ty mohou mít mnoho řádků a mohou provádět komplexní úlohy. Zde je důležité pouze jaký provádějí úkol.

Můžeme vám pouze doporučit, aby jste vytvářeli váš kód po logických krocích nebo v "modulech" kdykoli to bude možné.

Vkládání parametrů do metod

Občas se vám může stát, že potřebujete předat metodě pro její provádění nějaká data a to pro každé volání jiná. To jde provést snadno s parametry.

Parametry (argumenty) jsou části dat, které potřebuje metoda v určitém pořadí pro svůj chod. Termíny parametr a argument budou mnohokrát použity v tomto manuálu záměnně. Parametry se používají i u vestavěných příkazů 4th Dimension. V tomto příkladu je použit řetězec "Ahoj" jako argument příkazu ALERT:

```
ALERT ("Ahoj")
```

K metodám se parametry přiřazují stejným způsobem. Pokud například metoda s názvem UDĚLAT_NĚCO přijímá tři parametry, bude volání k metodě vypadat třeba následovně:

```
UDĚLAT_NĚCO (STím;ATamtím;TímtoZpůsobem)
```

Jednotlivé parametry jsou odděleny středníkem (;).

V podprogramech jsou parametry kopírovány do postupných místních proměnných $1, $2, $3 atd. Číslo místní proměnné je pořadí v jakém jsou parametry do metody zadány.

Místní proměnné/parametry nejsou platná pole, proměnné nebo výrazy předané do metody při **volání metody**; pouze obsahují hodnotu (kopii původní hodnoty), která byla do metody předána.

V podprogramu můžete použít parametry $1, $2 ... stejně jako kterékoli jiné místní proměnné.

Vzhledem k tomu, že to jsou místní proměnné, jsou použity pouze v tomto jednom podprogramu a po jeho skončení jsou vymazány. Z tohoto důvodu nemohou podprogramy změnit hodnotu platného pole nebo proměnné přiřazené jako parametry při volání metody. Například:

```
   ` Zde je kód z metody MOJE METODA
   ` ...
UDĚLAT NĚCO ([Lidé]příjmení) ` Řekněme, že hodnota [Lidé]Příjmení je "kovář"
ALERT([Lidé]Příjmení)

   ` Zde je kód metody UDĚLAT NĚCO
$1:=Uppercase($1)
ALERT($1)
```

Výstražné okno zobrazené metodou UDĚLAT NĚCO zobrazí "KOVÁŘ" a výstraha z metody MOJE METODA zobrazí "kovář". Metoda místně upraví parametr $1, ale neupraví hodnotu v poli [Lidé]Příjmení.

Jsou dva způsoby, jak metodou UDĚLAT NĚCO změnit hodnotu v původním poli:

1. Místo toho, aby jste do metody předali pole , předáte do ní ukazatel a to takto:

```
   ` Zde je kód z metody MOJE METODA
   0   ` ...
1  UDĚLAT NĚCO (→[Lidé]příjmení) ` Řekněme, že [Lidé]Příjmení je "williams"
```

```
2  ALERT([Lidé]Příjmení)

   3    ` Zde je kód metody UDĚLAT NĚCO
4  $1→:=Uppercase($1→)
5  ALERT($1→)
```

Zde parametrem není pole, ale ukazatel na něj. Nyní v metodě UDĚLAT NĚCO již $1 není hodnota pole ale ukazatel na němu. Objekt ke kterému se odkazuje $1 ($1→ v kódu nahoře) je aktuální pole. Pokud změníte ukazatel na něj, je to jako když změníte pole. V tomto případě zobrazí obě okna výstrahu "KOVÁŘ".

0 Jestli chcete vědět více informací o Ukazatelích, přečtěte si část Ukazatele.

1. Místo toho aby jste nechali metodu UDĚLAT NĚCO jen "něco dělat", můžete jí přepsat tak, aby ještě vracela hodnotu. Můžete to udělat následovně:

```
   ` Zde je kód z metody MOJE METODA
   ` ...
   ` Řekněme, že [Lidé]Příjmení je "williams"
[Lidé]příjmení:=UDĚLAT NĚCO ([Lidé]příjmení)
ALERT([Lidé]Příjmení)

   ` Zde je kód metody UDĚLAT NĚCO
$1:=Uppercase($1)
$0:=$1
ALERT($0)
```

V tomto případě zobrazí opět obě okna výstrahu "KOVÁŘ". Tato technika kde podprogram vrací hodnotu se nazývá "použití funkce." Je popsána v dalších odstavcích.

Funkce: Metoda projektu která vrací hodnotu

Metody mohou vracet určité hodnoty. Metoda, která vrací hodnotu se nazývá **funkce**.

Příkazy 4D a zásuvných modulů (Plug-in) 4D, které vracejí hodnotu se také nazývají funkce.

Například následující řádek používá vestavěnou funkci 4th Dimension Lengthk navrácení délky řetězce. Řádek dosadí délku řetězce do proměnné MyLength. Zde je tvrzení:

```
MyLength:= Length ("Jak jsem se sem dostal?")
```

Jakýkoli podprogram může vracet hodnotu. Hodnota se dosadí do místní proměnné $0.

Například následující funkce, nazvaná Zvětšení4 vrátí řetězec s prvními čtyřmi písmeny velkými:

```
$0:=Uppercase(Substring($1; 1; 4))+Substring($1; 5)
```

Následující je příklad který použije funkci Zvětšení4:

```
NovyVýr:=Zvětšení4 ("Toto je dobré.")
```

V tomto příkladu dá proměnná hodnotu "TOTO je dobré."

Výsledek funkce $0 je místní proměnná v podprogramu. Může být použita pouze v podprogramu. Například v předchozím příkladu DĚLAT NĚCO byla nejdříve k $0 přiřazena hodnota $1 a pak použita jako parametr k příkazu

ALERT. V podprogramu můžete použít místní proměnnou $0 stejně jako kteroukoli jinou proměnnou. Je to 4D která vrací hodnotu $0 do volací metody.

Opakovaná (rekurzivní) metoda projektu

Metoda projektu se může volat i sama sebe. Například:

• Metoda A může volat metodu B která volá metodu A, tak bude metoda A volat znovu metodu B atd.

• Metoda může volat sama sebe

Toto se nazývá **opakování (rekurzivita)**. Jazyk 4d plně podporuje rekurzivitu.

Zde je příklad. řekněme, že máte tabulku [Přételé a Vztahy] s tímto jednoduchým seznamem polí:

- [Přátelé a Vztahy]Jméno

- [Přátelé a Vztahy]Jméno dítěte

Pro tento příklad jsou všechna pole jedinečná (žádné jméno není obsaženo dvakrát). Předáním prvního jména můžete sestavit větu "Můj přítel, Honza, který je potomek Paul , který je potomek Jane , který je potomek Robert, který je potomek Eleanor, odešel!":

Větu můžete sestavit tímto způsobem:

```
$vsName:=Request("Zdejte jméno:";"Honza")
If (OK=1)
   QUERY([Přátelé a Vztahy];[Přátelé a Vztahy]Jméno=$vsName)
   If (Records in selection([Přátelé a Vztahy])>0)
      $vtCeláVěta:="Můj přítel, "+$vsName
      Repeat
         QUERY([Přátelé a Vztahy];[Přátelé a Vztahy]Jméno dítěte=$vsName)
         $vlQueryResult:=Records in selection([Přátelé a Vztahy])
         If ($vlQueryResult>0)
            $ vtCeláVěta:=$ vtCeláVěta+", který je potomek " +[Přátelé a Vztahy]Jméno
            $vsName:=[Přátelé a Vztahy]Jméno
         End if
      Until ($vlQueryResult=0)
      $ vtCeláVěta:=$ vtCeláVěta +", odešel!"
      ALERT($ vtCeláVěta)
   End if
End if
```

2. Nebo jí můžete sestavit tímto způsobem:

```
$vsName:=Request("Zadejte jméno:";"Honza")
If (OK=1)
   QUERY([Přátelé a Vztahy];[Přátelé a Vztahy]Jméno=$vsName)
   If (Records in selection([Přátelé a Vztahy])>0)
      ALERT("Můj přítel, "+Genealogie($vsName)+", odešel!")
   End if
End if
```

s funkcí Genealogie zobrazenou zde:

```
    ` Genealogie metoda projektu
    ` Genealogie ( Řetězec ) → Text
    ` Genealogie ( Jméno ) → část věty
$0:=$1
QUERY([Přátelé a Vztahy];[Přátelé a Vztahy]Jméno dítěte=$1)
If (Records in selection([Přátelé a Vztahy])>0)
    $0:=$0+", který je potomkem "+Genealogie ([Přátelé a Vztahy]Jméno)
End if
```

Všiměte si, že funkce Genealogie volá sama sebe.

První způsob se nazývá **iterační algoritmus** a druhý způsob se nazývá **rekurzivní algoritmus**.

Při vytváření kódů jako je v předchozím případu, je důležité si pamatovat,že můžete metodu vždy napsat buď iterační nebo rekurzivní. Většinou rekurzivní stručnější a přehlednější a lépe udržovatelný kód, ale není nutné ho používat.

Některé typické způsoby použití rekurzivity ve 4D jsou:

• Práce se záznamy v tabulkách, která jsou ve vzájemném vztahu, obdobně, jako v předchozím příkladu,

• Prohlížení dokumentů a složek na vašem disku s použitím příkazu FOLDER LIST a DOCUMENT LIST. Složky mohou obsahovat složky a dokumenty a podsložky mohou obsahovat další složky a dokumenty atd.

Důležité: Rekurzivní volání musí mít někde konec. V předchozím příkladu skončí metoda Genealogie pokud hledání nevrátí žádný záznam. Bez tohoto testu by mohla metoda běhat stále dokola až by jí 4D zastavila sama, protože by neměla paměť na ukládání místních proměnných a parametrů.

Dále si přečtěte

Řízení průběhu, Metody databáze, Metody.

Příkazy a odkazy pro Definice jazyka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Application type (Typ aplikace)

Příkazy a odkazy pro Prostředí 4D

Verze 6.0

Application type → Long Integer

Parametr	Typ	Popis
Tento příkaz nepotřebuje žádné parametry		

Výsledek funkce	Long Integer ←	Číselná hodnota určující typ aplikace

Popis

Příkaz Application type vrací číselnou hodnotu, která určuje aplikaci 4D, na které pracujete. 4D poskytuje následující konstanty:

Konstanta	***Typ***	***Hodnota***
4th Dimension	*Long Integer*	*0*
4D Engine	*Long Integer*	*1*
4D Runtime	*Long Integer*	*2*
4D Runtime Classic	*Long Integer*	*3*
4D Client	*Long Integer*	*4*
4D Server	*Long Integer*	*5*
4D First	*Long Integer*	*6*

Příklad

Občas můžete potřebovat otestovat, jestli pracujete na 4D Serveru. Můžete to udělat následovně:

```
If (Application type=4D Server)
   ` Provést určitou akci
End if
```

Dále si přečtěte

Application version, Version type.

Příkazy a odkazy pro Prostředí 4D

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Version type (Typ verze)

Příkazy a odkazy pro Prostředí 4D

Verze 6.0

Version type → Long Integer

Parametr	Typ		Popis
Tento příkaz nepotřebuje žádné parametry			
Výsledek funkce	Long Integer	←	0 → Plná verze 1 → Demo verze

Popis

Příkaz Version type vrací číselnou hodnotu, která označuje typ verze na které pracujete. 4D poskytuje následující předdefinované konstanty:

Konstanta	Typ	Hodnota
Plná verze	*Long Integer*	*0*
Demo verze	*Long Integer*	*1*

Příklad

4D obsahuje některé vlastnosti, které nejsou přístupné pokud používáte demo verzi. Můžete testovat demo pomocí následujícího kódu:

```
If (Version type=Full Version)
   ` Provést určitou akci
Else
   ALERT ("Tato možnost není přístupná v demo verzi!")
End if
```

Dále si přečtěte

Application type, Application version.

Příkazy a odkazy pro Prostředí 4D

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Application version (Verze aplikace)

Příkazy a odkazy pro Prostředí 4D

Verze 6.0

Application version (*) → Řetězec

Parametr	Typ		Popis
*	*	←	Dlouhé číslo verze pokud je možné jinak krátké číslo verze.
Výsledek funkce	*	→	Číslo verze převedené do řetězce

Popis

Příkaz Application version vrací číslo verze 4D na které pracujete.

Pokud nepřídáte volitelný znak *, vrací se 4 znaky dlouhý řetězec formátovaný následovně:

Znak	***Popis***
1-2	*Číslo verze*
3	*Číslo upravení*
4	*Číslo revize*

Příklad: Řetězec "0600" znamená verzi 6.0.0.

• Pokud zadáte volitelný parametr *, vrací se 8 znaků dlouhý řetězec formátovaný následovně:

Znak	***Popis***
1	*"F" značí konečnou verzi*
	"B" značí beta verzi
	Další znaky značí vnitřní verze ACI
2-3-4	*Vnitřní číslo kompilace ACI*
5-6	*Číslo verze*
7	*Číslo upravení*
8	*Číslo revize*

Příklad: Řetězec "B0120602" znamená Beta verzi 12 verze 6.0.2.

Příklady

1. Tento příklad zobrazí číslo verze aplikace 4D:

```
$vs4Dversion:=Application version
ALERT("Používáte verzi "+String(Num(Substring($vs4Dversion;1;2))) +"."+$vs4Dversion[[3]]+"."+
$vs4Dversion[[4]])
```

2. Tento příklad testuje, jestli používáte konečnou verzi:

```
If(Substring(Application version(*);1;1)#"F")
   ALERT("Ujistěte se, že používáte konečnou verzi produktu 4D pro tuto databázi!")
   QUIT 4D
End if
```

Dále si přečtěte

Application type, Version type.

Příkazy a odkazy pro Prostředí 4D

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Compiled application (Kompilovaná aplikace)

Příkazy a odkazy pro Prostředí 4D

Verze 6.0

Compiled application → Logické

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádný parametr			
Výsledek funkce	Logické	←	Kompilované (pravda), Nekompilované (nepravda)

Popis

Příkaz Compiled application testuje jestli používáte kompilovanou nebo nekompilovanou verzi.

Příklad

V některé vaší metodě můžete mít obsažené ladění, které funguje pouze v nezkompilované databázi. Můžete použít následující příklad k testování:

```
` ...
If(Not(Compiled application))
   ` Sem vložte kód pro ladění
End if
   ` ...
```

Dále si přečtěte

IDLE, Undefined.

Příkazy a odkazy pro Prostředí 4D

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

PLATFORM PROPERTIES (VLASTNOSTI PLATFORMY)

Příkazy a odkazy pro Prostředí 4D

Verze 3

PLATFORM PROPERTIES (platforma{; Systém{; Stroj}})

Parametr	Typ		Popis
platforma	Číslo	←	1 - 68K Macintosh 2 - Power Macintosh 3 - Windows
systém	Číslo	←	Záleží na verzi, kterou používáte
stroj	Číslo	←	Záleží na verzi, kterou používáte

Popis

Příkaz PLATFORM PROPERTIES vrací informace o typu platformy, kterou používáte, verzi operačního systému a procesoru instalovaného ve vašem počítači.

Příkaz PLATFORM PROPERTIES vrací informace o prostředí v parametrech platforma, systém a stroj.

Platforma může být 68K nebo PowerPC Macintosh nebo Windows. Tento parametr vrací jednu z následujících předdefinovaných konstant:

Konstanta	*Typ*	*Hodnota*
Macintosh 68K	*Long Integer*	*1*
Power Macintosh	*Long integer*	*2*
Windows	*Long Integer*	*3*

Informace o systému a počítači záleží na verzi 4D kterou používáte.

Macintosh (68K a PowerPC)

Pokud používáte verzi 4th Dimension pro MacOS, budou informace o systému a počítači následující:

• Parametr systém vrací 32 bitovou hodnotu, pro kterou vysoká úroveň slov je nepoužitá a nízká úroveň slov je struktura jako tato:

- Vysoký bajt obsahuje hlavní číslo verze,

- Nízký bajt je složen ze dvou částí (4 bity každý). První část je hlavní číslo verze a druhá část je vedlejší číslo verze. Příklad: Systém 7.5.1 je kódován jako $0751, tak, že dostanete desítkovou hodnotu 1873.

Poznámka: Z 4D můžete tyto hodnoty získat použitím % (modulo) nebo číselných operátorů dělění nebo operátorů obsažených ve verzi 6.

• Parametr stroj vrací hodnotu jedinečného ID udávající model Macintoshe.

Poznámka: Tato čísla jsou publikována ve vývojářské a technické dokumentaci vydávané Apple Computer, Inc.

Verze Windows

Pokud používáte 4D verzi pro Windows, budou parametry systému a počítače vracet následující informace:

• Parametr systém vrací 32 bitovou hodnotu, kde jsou bity a bajty rovnány následovně:

Pokud je hodnota 1, znamená to, že používáte jednu z verzí Windows NT.

Pokud je hodnota 0, znamená to, že používáte Windows 3.1 nebo Windows 95.

Pokud první vrácené číslo bude 3, používáte verzi 3.x Windows nebo Windows NT. Pokud je vrácené číslo 4, používáte Windows 95 nebo Windows NT 4. V obou případech vám číslo řekne, kdy používáte Windows NT.

Další číslo je následující číslo verze Windows.

Poznámka: Z 4D můžete tyto hodnoty získat použitím % nebo číselných operátorů nebo operátorů obsažených ve verzi 6.

• Informace o počítači bude vracet jednu z následujících předdefinovaných konstant.

Konstanta	***Typ***	***Hodnota***
INTEL 386	*Long Integer*	*386*
INTEL 486	*Long Integer*	*486*
Pentium	*Long Integer*	*586*
PowerPC 601	*Long Integer*	*601*
PowerPC 603	*Long Integer*	*603*
PowerPC 604	*Long Integer*	*604*

Poznámka: Na Windows 3.1.x se bude vracet hodnota 486 pouze pokud je počítač vybaven procesorem Pentium.

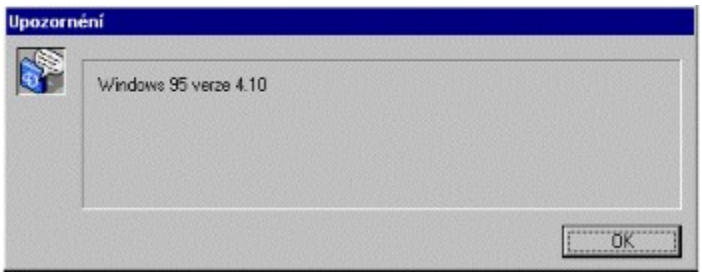
Příklad

Následující metoda projektu zobrazí který OS používáte:

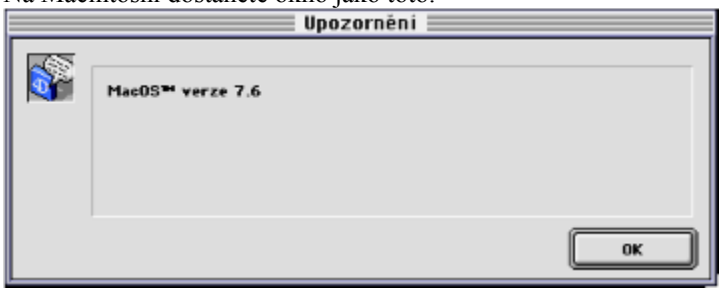
```
` Ukázat OS VERZI metoda projektu
PLATFORM PROPERTIES($vlPlatform;$vlSystem;$vlMachine)
If (($vlPlatform<1) | (3<$vlPlatform))
   $vsPlatformOS:=""
Else
   If ($vlPlatform=3)
      $vsPlatformOS:=""
      If ($vlSystem<0)
         $winMajVers:=((2^31)+$vlSystem)%256
         $winMinVers:=(((2^31)+$vlSystem)56)%256
         If ($winMajVers>=4)
            $vsPlatformOS:="Windows™ 95"
         Else
            $vsPlatformOS:="Windows™ (with Win32s)"
         End if
      Else
         $winMajVers:=$vlSystem%256
         $winMinVers:=($vlSystem56)%256
         $vsPlatformOS:="Windows™ NT"
      End if
   $vsPlatformOS:=$vsPlatformOS+" verze "+String($winMajVers)
                  + "."+String($winMinVers)
   Else
      $vsPlatformOS:="MacOS™ verze "+String($vlSystem56)
                     +"."+String(($vlSystem6)%16)
                     +(("."+String($vlSystem%16))*Num(($vlSystem%16) # 0))
   End if
```

```
End if
ALERT($vsPlatformOS)
```

Na Windows dostanete hlášení jako toto:



Na Macintoshi dostanete okno jako toto:



Dále si přečtěte

Bitwise Operátory.

Příkazy a odkazy pro Prostředí 4D

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Application file (Soubor aplikace)

Příkazy a odkazy pro Prostředí 4D

Verze 6.0

Application file → Řetězec

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry			
Výsledek funkce	Řetězec	←	Název spustitelného souboru nebo aplikace 4D

Popis

Příkaz Application file vrací dlouhý název spustitelného souboru nebo aplikace 4D.

Na Windows

Pokud například používáte 4D umístěnou ve složce DWINna disku E, vrátí vám příkaz: E:DWIND.EXE.

Na Macintoshi

Pokud používáte 4th Dimension umístěnou vce složce 4th Dimension® 6.0 na disku Macintosh HD, vrátí vám příkaz Macintosh HD:4th Dimension® 6.0:4th Dimension.

Příklad

Při zapnutí Windows potřebujete ověřit jestli je soubor .DLL na stejné úrovni jako soubor 4D. Do metody Při spuštění můžete umístit následující kód:

```
If (Na windows  & (Application type#4D Server))
   If (Test path name (Dlouhý název cesty(Application file)+"XRAYCAPT.DLL")#Is a document)
      ` Zobrazí dialogové okno ukazující, že knihovna XRAYCAPT.DLL
      ` chybí. Proto, X-rays vlastnosti nebudou fungovat.
   End if
End if
```

Poznámka: Metoda projektu Na windows a Long name to path jsou zobrazeny v části Systémové dokumenty.

Dále si přečtěte

Data file, DATA SEGMENT LIST, Structure file.

Příkazy a odkazy pro Prostředí 4D

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Structure file (Soubor struktury)

Příkazy a odkazy pro Prostředí 4D

Verze 6.0

Structure file → Řetězec

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry			
Výsledek funkce	Řetězec	←	Název a cesta k souboru struktury

Popis

Příkaz Structure file vrací název a cestu k souboru struktury na které právě pracujete.

Na Windows

Pokud například pracujete na databázi nazvané Kontakty umístěné ve složce na disku G, vrátí vám příkaz G:.4DB.

Na Macintoshi

Pokud například pracujete na databázi Kontakty umístěné ve složce Dokumenty:Kontakty na disku Macintosh HD, vrátí vám příkaz Macintosh HD:Dokumenty:Kontakty:Kontakty.

UPOZORNĚNÍ: Pokud pracujete na 4D Client je zobrazen pouze název struktury a ne celá cesta.

Příklad

Tato metoda zobrazí cestu a název ke struktuře kterou používáte:

```
If (Application type#4D Client)
   $vsStructureFilename:=Dlouhý název dokumentu (Structure file)
   $vsStructurePathname:=Dlouhý název cesty(Structure file)
   ALERT("Používáte databázi "+Char(34)+$vsStructureFilename
      +Char(34)+" umístěnou "+Char(34)+$vsStructurePathname+Char(34)+".")
Else
   ALERT("Jste připojeni k databázi "+Char(34)+Structure file+Char(34))
End if
```

Poznámka: Metody projektu Dlouhý název dokumentu a Dlouhý název cesty čtěte v části Systémové dokumenty.

Dále si přečtěte

Application file, Data file, DATA SEGMENT LIST.

4th Dimension 6.5, Seznam témat konstant

Data file (Datový soubor)

Příkazy a odkazy pro Prostředí 4D

Verze 6.0

Data file ((část)) → Řetězec

Parametr	Typ		Popis
část	číslo	→	číslo části
výsledek funkce	Řetězec	←	Název datového souboru databáze

Popis

Příkaz Data file vrací název datového souboru nebo části dat se kterou nyní pracujete.

Pokud nezadáte číslo části, vrátí vám příkaz název datového souboru nebo jeho první části (pokud je datový soubor rozdělen). Pokud zadáte číslo části, vrátí vám příkaz umístění a název patřičné části. Pokud zadáte číslo části větši, než existuje, vrátí se vám prázdný řetězec.

Na Windows

Pokud například používáte databázi Kontakty umístěnou ve složce DOCna disku G, příkaz Data file vrátí řetězec G:.4DD (za předpokladu, že přijmete navržený název datového souboru).

Na Macintoshi

Pokud používáte databázi umístěnou ve složce DOC:Kontakty na disku Macintosh HD, vrátí příkaz Data file Macintosh HD:DOC:Kontakty:Kontakty.data (za předpokladu, že přijmete navržený název datového souboru).

UPOZORNĚNÍ: Pokud použijete tento příkaz při práci na 4D Client, vrátí se pouze název datového soubotu nebo jeho první části a ne celá cesta k němu. To znamená, že příkaz bude pro ostatní datové části vracet prázdný řetězec. Pokud chcete z klienta zjistit seznam částí dat, musíte tento seznam vytvořit a uložit do proměnné na serveru a pak pomocí příkazu GET PROCESS VARIABLE vzít hodnotu této proměnné.

Příklad

Následující příklad jde přes datové části databáze:

```
If (Application type#4D Client)
   $vlDataSegNum:=0
   Repeat
      $vlDataSegNum:=$vlDataSegNum+1
      $vsDataSegName:=Data file($vlDataSegNum)
      If ($vsDataSegName#"")
         ALERT ("Datová část "+String($vlDataSegNum)+":"+Char(34)
            + $vsDataSegName+Char(34)+".")
      End if
   Until ($vsDataSegName="")
   ALERT("Existuje "+String($vlDataSegNum-1)+" datových částí.")
End if
```

Dále si přečtěte

Application file, DATA SEGMENT LIST, Structure file.

Příkazy a odkazy pro Prostředí 4D

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ACI folder (Složka ACI)

Příkazy a odkazy pro Prostředí 4D

Verze 6.0

ACI folder → Řetězec

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry			
Výsledek funkce	Řetězec	←	Cesta ke složce ACI

Popis

Příkaz ACI folder vrací cestu ke složce ACI umístěné v aktivní složce systému.

Na Windows

Složka ACI je nazvána ACI a je umístěna v aktivní složce WINDOWS (obvykle C:). To je důvod, proč tento příkaz většinou vrátí řetězec C:\WINDOWS. Na PC počítače mohou být nastaveny jinak a umístění složky WINDOWS může být upraveno během instalace. Pokud tedy potřebujete uložit některé soubory do složky ACI, tento příkaz vám ukáže přímou cestu k platné složce.

Na Macintoshi

Složka ACI je nazvána ACI a je umístěna ve složce Předvolby v aktivní složce systému. Většinou je cesta vrácená příkazem ACI folder Macintosh HD:Sytém:Předvolby:ACI. Vzhledem k tomu, že uživatel si může přezvat disk a systémovou složku, může se cesta změnit. Proto, pokud chcete uložit některé soubory do složky ACI, tento příkaz vám dá aktuální cestu k této složce.

Nezávislost na platformě a systému: S použitím příkazu ACI folder můžete cestu ke složce ACI získat na kterékoli platformě a systému.

Prostředí 4D využívá složku ACI k uložení následujících informací:

• Soubory registrace uživatele

• Soubory předvoleb prostředí 4D aplikací a nástrojů

• 4D Client nebo Internetkomponenty (na Windows složka ....ACIK) a jejich soubory

• .rex a res soubory vytvořené 4D Client pro ukládání zdrojů stažených ze 4D Server

• Místní databázové složky vytvořené 4D Client pro ukládání 4D Extension stažených ze 4D Server

UPOZORNĚNÍ: Do složky ACI můžete uložit jakékoli soubory u kterých není dobré s nimi pohybovat a jsou důležité pro chod programu.

Příklad

Během zapnutí jednouživatelské databáze můžete chtít načíst (nebo vytvořit) vaše vlastní nastavení v souborech uložených ve složce ACI. Aby jste to provedli můžete vložit do metody databáze Při spuštění podobný kód tomuto:

```
MAP FILE TYPES("PREF";"PRF";"Preferences file")
   ` Mapovat PREF MacOS typ souboru do .PRF Windows přípony souboru
$vsPrefNazev:=ACI folder+"MyPrefs" ` Build pathname to the Preferences file
   ` Textovat jestli soubor existuje
If (Test path name ($vsPrefNazev+(".PRF"*Num(Na windows )))#Is a document)
   $vtPrefDocRef:=Create document($vsPrefNazev;"PREF") ` Pokud není vytvořit
```

```
Else
   $vtPrefDocRef:=Open document($vsPrefNazev;"PREF") ` Pokud ano, otevřít
End if
If (OK=1)
   ` provést obsah dokumentu
   CLOSE DOCUMENT($vtPrefDocRef)
Else
   ` ovládat chyby
End if
```

Dále si přečtěte

System folder, Temporary folder, Test path name.

Příkazy a odkazy pro Prostředí 4D

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DATA SEGMENT LIST (SEZNAM ČÁSTÍ DAT)

Příkazy a odkazy pro Prostředí 4D

Verze 6.0

DATA SEGMENT LIST (Části)

Parametr	Typ	Popis
části	Řetězcový array	Název částí dat pro databázi

Popis

Příkaz DATA SEGMENT LIST vytvoří array jako seznam datových částí s jejich plným názvem pro databázi kterou právě používáte.

UPOZORNĚNÍ: Tento příkaz neprovede nic pokud je spuštěn na 4D Client. Pokud chcete sestavit seznam částí na stanici klienta, musíte použít Uloženou proceduru (Store Procedure) k vytvoření seznamu na serveru a jeho uložení do proměnné a pak si převzít tuto proměnnou pomocí příkazu GET PROCESS VARIABLE.

Příklady

1. Ve formuláři Informace o částech v tabulce [Dialogy] můžete chtít vytvořit rozevírací seznam zobrazující jednotlivé části dat. Aby jste to udělali, napište následující:

```
   ` Metoda formuláře [Dialogy];"Informace o částech"
Case of
   ÷ (Form event=On Load )
         ` ...
      ARRAY STRING(255;asDataSegName;0)
      DATA SEGMENT LIST(asDataSegName)
         ` ...
End case
```

2. Následující metoda vám řekne, jestli je databáze rozdělena na části:

```
` Je datový soubor rozdělen → Logické
C_BOOLEAN ($0)
DATA SEGMENT LIST($asDataSegName)
$0:=(Size of array($asDataSegName)>1)
```

3. Po použití příkazu ADD DATA SEGMENT můžete testovat, jestli uživatel přidal novou část:

```
DATA SEGMENT LIST($asBefore)
ADD DATA SEGMENT
DATA SEGMENT LIST($asAfter)
If(Size of array($asBefore)# Size of array($asAfter))
   ` Ano, jsou další části dat
Else
   ` Stejný počet datových částí
End if
```

Dále si přečtěte

Application file, Data file, Structure file.

Příkazy a odkazy pro Prostředí 4D

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ADD DATA SEGMENT (PŘIDAT ČÁST DAT)

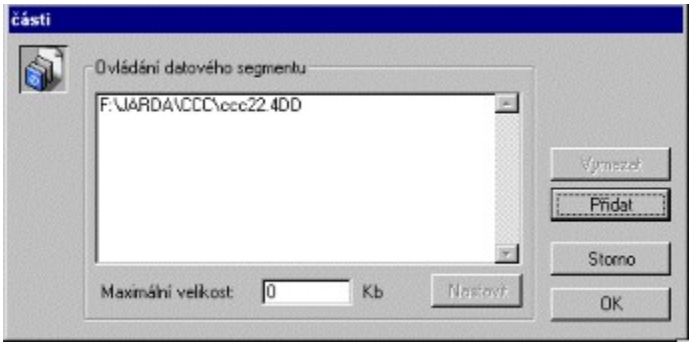
Verze 3

ADD DATA SEGMENT

Parametr	Typ	Popis
Tento příkaz nevyžaduje žádné parametry		

Popis

Příkaz ADD DATA SEGMENT zobrazí dialogové okno Části zobrazené zde:



Jakmile uživatel klepne na tlačítko OK k uložení, proměnná OK se nastaví na 1. Pokud uživatel klepne na Zrušit, nastaví se proměnná OK na 0.
Poznámka: Tento příkaz nedělá nic pokud je použit na 4D Server.
Jakmile jsou všechny části dat plné, zobrazí 4th Dimension nebo 4D Server chybu -9999. Zobrazí se dialogové okno a nahlásí, že je disk plný.
Pokud používáte 4th Dimension, můžete použít ON ERR CALL metodu k zachycení chyby. Můžete pak použít příkaz ADD DATA SEGMENT a umožnit tak uživateli přidat novou část dat.

Pokud používáte 4D Server, můžete zobrazit dialogové okno, že Správce dat musí přidat novou část dat.

Dále si přečtěte

ON ERR CALL.

Systémové proměnné a Sady

Systémová proměnná OK je nastavena na 1 pokud je okno Části používáno.

Příkazy a odkazy pro Prostředí 4D

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

FLUSH BUFFERS (ULOŽIT BUFFERY)

Verze 3

FLUSH BUFFERS

Parametr	Typ	Popis
Tento příkaz nevyžaduje žádné parametry		

Popis

Příkaz FLUSH BUFFERS okamžitě ukládá data z paměti na disk. Všechny změny které byly provedeny v databázy jsou uloženy na disk.

Obvykle nepotřebujete volat tento příkaz, pokud 4D ukládá data podle normálních pravidel. Předvolby databáze **Uložit data** (v Prostředí návrháře), definuje jak často ukládat data a 4D se pak stará o ukládání dat.

Příkazy a odkazy pro Prostředí 4D

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

QUIT 4D (UKONČIT 4D)

Verze 3

QUIT 4D

Parametr	Typ	Popis
Tento příkaz nevyžaduje žádné parametry		

Popis

Příkaz QUIT 4D vypne 4th Dimension a vrátí vás do systému. Po použití příkazu QUIT 4D se platný proces zastaví a pak 4D udělá následující:

• Pokud existuje metoda Při ukončení, 4D jí spustí v novém místní procesu. Například lze použít tuto metodu databáze k informování jiných procesů, přes meziprocesovou komunikaci, že tyto musí ukončit vstup dat nebo zastavit provádění operací započatých metodou databáze Při spuštění (připojení 4D k jiným serverům). To že je zaznamenáno, že 4D bude skončena znamená, že metoda databáze Při ukončení může provést všechny uzavírací operace které chcete, může počkat na ukončení těchto operací, ale rozhodně nemůže zrušit ukončení, které nastane po konci jejího provádění.

• Pokud neexistuje metoda databáze Při ukončení, 4D zruší každý běžící proces bez rozdílu.

Pokud uživatel zrovna zadává data, bude záznam zrušen bez uložení.

Aby jste uživateli umožnili uložit data v okně které má otevřené, můžete použít meziprocesovou komunikaci a dát signál všem procesům, že databáze bude skončena. Můžete to udělat dvěma způsoby:

• Provést to ještě před použitím příkazu QUIT 4D

• Nastavit toto upozornění do metody databáze Při ukončení.

Třetí způsob je také možný. Před voláním příkazu QUIT 4D, můžete ověřit, jestli otevřená okna potřebují potvrzení: pokud je to tento případ, můžete se uživatele zeptat, jestli chce tyto vstupy potvrdit nebo zrušit a pak jej nechat vybrat znovu Konec z nabídky. Z uživatelského hlediska jsou první dva způsoby výhodnější.

Příklad

Tato metoda projektu je přiřazena k položce Konec v nabídce Soubor.

```
   ` Metoda projektu M_SOUBOR_KONEC
CONFIRM("Jste si jistý že chcete vypnout program?")
If (OK=1)
   QUIT 4D
End if
```

Dále si přečtěte

Metoda databáze Při ukončení.

SELECT LOG FILE (VYBRAT LOG SOUBOR)

Příkazy a odkazy pro Prostředí 4D

Verze 3

SELECT LOG FILE (logSoubor | *)

Parametr	Typ	Popis
LogSoubor | *	Řetězec | *	Název LogSouboru nebo "*" pro uzavření platného log souboru

Popis

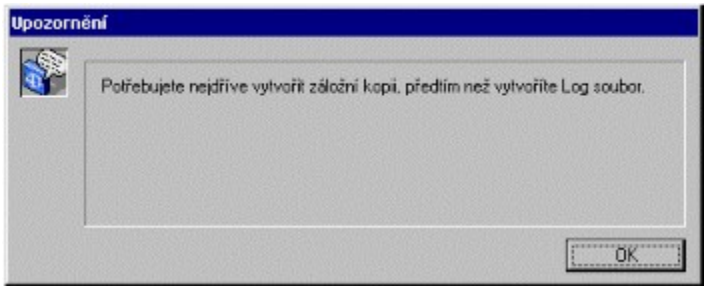
Příkaz SELECT LOG FILE otevře, vytvoří nebo uzavře Log soubor podle hodnoty, kterou do příkazu předáte parametrem logSoubor.

DŮLEŽITÉ: Volání příkazu SELECT LOG FILE je stejné jako vybrat položku Log soubor z nabídky Soubor v prostředí uživatele. Může být použit pouze v případě, že je k databázi instalován 4D Backup.

Pokud parametrm logSoubor předáte prázdný řetězec, zobrazí SELECT LOG FILE dialogové okno Otevřít soubor, které uživateli umožní otevřít Log soubor nebo vytvořit nový. Pokud uživatel klepne na Otevřít a soubor bude správně otevřen, nastaví se systémová proměnná OK na 1. Jinak pokud uživatel klepne na Zrušit nebo pokud Log soubor nemůže být otevřen nebo vytvořen, je nastavena OK na 0.

Pokud předáte parametr "*", SELECT LOG FILE uzavře log soubor přiřazený k databázi. OK je nastavena na 1 pokud je soubor správně uzavřen.

Pokud použijete SELECT LOG FILE k vytvoření nebo otevření log souboru, pokud je právě prováděna úplná záloha dat a datový soubor již obsahuje záznamy, 4th Dimension zobrazí následující upozornění:



4D pak vygeneruje chybu -4447, kterou můžete zachytit metodou ON ERR CALL.

Poznámka: Příkaz SELECT LOG FILE neudělá nic, pokud je použit na 4D Server. Jestli chcete vědět více informací o tomto příkazu, přečtěte si dokumentaci k *4D Backup*.

Dále si přečtěte

ON ERR CALL.

Systémové proměnné a Sady

OK je nastavena na 1 pokud je log soubor správně otevřen, vytvořen nebo uzavřen.

Ovládání chyb

Chyba -4447 je vygenerována pokud operace nemůže být provedena protože databáze musí být dozálohována. Chybu můžete zachytit metodou ON ERR CALL.

Příkazy a odkazy pro Prostředí 4D

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Array

Příkazy a odkazy pro Array

Verze 6.0

Array je řada proměnných stejného typu. Každá proměnná se nazývá **prvek** array. Velikost array je počet prvků, které obsahuje. Velikost array zadáváte při jeho vytvoření. Velikost můžete kdykoli změnit předáním nebo vymazáním některých prvků nebo upravení velikosti stejným příkazem, kterým jste array vytvořili.

Array vytváříte jedním z příkazů deklarace. Pro podrobnosti si přečtěte část Vytváření array.

Prvky jsou číslovány od **1 do N**, kde N je velikost array. Array má vždy **prvek nula**, s kterým můžete pracovat, jako se všechmi jinými prvky array, ale tento prvek nebude ukázán při zobrazení array ve formuláři. Třebaže prvek nula není ukázán při použití array v objektu formuláře, nejsou žádná omezení v jeho použití jazykem. Jestli chcete vědět více informací, přečtěte si část Použití provku nula v array.

Array jsou proměnné 4D. Jako každá proměnné, má array rozsah platnosti a dodržuje pravidla jazyka 4D pouze s malými rozdíly. Jestli chcete vědět více informací, přečtěte si část Array a jazyk 4D a Array a Ukazatele.

Array jsou objekty jazyka; můžete vytvářet a používat array, které se nikdy neobjeví na obrazovce. Array jsou objekty rozhraní uživatele. Jestli chcete vědět více informací o spolupráci mezi array a objekty formuláře, přečtěte si Array a objekty formuláře a Vytvoření skupiny posuvných oblastí.

Array jsou navrženy k uchování různých typů dat na krátkou dobu. Protože array jsou uchovávány v paměti, jsou snadno ovladatelné a dá se s nimi rychle manipulovat. Jestli chcete vědět více informací, přečtěte si Array a paměť.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Vytváření array

Příkazy a odkazy pro Array

Verze 6.0

Array vytváříte jedním z příkazů popsaném v této kapitole. Následující tabulka ukazuje tyto příkazy:

Příkaz	**Vytvoří nebo změní velikost array z:**
ARRAY INTEGER	2-bytové Integer hodnoty
ARRAY LONGINT	4-bytové Integer hodnoty
ARRAY REAL	Real hodnoty
ARRAY TEXT	Textové hodnoty (od 0 fo 32 000 znaků na prvek)
ARRAY STRING	Řetězcové hodnoty (od 0 do 255 znaků na prvek)
ARRAY DATE	Datumové hodnoty
ARRAY BOOLEAN	Logické hodnoty
ARRAY PICTURE	Obrázkové hodnoty
ARRAY POINTER	Ukazatele

Každý z těchto příkazů může vytvořit nebo změnit velikost jedno nebo dvojrozměrných array. Jestli chcete vědět více informací o o dvojrozměrných array přečtěte si Dvojrozměrné array.

Poznámka: Rozdíl mezi Textovým a Řetězcovým array leží v jejich prvcích. V obou druzích array mohou prvky uchovávat textové hodnoty (znaky). Nicméně:

• V textové array je každý prvek proměnné délky a samotné znaky uchovává v oddělené části paměti,

• V řetězovém array mají všechny prvky stejnou délku (délka zadaná při vytvoření array). Všechny prvky jsou postupně uloženy ve stejné části paměti, bez rozdílu obsahu.

Kvůli tomuto rozdílu jsou řetězcové array rychlejší než textové. Je to proto, že prvky řetězcové array jsou maximálně 255 znaků dlouhé.

Poznámka: Array integer umožňují rovněž ukládat čas v číselné formě (sekund). Zobrazení typu čas lze provést ve formuláři pomocí formátu zobrzazení &/x, kde x reprezentuje pořadí formátu času. Například &/4 je formát ve tvaru Hodiny:minuty.

Následující řádek kódu vytvoří Integer array s 10-ti položkami:

```
ARRAY INTEGER(aiAnArray;10)
```

Další kód upraví stejný array na 20 prvků:

```
ARRAY INTEGER(aiAnArray;20)
```

Pak tento kód upraví array na nula položek:

```
ARRAY INTEGER(aiAnArray;0)
```

Na prvky array se odkazujete použitím složených závorek ({...}). Číslo použité v závorkách slouží k adresování určitého prvku: toto číslo se nazývá **číslo prvku**. Následující řádky vloží pět jmen do array s názvem atPrijmeni a pak je zobrazí ve výstražném okně:

```
ARRAY TEXT (atPrijmeni;5)
atPrijmeni{1} := "Richard"
atPrijmeni{2} := "Sarah"
atPrijmeni{3} := "Sam"
atPrijmeni{4} := "Jane"
```

```
atPrijmeni{5} := "John"
For ($vlElem;1;5)
   ALERT ("Položka #"+String({$vlElem})+" je: "+atPrijmeni{$vlElem})
End for
```

Všiměte si zápisu atPrijmeni{$vlElem}. Lépe než nastavení číselné hodnoty jako atPrijmeni{3}, můžete použít číselnou proměnnou k určení který prvek array chcete použít.

Při použití opakovací struktury (For...End for, Repeat...Until nebo While ... End While, jednotlivé části kódu mohou adresovat všechny nebo část prvků array.

Array a další oblasti jazyka 4D

Existují ještě nějaké příkazy které mohou vytvořit a pracovat s array. Jestli chcete vědět více informací, přečtěte si popis následujících příkazů:

• K práci s array a výběrem záznamů použijte příkazy SELECTION RANGE TO ARRAY, SELECTION TO ARRAY, ARRAY TO SELECTION a DISTINCT VALUES.

• Můžete vytvářet grafy a diagramy ze sérií hodnot z tabulek, podtabulek a array. Jestli chcete vědět více informací, přečtěte si příkaz GRAPH.

• Verze 6 obsahuje nové příkazy pro práci s hierarchickými seznamy. Příkazy LIST TO ARRAY a ARRAY TO LIST (z předchozí verze) byly ponechány pro kompatibilitu.

• Nové příkazy ve verzi 6 vytvoří array jedním voláním. Tyto příkazy jsou FONT LIST, WINDOW LIST, VOLUME LIST, FOLDER LIST, a DOCUMENT LIST.

Dále si přečtěte

ARRAY BOOLEAN, ARRAY DATE, ARRAY INTEGER, ARRAY LONGINT, ARRAY PICTURE, ARRAY POINTER, ARRAY REAL, ARRAY STRING, ARRAY TEXT, Array, Dvojrozměrné Array.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Array a objekty formuláře

Příkazy a odkazy pro Array

Verze 6.0

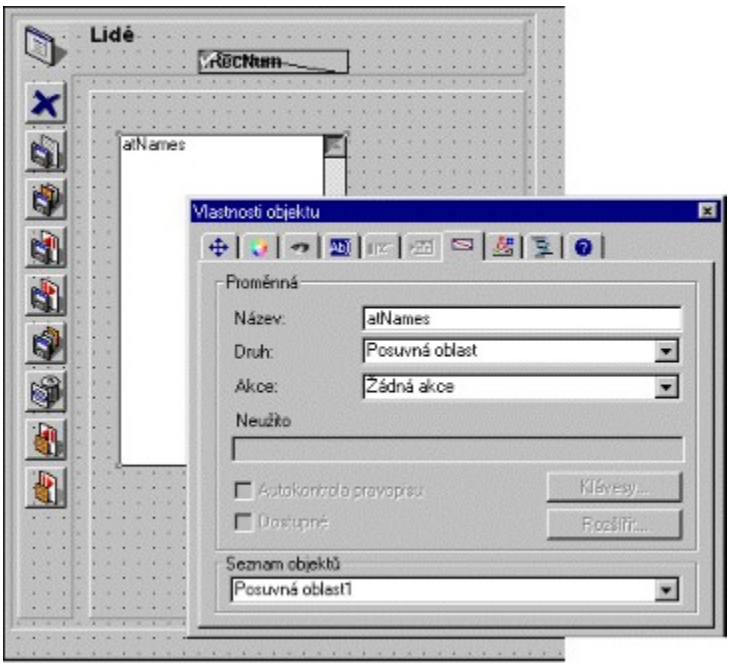
Array jsou objekty jazyka - můžete vytvářet array, které se nikdy neobjeví na obrazovce. Rovněž jsou ale array i objekty rozhraní uživatele. Následující typy **objektů formuláře** jsou podporovány array:

- Rozevírací nabídky
- Rozevírací seznamy
- Text se seznamem
- Posuvné oblasti
- Ovládací karty

Tyto objekty můžete předdefinovat v prostředí návrháře pomocí Editoru formulářů (s použitím tlačítka Výchozí hodnoty v okně Vlastnosti objektu) a můžete je rovněž definovat i programově pomocí příkazů array. V obou případech je objekt formuláře podporován array které jste vytvořili vy nebo 4D.

Při používání těchto objektů, můžete určit, který prvek byl vybrán v array uživatelem pomocí myši, testováním **Vybrného prveku pomocí jazyka**. Můžete také vybrat určitou položku v objektu jazykem 4D nastavením vybraného prvku pro array.

Pokud je array použit k podpoře objektu formuláře, má dvě podstaty: je to jak objekt jazyka tak objekt uživatelského rozhraní. Pokud například při vytváření formuláře vytvoříte posuvnou oblast: na straně **Proměnná** v okně **Vlastnosti objektu** pojmenujte proměnnou objektu:



Název, v tomto případě atPrijmeni, je název array, které používáte pro vytvoření a udržování prvků posuvné oblasti.
Poznámka: Nemůžete ve formuláři zobrazit dvojrozměrné array a array ukazatelů.

Příklad: Vytvoření rozevíracího seznamu

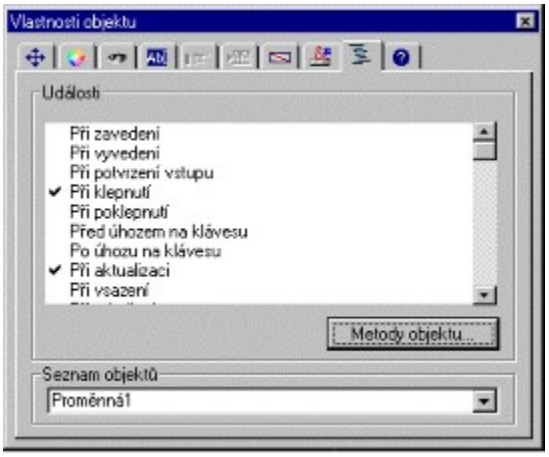
Následující příklad ukazuje jak vyplnit array a zobrazit jej v rozevíracím seznamu. Array arSalaries je vytvořen příkazem ARRAY REAL. Obsahuje všechny standardní platy placené zaměstnancům společnosti. Jakmile uživatel vybere položku z rozevíracího seznamu, je k poli [Zaměstnanci]Plat přiřazena hodnota vybraná v prostředí uživatele nebo Vlastních nabídek.

Vytvoření rozevíracího seznamu arSalaries ve formuláři

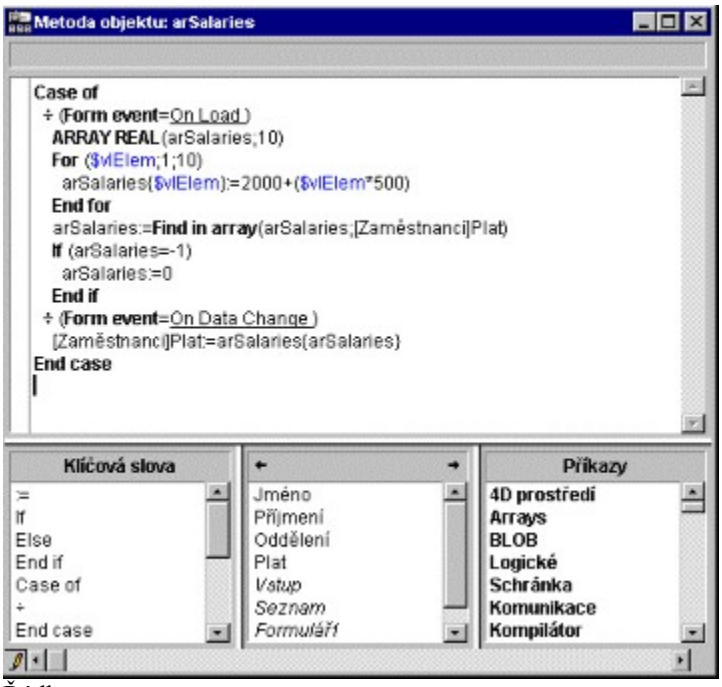
Vytvořte rozevírací seznam a pojmenujte jej arSalaries. Název rozevíracího seznamu může být stejný jako název array.

Naplnění array

Naplňte array arSalaries s použitím události Při zavedení pro objekt. Nezapomeňte aktivovat tuto událost v okně **Vlastnosti objektu** jak je ukázáno:



Klepněte na tlačítko **Metoda objektu** a vytvořte metodu jak je ukázáno:



Řádky:

ARRAY REAL(arSalaries;10)
For($vlElem;1;10)
arSalaries{$vlElem}:=2000+($vlElem*500)
End for

vytvoří číselný array 2500, 3000,... 7000, souhlasící s celkovým platem 30000 až 84000.

Řádky:

arSalaries:=**Find in array**(arSalaries;[Zaměstnanci]Plat)
If (arSalaries=-1)
arSalaries:=0
End if

obstarává jak vytvoření nového záznamu tak i upravení již existujícího záznamu.

• Jestliže vytvoříte nový záznam, pole [Zaměstnanci]Plat je rovno nula. V tomto případě nenajde Find in array hodnotu v array a vrátí -1. Test If (arSalaries=-1) nastaví arSalaries na nula, hodnotu označující že žádný prvek v rozevíracím seznamu není vybrán.

• Pokud upravujete existující záznam, Find in array nalezne hodnotu v array a nastaví vybraný prvek do rozevíracího seznamu jako výchozí hodnotu pole. Pokud hodnota pro zaměstnance není v array, test If(arSalaries=-1) odvybere každý prvek v seznamu.

Poznámka: Jestli chcete vědět více informací o vybraných prvcích array, přečtěte si následující část.

Přenesení vybrané hodnoty do pole [Zaměstnanci]Plat

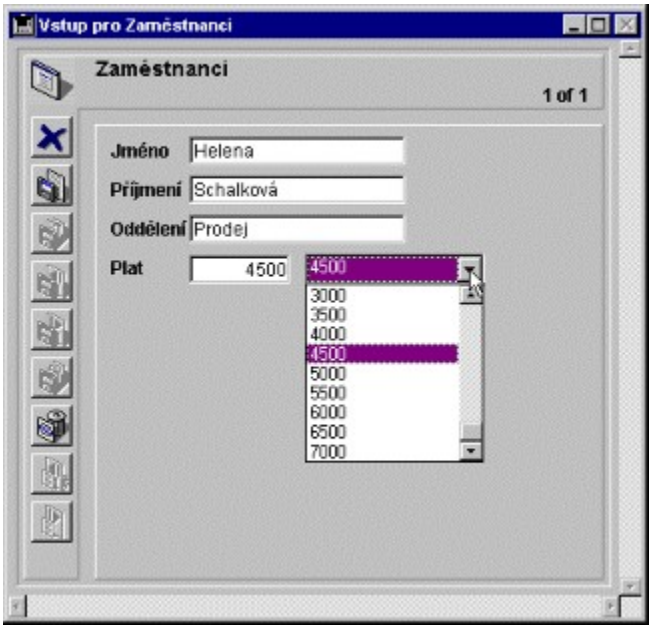
K přiřazení hodnoty vybrané z rozevíracího seznamu arSalaries potřebujete pouze přiřadit k objektu události Při klepnutí nebo Při aktualizaci. Číslo prvku vybraného prvku je hodnota array asSalaries. Proto výraz arSalaries(arSalaries) vrací hodnotu vybranou z rozevíracího seznamu.

Dokončete metodu objektu pro arSalaries následovně:

Case of
÷ (**Form event**=On Load)
ARRAY REAL(arSalaries;10)

```
        For($vlElem;1;10)
            arSalaries{$vlElem}:=2000+($vlElem*500)
        End for
        arSalaries:=Find in array(arSalaries;[Zaměstnanci]Plat)
        If (arSalaries=-1)
            arSalaries:=0
        End if
    ÷ (Form event=On Data Change)
        [Zaměstnanci]Plat:=arSalaries{arSalaries}
End case
```

V prostředí uživatele nebo vlastní nabídky bude rozevírací seznam vypadat následovně:



Následující část popisuje obecné a základní operace které budete provádět s array při jejich použití jako objekty formuláře.

Zadání velikosti array

Současnou velikost array můžete zjistit použitím příkazu Size of array. S použitím předchozího příkladu zobrazí následující řádek 5:

```
ALERT("Velikost array atPrijmeni je: "+String(Size of array(atPrijmeni)))
```

Změna pořadí prvků array

S použitím příkazu SORT ARRAY můžete přetřídit položky array. Použijeme předchozí příklad a zobrazíme array do posuvné oblasti:

a. Inicializace a oblast bude vypadat jako seznam nalevo.

b. Po spuštění následujícího řádku:

SORT ARRAY(atPrijmeni;>)

bude oblast vypadat jako seznam uprostřed.

c. Po spuštění následujícího řádku:

SORT ARRAY(atPrijmeni;<)

bude oblast vypadat jako seznam napravo.

Přidání a vymazání prvku

Prvky array můžete přidávat, vkládat nebo mazat s použitím příkazů INSERT ELEMENT a DELETE ELEMENT.

Obsluha kliknutí v array: testování vybraných prvků

S použitím předchozího příkladu a za předpokladu zobrazení array jako posuvné oblasti, můžete ovládat kliknutí do oblasti následovně:

```
   ` metoda objektu atPrijmeni - posuvná oblast
Case of
   ÷ (Form event=On Load)
      ` Nastavení array (bylo ukázáno výše)
      ARRAY TEXT (atPrijmeni;5)
      ` ...
   ÷ (Form event=On Unload)
      ` Již nepotřebujeme array
      CLEAR VARIABLE(atPrijmeni)
   ÷ (Form event=On Clicked)
      If (atPrijmeni#0)
         vtInfo:="Klepnul(a) jste na: "+atPrijmeni{atPrijmeni}
      End if
   ÷ (Form event=On Double Clicked)
      If (atPrijmeni#0)
         ALERT ("Poklepali jste na: "+atPrijmeni{atPrijmeni}
      End if
End case
```

Poznámka: Události musí být aktivované v okně Vlastnosti objektu.

Zatímco syntaxe atPrijmeni{$vlElem} je práce s jednotlivými prvky array, syntaxe atPrijmeni vrací číslo vybraného prvku prvku v array. Zápis atPrijmeni{atPrijmeni} znamená "hodnota vybraného prvku v array atPrijmeni." Pokud není označen žádný prvek, číslo atPrijmeni je 0 (nula), pak test If (atPrijmeni#0) detekuje kdy je a kdy není nějaký prvek vybrán.

Nastavení vybraného prvku

Stejným způsobem můžete změnit hodnotu vybraného prvku programem - přiřazením číselné hodnoty k array.

Příklady

```
    ` Označení prvního prvku (pokud array není prázdný)
atPrijmeni:=1


    ` Označení posledního prvku (pokud není array prázdný)
atPrijmeni:=Size of array(atPrijmeni)


    ` Odznačení vybraného prvku (pokud je nějaký), po tomto nebude vybraný žádný
atPrijmeni:=0

If ((0<atPrijmeni)&(atPrijmeni< Size of array(atPrijmeni)))
        ` Pokud je to možné vyber následující prvek
   atPrijmeni:=atPrijmeni+1
End if

If (1<atPrijmeni)
        ` Pokud je to možné, vyber předchozí prvek
   atPrijmeni:=atPrijmeni-1
End if
```

Hledání hodnoty v array

Příkaz **Find in array** vyhledá příslušnou hodnotu v array. S použitím předchozího příkladu, vyhledá následující kód prvek s hodnotou "Richard" pokud bude tato hodnota zadána do okna Žádosti:

```
$vsName:=Request("Zadejte jméno:")
If (OK=1)
   $vlElem:=Find in array (atPrijmeni;$vsName)
   If ($vlElem>0)
      atPrijmeni:=$vlElem
   Else
      ALERT ("Toto jméno "+$vsName+" neexistuje v seznamu jmen.")
   End if
End if
```

Rozevírací nabídky, rozevírací seznamy, posuvné oblasti a ovládací karty mohou být řízeny stejným způsobem. Samozřejmě není potřeba žádný příkaz k překreslení objektů na obrazovce, pokaždé když změníte hodnotu prvku nebo přidáte nebo vymažete prvek překrelení proběhne automaticky.

Poznámka: K vytvoření a použití Ovládací karty s ikonami a aktivování nebo potlačení karet musíte použít hierarchický seznam jako podporu objektu pro ovládací kartu. Jestli chcete vědět více informací, přečtěte si příklad pro příkaz New list.

Ovládání textu se seznamem

Ve chvíli kdy ovládáte rozevírací nabídku, rozevírací seznam, posuvnou oblast a ovládací kartu algoritmem popsaným v předchozí části, musíte ovládat text se seznamem jiným způsobem.

Text se seznamem je ve skutečnosti dostupná textová oblast, ke které je přiřazen seznam hodnot (prvků z array). Uživatel může použít hodnotu ze seznamu a pak text upravit. V textu se seznamem není pojem vybraný prvek důležitý.

V textu se seznamem není nikdy vybraný prvek. Pokaždé když uživatel vybere nějaký prvek v textu se seznamem je jeho hodnota předána do prvku nula (a zadávací oblasti). Pokud uživatel tuto hodnotu upraví, je tato upravená hodnota ze zadávací oblasti předána zpět do prvku nula.

Příklad

```
    ` metoda objektu asColors - text se seznamem
Case of
  ÷ (Form event=On Load)
    ARRAY STRING(31;asColors;3)
    asColors{1}:="Modrá"
    asColors{2}:="Bílá"
    asColors{3}:="Červená"
  ÷ (Form event=On Clicked)
    If (asColors{0}#"")
      ` Objekt automaticky změní svojí hodnotu
      ` Použití události Při klepnutí (On Clicked) v Textu se seznamem
      ` je vyžadováno pouze když musí být provedeny jiné akce
    End if
  ÷ (Form event=On Data Change)
      ` Find in array ignoruje prvek 0, tak vrátí -1 nebo >0
    If (Find in array(asColors;asColors{0})<0)
      ` Předáná hodnota není hodnota přiřazená k objektu
      ` Vložte hodnotu do seznamu pro příště
      $vlElem:=Size of array(asColors)+1
      INSERT ELEMENT(asColors;$vlElem)
      asColors{$vlElem}:=asColors0
    Else
      ` Předáná hodnota je mezi hodnotami přiřazenými k objektu
    End if
End case
```

Dále si přečtěte

Array, Vytvoření skupiny posuvných oblastí.

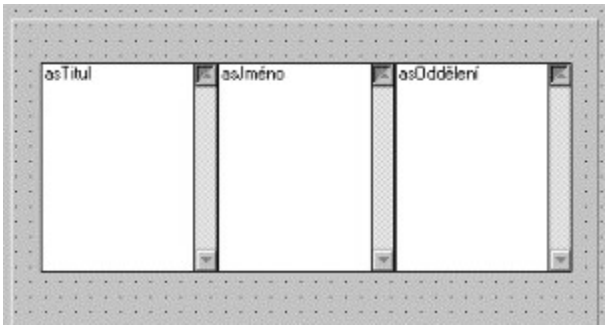
Příkazy a odkazy pro Array

Vytvoření skupiny posuvných oblastí

Příkazy a odkazy pro Array

Verze 6.0

Posuvné oblasti můžete slučovat pro zobrazení ve formuláři do skupin. Pokud je z více posuvných oblastí vytvořena skupina, fungují jako jediná posuvná oblast. Každá oblast může mít vlastní písmo a styl; nicméně doporučujeme použít stejnou výšku písma (která závisí přeně na typu písma) pro každý sloupec. Při zobrazení během vstupu dat bude zobrazovat posuvník pouze ta oblast která je na popředí. Na následujícím obrázku jsou tři posuvné oblasti sloučené do skupiny v prostředí návrháře:

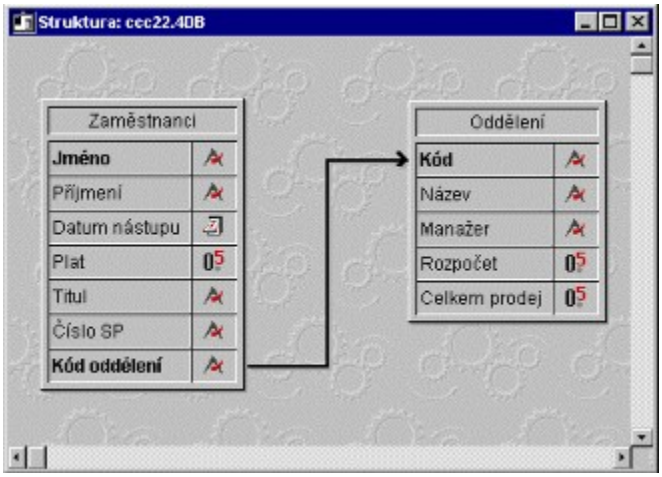


Následuje několik tipů k vytvoření skupiny posuvných oblastí:
• Ujistěte se, že všechny array mají stejný počet prvků (size)
• Použijte stejnou velikost písma pro každou oblast
• Udělejte všechny oblasti stejně vysoké
• Zarovnejte vršky všech oblastí
• Ujistěte se, že se oblasti nepřekrývají
• Ujistěte se, že oblast která je napravo je v popředí, protože posuvník se objeví u oblasti na popředí
• Vytvořte z oblastí skupinu (s použitím položky nabídky Definovat skupinu) aby mohly fungovat jako celek

Následující metoda projektu naplní tři array a zobrazí je na obrazovku:

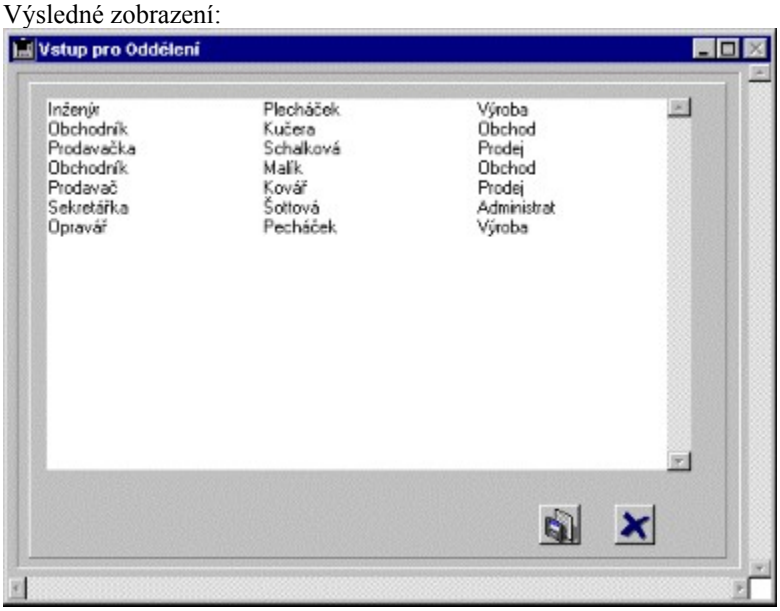
```
ALL RECORDS([Zaměstnanci])
SELECTION TO ARRAY([Zaměstnanci]Příjmení;asName;
                   [ Zaměstnanci]Titul ;asTitle;[Oddělení]Název;asDepartment)
DIALOG([Oddělení];"PříkladSkupin")
```

Tato metoda používá data z tabulek [Zaměstnanci] a [Oddělení]. Tyto tabulky jsou ukázány níže:



Poznámka: Tabulka [Oddělení] může být použita v příkaze SELECTION TO ARRAY, spolu s tabulkou

[Zaměstnanci], protože je vytvořen automatický vztah z tabulky [Zaměstnanci] k tabulce [Oddělení].
Výsledné zobrazení:

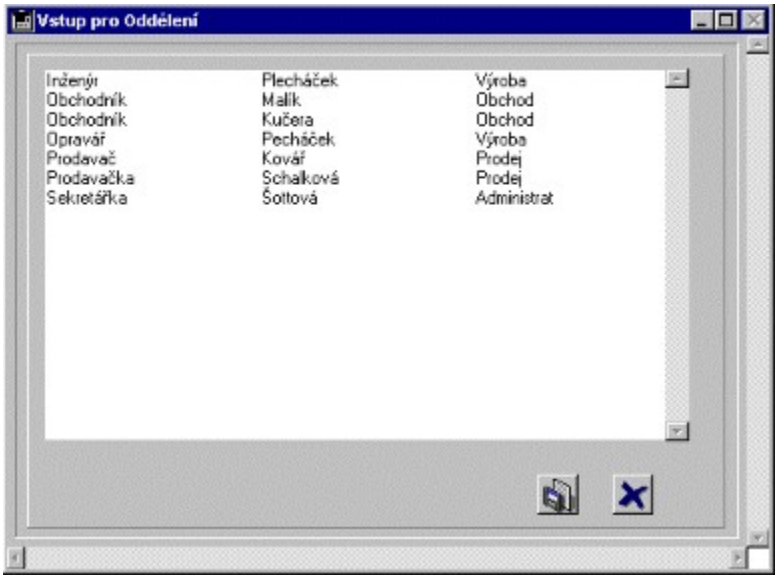


Všimněte si, že je zobrazen pouze jeden posuvník: a ten je vždy umístěn na přední posuvné oblasti. Tento posuvník posunuje všechny oblasti, jako by byly jedna. Jakmile uživatel klepne na jeden řádek, označí se tento řádek ve všech oblastech. Číselná proměnná přiřazená array se nastaví ke každé oblasti na číslo řádku na který uživatel klepne; provede se pouze ta jedna metoda objektu která je přiřazena přesně k té oblasti, na kterou uživatel klepne. Pokud například uživatel klepne na příjmení "Kučera" asName, asTitle a asDepartment jsou nastaveny na 2, ale provede se pouze metoda asName. Jakmile označíte jeden prvek v array ve skupině oblastí, bude i v ostatních array nastaven stejný prvek pro využití v další události a celá řádka v posuvné oblasti bude označena.

Array mohou být tříděna jako skupina příkazem SORT ARRAY. Například:

SORT ARRAY (asTitle;asName;asDepartment;>)

Následující obrázek ukazuje výsledek třídění:



Všimněte si, že array byla tříděna na základě prvního argumentu příkazu SORT ARRAY; další dvě oblasti byli tříděny podle první aby se zachovala synchronizace položek. Příkaz SORT ARRAY vždy třídí array (pokud je jich určeno více) podle hodnot prvního array a třídí jednotlivé řádky i v ostatních oblastech podle první.

Poznámka: SORT ARRAY neprovádí víceúrovňové třídění array. Pokud nutně potřebujete ukázat tabulku podobnou

oblastem výše a také splnit víceúrovňové třídění (podle oddělení pak podle titulu a pak podle jména), použijte podformulář ve kterém zobrazíte tabulku a pak použijte ORDER BY.

Dále si přečtěte

Array, Array a objekty formuláře.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Array a jazyk 4D

Příkazy a odkazy pro Array

Verze 6.0

Array jsou proměnné 4D. Jako jiné proměnné mají array rozsah platnosti a splňují pravidla jazyka 4D pouze s malými rozdíly.

Místní, procesové a meziprocesové array

Můžete vytvořit místní, procesové a meziprocesové array. Například:

```
ARRAY INTEGER ($aiCodes;100) ` Vytvoří místní array se 100 2 bytovými Integer hodnotami
ARRAY INTEGER (aiCodes;100) ` Vytvoří procesové array se 100 2 bytovými Integer hodnotami
ARRAY INTEGER (<>aiCodes;100) ` Vytvoří meziprocesový array se 100 2 bytovými Integer hodnotami
```

Rozsah planosti těchto array je stejný jako rozsah platnosti jiných místních, procesových a meziprocesových proměnných:

Místní array

Místní array je vytvořen pokud před jeho názvem je znak dolaru ($).

Rozsah platnosti místního array je metoda, ve které byl vytvořen. Tento array je vymazán, jakmile metoda skončí. Místní array se stejným názvem ve dvou různých metodách mohou být různého typu, protože to jsou dvě rozdílné proměnné s rozdílným rozsahem platnosti.

Pokud vytvoříte místní array v metodě formuláře, metodě objektu a v metodě projektu volané jako podprogram dvěma předchozími typy metod, array je vytvořen a vymazán pokaždé když je metoda formuláře nebo objektu spuštěna. Jinými slovy, je array vytvořen pro každou událost formuláře nebo objektu. V důsledku toho nemůžete používat místní proměnné ve formuláři ani pro zobrazení a tisk.

Jako u místních proměnných je dobré použít místní array kdykoli je to možné. Pokud to budete dělat, zminimalizujete velikost paměti potřebnou pro běh aplikace.

Procesové proměnné

Procesový array je vytvořen, pokud jeho název začíná písmenem.

Rozsah procesové array je proces, ve které byl vytvořen. Array je vymazán jakmile je proces skončen nebo přerušen. Procesové array má automaticky jeden výskyt pro proces. Tento array může být pouze jedenoho typu pro celý proces a jeho obsah je stejný pro celý proces.

Meziprocesový array

Meziprocesový array je vytvořen pokud před jeho názvem jsou znaky <> (na Windows a Macintoshi) nebo diamant (pouze na Macintoshi - Option-Shift-V na US klávesnici).

Meziprocesový array je stejný pro celou databázi. Může být využit pro přenášení dat mezi jednotlivými procesy.

Tip: Pokud předem víte že meziprocesová array bude měněn z několika procesů, což může způsobit jisté problémy, bude lepší, když budete chránit přístup k array pomocí semaforu. Jestli chcete vědět více informací, přečtěte si popis k příkazu Semaphore.

Poznámka: Procesové a meziprocesové array můžete použít k vytvoření objektů formuláře jako jsou posuvné oblasti, rozevírací nabídky, atd.

Předání array jako parametru

Array můžete předat některým příkazům nebo rutinám 4D plug-in jako parametr. Ale na druhou stránku nemůžete předat array jako parametr do metod uživatele. Jedním z řešení je předat jako parametr do metody ukazatel na array. Jestli chcete vědět více informací, přečtěte si část Array a Ukazatele.

Přiřazení array k jinému array

Na rozdíl od textových nebo řetězcových proměnných, nemůžete přiřadit jednen array k druhému jako celek, příkazem přiřazení (to je možné jen po prvcích). Ke zkopírování (přiřazení) jednoho array do jiného, jako celku, použijte příkaz COPY ARRAY.

Dále si přečtěte

Array, Array a Ukazatele.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Array a Ukazatele

Příkazy a odkazy pro Array

Verze 6.0

Array můžete předat příkazům 4D a rutinám 4D plug-in jako parametr. Ale na druhou stránku nemůžete předat array jako parametr do metod uživatele. Jedním z řešení je do metody předat jako parametr ukazatel na array.

Poznámka: Jako parametr můžete předat procesové a meziprocesové array ale ne místní array.

Zde jsou nějaké příklady:

```
If ((0<atPrijmeni)&(atPrijmeni< Size of array (atPrijmeni))
      ` Pokud je to možné, označí další prvek k označenému prvku
   atPrijmeni:=atPrijmeni+1
End if
```

Pokud potřebujete udělat stejnou věc s 50-ti různými array v různých formulářích, můžete se vyhnout nutnosti napsat stejnou věc 50 krát použitím následující metody:

```
   ` SELECT NEXT ELEMENT metoda projektu
   ` SELECT NEXT ELEMENT ( Ukazatel )
   ` SELECT NEXT ELEMENT ( → Array )
C_POINTER ($1)
If ((0<$1→)&($1→< Size of array ($1→))
   $1→:=$1→+1 ` Pokud je to možné, označí další prvek k označenému prvku
End if
```

Pak můžete napsat:

```
SELECT NEXT ELEMENT ( →atPrijmeni)
   `...
SELECT NEXT ELEMENT ( →asZipCodes)
   `...
SELECT NEXT ELEMENT ( →alRecord)
   ` atd.
```

Následující metoda vrátí součet všech prvků v číselném array (Real, Integer, Long Integer):

```
   ` Array sum
   ` Array sum ( Ukazatel)
   ` Array sum (→Array)

C_REAL ($0)

$0:=0
For ($vlElem;1;Size of array($1→))
   $0:=$0+$1→($vlElem)
End for
```

Pak můžete napsat:

```
$vlSum:= Array sum (→arPlaty)
   ` ...
$vlSum:= Array sum (→aiPopulace)
```

```
    ` ...
$vlSum:= Array sum (→alCena)
```

Následující příklad zvětší písmena všech prvků řetězcového nebo textového array:

```
    ` CAPITALIZE ARRAY
    ` CAPITALIZE ARRAY (Ukazatel)
    ` CAPITALIZE ARRAY (→Array)
For ($vlElem;1;Size of array($1→))
If ($1→{$vlElem}#"")
   $1→{$vlElem}:=Uppercase($1→{$vlElem}[[1]])
                        +Lowercase(Substring($1→{$vlElem};2))
   End if
End for
```

Pak můžete napsat:

```
CAPITALIZE ARRAY (→atObjekty)
    ` ...
CAPITALIZE ARRAY (→asPříjmení)
```

Kombinace array, ukazatelů a opakovacích struktur jako For...End for vám umožní napsat mnoho užitečných malých metod pro upravování array.

Dále si přečtěte

Array, Array a jazyk 4D.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Použití prvku nula v array

Příkazy a odkazy pro Array

Verze 6.0

Array vždy obsahuje prvek nula. Zatímco není prvek nula zobrazován při použití array ve formuláři, nejsou žádná omezení při použití s jazykem.

Jeden z příkladů použití prvku nula je v případě Textu se seznamem probíraném v části Array a objekty formuláře.

Zde jsou další dva příklady:

1. Pokud chcete spustit nějakou akci pouze v případě, že klepnete na jinou položku než tu předchozí vybranou, musíte mít někde držet každý předchozí vybraný prvek. Jeden ze způsobů jak to udělat je použít procesovou proměnnou do které budete ukládat číslo vybraného prvku. Jiný způsob je použít prvek nula:

```
    ` Metoda objektu atPrijmeni - posuvná oblast
Case of
  ÷ (Form event=On Load)
        ` Nastavení array (je ukázáno výše)
    ARRAY TEXT (atPrijmeni;5)
        ` ...
        ` Přiřazení prvku nula číslo
        ` platného označeného prvku ve formuláři
        ` Začínáte s žádným označeným prvkem
    atPrijmeni{0}:="0"
  ÷ (Form event =On Unload)
        ` Již nepotřebujeme array
    CLEAR VARIABLE(atPrijmeni)
  ÷ (Form event=On Clicked)
    If (atPrijmeni#0)
      If (atPrijmeni# Num(atPrijmeni{0}))
        vtInfo:="Klepnul jste na: "+atPrijmeni{atPrijmeni}+" tato položka nebyla označena."
        atPrijmeni{0}:=String(atPrijmeni)
      End if
    End if
  ÷ (Form event=On Double Clicked)
    If (atPrijmeni#0)
      ALERT ("Dvakrát jste klepnul na položku: "+atPrijmeni{atPrijmeni}
    End if
End case
```

2. Pokud odesíláte nebo přijímáte sérii znaků z nebo do dokumentu nebo sériového portu, 4D umožňuje způsob filtrovat (převádět) ASCII kódy mezi platformami a systémy použitím převodníku odlišných ASCII map - příkazy USE ASCII MAP, Mac to ISO, ISO to Mac, Mac to Win a Win to Mac.

Ve určitých případech budete chtít plnou kontrolu nad způsobem jak se ASCII mapa překládá. Jedním ze způsobů je použití Integer array s 255 prvky, kde Ntý prvek je nastaven na přeložení ASCII znaku který má v ASCII tabulce číslo N. Například pokud má být znak #187 přeložen na znak 156, můžete napsat <>aiCustomOutMap{187}:=156 a <>aiCustomInMap{156}:=187 do metody která nastavuje meziprocesový array použitelný v celé databázi. Pak můžete poslat sérii znaků podobnou metodou projektu:

```
    ` X SEND PACKET ( Text { ; čas } )
For ($vlChar;1;Length($1))
```

```
    $1[[vlChar]]:=Char(<>aiCustomOutMap{Ascii($1[[vlChar]])})
End for
If (Count parameters>=2)
    SEND PACKET ($2;$1)
Else
    SEND PACKET ($1)
End if

    ` X navrácení série ( Text { ; čas} ) → Text
If (Count parameters>=2)
    RECEIVE PACKET ($2;$1)
Else
    RECEIVE PACKET ($1)
End if
$0:=$1
For ($vlChar;1;Length($1))
    $0[[vlChar]]:=Char(<>aiCustomInMap{Ascii($0[[vlChar]])})
End for
```

V tomto složitějším příkladu, jestliže série obsahuje NULOVÉ znaky (ASCII kód nula), ať již odeslané nebo přijaté, prvek nula v array <>aiCustomOutNap a <>aiCustomInMap bude hrát stejnou roli jako kterýkoli jiný prvek.

Dále si přečtěte

Array.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Dvojrozměrné array

Příkazy a odkazy pro Array

Verze 6.0

Každý z příkazů pro vytvoření array může vytvořit nebo upravit jedno a dvoj-rozměrné array. Příklad:

```
   ` Vytvoří textový array složený ze 100 řad a 50 sloupců
ARRAY TEXT(atNámět;100;50)
```

Dvojrozměrné array jsou objekty jazyka: nemůžete je ani zobrazit ani tisknout.

V předchozím příkladu:
- atNámět je dvojrozměrný array
- atNámět{8}{5} je pátý prvek (5-tý sloupec) v 8. řadě
- atNámět{20} je 20. řada a sama o sobě je jednorozměrný array
- Velikost array – Size of array(atNámět) vrací 100, což je počet řad
- Velikost array – Size of array(atNámě{t17}) vrátí 50, protože to je počet sloupců na řádku 17

V následujícím příkladu jsou v dvoj-rozměrném array uloženy ukazatele ke každému poli každé tabulky v databázi:

```
   ` Vytvořit tolik řad, kolik je tabulek
ARRAY POINTER (<>apFields; Count tables;0)
   ` Pto každou tabulku
For ($vlTabulka;1;Size of array(<>apFields))
      ` Vytvořit tolik sloupců v řadě, kolik je polí
   INSERT ELEMENT (<>apFields{$vlTabulka};1;Count fields($vlTabulka))
      ` Nastavit hodnotu prvků
   For ($vlField;1;Size of array(<>apFields{$vlTabulka}))
      <>apFields{$vlTabulka}{$vlField}:=Field($vlTabulka;$vlField)
   End for
End for
```

Dvojrozměrný array je vytvořen a nyní můžeme přidat ukazatele k patřičným polím následujícím způsobem:

```
   ` Dát ukazatele k poli pro tabulku zobrazenou na obrazovce:
COPY ARRAY (<>apFields{Table(Current form table)};$apTheFieldsIamWorkingOn)
   ` Nastavení logických a datumových polí
For ($vlElem;1;Size of array($apTheFieldsIamWorkingOn))
   Case of
      ÷ (Type($apTheFieldsIamWorkingOn{$vlElem}→)=Is Date)
         $apTheFieldsIamWorkingOn{$vlElem}→:=Current date
      ÷ (Type($apTheFieldsIamWorkingOn{$vlElem}→)=Is Boolean)
         $apTheFieldsIamWorkingOn{$vlElem}→:=True
   End case
End for
```

Poznámka: Jak již tento příklad napovídá, mohou být jednotlivé řady různě nebo stejně dlouhé.

Dále si přečtěte

Array.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Array a paměť

Příkazy a odkazy pro Array

Verze 6.0

Na rozdíl od dat která uložíte na disk pomocí tabulek a záznamů, jsou array vždy uloženy pouze v paměti.

Například pokud budou všechna PSČ uložena v tabulce [PSČ], tato tabulka bude obsahovat kolem 100000 záznamů. Tato tabulka bude samozřejmě obsahovat více polí: PSČ, Město a Stát. Pokud budete například potřebovat pouze PSČ Prahy, 4D vytvoří výběr záznamů v tabulce [PSČ] a jednotlivé záznamy bude načítat až když budou potřeba (pokud bude záznam otevřen nebo tištěn). Pracujete s určitou uspořádanou množinou hodnot (tentýž typ pro každé pole), která bude načítána z disku do paměti strojem 4D.

Udělat stejnou věc s array může být nevýhodné z těchto důvodů:

• Protože potřebujete tři typy informací (psč, město, stát), potřebovali by jste tři velké array v paměti,
• Protože array jsou stále drženy v paměti, budete mít v paměti stále všechny informace o PSČ i když s nimi nepracujete,
• Pokaždé když zapnete nebo vypnete databázi, budou se tato tři array načítat z disku a opět ukládat i když je nebudete používat.

Závěr: Array jsou určeny pro uchovávání dat po krátkou dobu. Ale na druhou stránku, protože array jsou stále v paměti, je práce s nimi rychlá a snadná.

V některých případech můžete potřebovat v array stovky nebo tisíce prvků. Následující tabulka ukazuje výrazy k vypočítání množství paměti použité pro každý typ array:

Typ array	**Výraz pro výpočet paměti v bytech**
Logické	*(31+počet prvků) 8*
Datum	*(1+ počet prvků) * 6*
Řetězec	*(1+ počet prvků) * Zadaná délka (+1 za liché, +2 za sudé)*
Integer	*(1+ počet prvků) * 2*
Long Integer	*(1+ počet prvků) * 4*
Obrázek	*(1+ počet prvků) * 4 + součet velikosti každého obrázku*
Ukazatel	*(1+ počet prvků) * 16*
Real	*(1+ počet prvků) * 8 (Windows, PPC) nebo 10 (68K)*
Text	*(1+ počet prvků) * 6 + součet velikosti všech textů*
Dvojrozměrné	*(1+ počet prvků) * 12 + součet velikosti každého array*

Poznámka: Další paměť je vyžadována pro udržování počtu prvků array, vybraného prvku a definici samotného array.

Při práci s velkými array je nejlepší způsob ovládání celé paměti hlídat vytvoření array metodou projektu ON ERR CALL.

Příklad:

```
   ` Chystáte se spustit operaci na celou noc
   ` která vyžaduje vytvoření velkých array.
   ` Místo aby jste riskovali objevení chyby
   ` uprostřed noci, umístíte vytvoření array
   ` na začátek operace a otestujete chyby ihned:
gError:=0 ` Není žádná chyba
ON ERR CALL ("ERROR HANDLING") ` Naistaluje metodu pro zachytávání chyb
ARRAY STRING (63;asThisArray;50000) ` 3125K
ARRAY REAL (arThisAnotherArray;50000) ` 488K
ON ERR CALL ("") ` Již není potřeba chytat chyby
If (gError=0)
```

```
    ` Array byly vytvořeny
    ` a můžete spustit operaci
Else
    ALERT ("Tato operace vyžaduje více paměti!")
End if
    ……..` Provést cosi
    ` V tomto případě již nepotřebujeme array
CLEAR VARIABLE (asThisArray)
CLEAR VARIABLE (arThisAnotherArray)
```

Metoda projektu ERROR HANDLING je napsána zde:

```
    ` Metoda projektu ERROR HANDLING
gError:=Error     ` Pouze přiřadí kód chyby
```

Dále si přečtěte

Array, ON ERR CALL.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ARRAY INTEGER

Verze 3

ARRAY INTEGER (NázevArray; Velikost {; velikost2})

Parametr	Typ		Popis
NázevArray	Array	→	Název array
Velikost	Číslo	→	Počet prvků v array nebo počet řádků, pokud je zadána velikost2
velikost2	Číslo	→	Počet sloupců v dvojrozměrném array

Popis

Příkaz ARRAY INTEGER vytvoří nebo upraví Integer array a uloží jej do paměti.

- Parametr *NázevArray* je název array,
- Parametr *Velikost* je počet prvků v array,
- Parametr *velikost2* je volitelný: pokud je zadán tento parametr, příkaz vytvoří dvojrozměrný array. V tomto případě Velikost udává počet řádků a velikost2 počet sloupců v každém array. Každý řádek může být brán jako prvek nebo samostatný array. To znamená, že pokud pracujete s první úrovní array, můžete použít další příkazy array k předání nebo vymazání vnitřních array v dvojrozměrném array.

Při použití ARRAY INTEGER na existující array:

- Pokud zvětšíte array, zůstanou původní prvky nezměněné a nové jsou nastaveny na 0
- Pokud zmenšíte velikost array, poslední prvky se vymažou.

Příklady

1. Tento příklad vytvoří procesový Integer array se 100 prvky o 2 bytech:

```
ARRAY INTEGER(aiHodnoty;100)
```

2. Tento příklad vytvoří místní array se 100 řádky s 50 integer prvky o 2 bytech:

```
ARRAY INTEGER($aiHodnoty;100;50)
```

3. Tento příklad vytvoří meziprocesní array s 50 prvky (2byty) a do každého prvku přiřadí jeho číslo:

```
ARRAY INTEGER(<>aiHodnoty;50)
For ($vlElem;1;50)
   <>aiHodnoty($vlElem):=$vlElem
End for
```

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ARRAY LONGINT

Příkazy a odkazy pro Array

Verze 3

ARRAY LONGINT (NázevArray; velikost {; velikost2})

Parametr	Typ		Popis
NázevArray	Array	→	Název array
Velikost	Číslo	→	Počet prvků v array nebo počet řádků, pokud je zadána velikost2
velikost2	Číslo	→	Počet sloupců v dvojrozměrném array

Popis

Příkaz ARRAY LONGINT vytvoří nebo upraví Long Integer array a uloží jej do paměti.

- Parametr *NázevArray* je název array,
- Parametr *Velikost* je počet prvků v array,
- Parametr *velikost2* je volitelný: pokud je zadán tento parametr, příkaz vytvoří dvojrozměrný array. V tomto případě Velikost udává počet řádků a velikost2 počet sloupců v každém array. Každý řádek může být brán jako prvek nebo samostatný array. To znamená, že pokud pracujete s první úrovní array, můžete použít další příkazy array k předání nebo vymazání vnitřních array v dvojrozměrném array.

Při použití ARRAY LONGINT na existující array:

- Pokud zvětšíte array, zůstanou původní prvky nezměněné a nové jsou nastaveny na 0
- Pokud zmenšíte velikost array, poslední prvky se vymažou.

Příklady

1. Tento příklad vytvoří procesový Long Integer array se 100 prvky o 4 bytech:

```
ARRAY LONGINT (alHodnoty;100)
```

2. Tento příklad vytvoří místní array se 100 řádky s 50 Long Integer prvky o 4 bytech:

```
ARRAY LONGINT ($alHodnoty;100;50)
```

3. Tento příklad vytvoří meziprocesní array s 50 prvky (4byty) a přiřadí do každého jeho číslo:

```
ARRAY LONGINT (<>alHodnoty;50)
For ($vlElem;1;50)
   <>alHodnoty($vlElem):=$vlElem
End for
```

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ARRAY REAL

Příkazy a odkazy pro Array

Verze 3

ARRAY REAL (NázevArray; velikost {; velikost2})

Parametr	Typ		Popis
NázevArray	Array	→	Název array
Velikost	Číslo	→	Počet prvků v array nebo počet řádků, pokud je zadána velikost2
velikost2	Číslo	→	Počet sloupců v dvojrozměrném array

Popis

Příkaz ARRAY REAL vytvoří nebo upraví Real array a uloží jej do paměti.

• Parametr *NázevArray* je název array,
• Parametr *Velikost* je počet prvků v array,
• Parametr *velikost2* je volitelný: pokud je zadán tento parametr, příkaz vytvoří dvojrozměrný array. V tomto případě Velikost udává počet řádků a velikost2 počet sloupců v každém array. Každý řádek může být brán jako prvek nebo samostatný array. To znamená, že pokud pracujete s první úrovní array, můžete použít další příkazy array k předání nebo vymazání vnitřních array v dvojrozměrném array.

Při použití ARRAY REAL na existující array:

• Pokud zvětšíte array, zůstanou původní prvky nezměněné a nové jsou nastaveny na 0
• Pokud změnšíte velikost array, poslední prvky se vymažou.

Příklady

1. Tento příklad vytvoří procesový Real array se 100 prvky:

```
ARRAY REAL(arHodnoty;100)
```

2. Tento příklad vytvoří místní array se 100 řádky s 50 real prvky:

```
ARRAY REAL($arHodnoty;100;50)
```

3. Tento příklad vytvoří meziprocesní array s 50 prvky a každý prvek nastaví na jeho číslo:

```
ARRAY REAL(<>arHodnoty;50)
For ($vlElem;1;50)
   <>arHodnoty($vlElem):=$vlElem
End for
```

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ARRAY STRING

Příkazy a odkazy pro Array

Verze 3

ARRAY STRING(délka řetězce; NázevArray; velikost {; velikost2})

Parametr	Typ		Popis
Délkařetězce	Číslo	→	Délka řetězce
NázevArray	Array	→	Název array
velikost	Číslo	→	Počet prvků v array Pokud je zadána velikost2 počet řad
velikost2	Číslo	→	Počet sloupců dvojrozměrného array

Popis

Příkaz ARRAY STRING vytvoří nebo upraví array řetězcových prvků.

• Parametr *délkařetězce* definuje maximální počet znaků v jednom prvku. Délka může být od 1 do 255
• Parametr *NázevArray* je název array
• Parametr *velikost* udává počet prvků v array
• Parametr *velikost2* je volitelný: pokud je zadán tento parametr, příkaz vytvoří dvojrozměrný array. V tomto případě Velikost udává počet řádků a velikost2 počet sloupců v každém array. Každý řádek může být brán jako prvek nebo samostatný array. To znamená, že pokud pracujete s první úrovní array, můžete použít další příkazy array k předání nebo vymazání vnitřních array v dvojrozměrném array.

Použití příkazu ARRAY STRING na existujíxí array:

• Pokud budete array zvětšovat, exitující prvky zůstanou nezměněné a nové jsou nastaveny na "" (prázdný řetězec)
• Pokud array zmenšujete, budou vymazány poslední prvky.

Příklady

1. Tento příklad vytvoří procesový array se 100 prvky o 31 znacích:

```
ARRAY STRING(31;asHodnoty;100)
```

2. Tento příklad vytvoří místní array se 100 řádky s 50 prvky o délce 63 znaků:

```
ARRAY STRING (63;$asHodnoty;100;50)
```

3. Tento příklad vytvoří meziprocesní array s 50 prvky o délce 255 znaků a každý prvek nastaví na hodnotu Prvek #:

```
ARRAY STRING (255;<>asHodnoty;50)
For ($vlElem;1;50)
   <>asHodnoty($vlElem):="Prvek #" +String($vlElem)
End for
```

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

ARRAY TEXT

Příkazy a odkazy pro Array

Verze 3

ARRAY TEXT (NázevArray; velikost {; velikost2})

Parametr	Typ		Popis
NázevArray	Array	→	Název array
Velikost	Číslo	→	Počet prvků v array nebo počet řádků, pokud je zadána velikost2
velikost2	Číslo	→	Počet sloupců v dvojrozměrném array

Popis

Příkaz ARRAY TEXT vytvoří nebo upraví array textových prvků.

• Parametr *NázevArray* je název array
• Parametr *velikost* udává počet prvků v array
• Parametr *velikost2* je volitelný: pokud je zadán tento parametr, příkaz vytvoří dvojrozměrný array. V tomto případě Velikost udává počet řádků a velikost2 počet sloupců v každém array. Každý řádek může být brán jako prvek nebo samostatný array. To znamená, že pokud pracujete s první úrovní array, můžete použít další příkazy array k předání nebo vymazání vnitřních array v dvojrozměrném array.

Použití příkazu ARRAY TEXT na existující array:

• Pokud budete array zvětšovat, existující prvky zůstanou nezměněné a nové jsou nastaveny na "" (prázdný řetězec)
• Pokud array zmenšujete, budou vymazány poslední prvky.

Příklady

1. Tento příklad vytvoří procesový array se 100 textovými prvky:

```
ARRAY TEXT(atHodnoty;100)
```

2. Tento příklad vytvoří místní array se 100 řádky s 50 textovými prvky:

```
ARRAY TEXT ($atHodnoty;100;50)
```

3. Tento příklad vytvoří meziprocesní array s 50 textovými prvky a každý prvek nastaví na hodnotu Prvek #:

```
ARRAY TEXT (<>atHodnoty;50)
For ($vlElem;1;50)
   <>atHodnoty($vlElem):="Prvek #" +String($vlElem)
End for
```

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ARRAY DATE

Příkazy a odkazy pro Array

Verze 3

ARRAY DATE (NázevArray; velikost {; velikost2})

Parametr	Typ		Popis
NázevArray	Array	→	Název array
Velikost	Číslo	→	Počet prvků v array nebo počet řádků, pokud je zadána velikost2
velikost2	Číslo	→	Počet sloupců v dvojrozměrném array

Popis

Příkaz ARRAY DATE vytvoří nebo upraví array datumových prvků.

• Parametr *NázevArray* je název array
• Parametr *velikost* udává počet prvků v array
• Parametr *velikost2* je volitelný: pokud je zadán tento parametr, příkaz vytvoří dvojrozměrný array. V tomto případě Velikost udává počet řádků a velikost2 počet sloupců v každém array. Každý řádek může být brán jako prvek nebo samostatný array. To znamená, že pokud pracujete s první úrovní array, můžete použít další příkazy array k předání nebo vymazání vnitřních array v dvojrozměrném array.

Použití příkazu ARRAY DATE na existující array:

• Pokud budete array zvětšovat, exitující prvky zůstanou nezměněné a nové jsou nastaveny na nulové datum (! 00:00:00!)
• Pokud array zmenšujete, budou vymazány poslední prvky.

Příklady

1. Tento příklad vytvoří procesový array se 100 datumovými prvky:

```
ARRAY DATE(adHodnoty;100)
```

2. Tento příklad vytvoří místní array se 100 řádky s 50 datumovými prvky:

```
ARRAY DATE ($adHodnoty;100;50)
```

3. Tento příklad vytvoří meziprocesní array s 50 datumovými prvky a každý prvek nastaví na platné datum plus počet dní stejný jako číslo prvku:

```
ARRAY DATE (<>adHodnoty;50)
For ($vlElem;1;50)
   <>adHodnoty($vlElem):=Current date+$vlElem
End for
```

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ARRAY BOOLEAN

Příkazy a odkazy pro Array

Verze 3

ARRAY BOOLEAN (NázevArray; velikost {; velikost2})

Parametr	Typ		Popis
NázevArray	Array	→	Název array
Velikost	Číslo	→	Počet prvků v array nebo počet řádků, pokud je zadána velikost2
velikost2	Číslo	→	Počet sloupců v dvojrozměrném array

Popis

Příkaz ARRAY BOOLEAN vytvoří nebo upraví array logických prvků.

• Parametr *NázevArray* je název array
• Parametr *velikost* udává počet prvků v array
• Parametr *velikost2* je volitelný: pokud je zadán tento parametr, příkaz vytvoří dvojrozměrný array. V tomto případě Velikost udává počet řádků a velikost2 počet sloupců v každém array. Každý řádek může být brán jako prvek nebo samostatný array. To znamená, že pokud pracujete s první úrovní array, můžete použít další příkazy array k předání nebo vymazání vnitřních array v dvojrozměrném array.

Použití příkazu ARRAY BOOLEAN na existující array:

• Pokud budete array zvětšovat, exitující prvky zůstanou nezměněné a nové jsou nastaveny False (Nepravda)
• Pokud array zmenšujete, budou vymazány poslední prvky.

Tip: V některých případech je možné místo Logického array použít Integer array, kde pokud je číslo jiné než nula, znamená Pravda a pokud je nula, znamená Nepravda.

Příklady

1. Tento příklad vytvoří procesový array se 100 logickými prvky:

```
ARRAY BOOLEAN (abHodnoty;100)
```

2. Tento příklad vytvoří místní array se 100 řádky s 50 logickými prvky:

```
ARRAY BOOLEAN ($adHodnoty;100;50)
```

3. Tento příklad vytvoří meziprocesní array s 50 logickými prvky a nastaví každý sudý prvek na Pravda (True):

```
ARRAY BOOLEAN (<>abHodnoty;50)
For ($vlElem;1;50)
   <>abHodnoty($vlElem):=(($vlElem%2)=0)
End for
```

ARRAY PICTURE

Příkazy a odkazy pro Array

Verze 3

ARRAY PICTURE (NázevArray; velikost {; velikost2})

Parametr	Typ		Popis
NázevArray	Array	→	Název array
Velikost	Číslo	→	Počet prvků v array nebo počet řádků, pokud je zadána velikost2
velikost2	Číslo	→	Počet sloupců v dvojrozměrném array

Popis

Příkaz ARRAY PICTURE vytvoří nebo upraví array obrázkových prvků.

• Parametr *NázevArray* je název array
• Parametr *velikost* udává počet prvků v array
• Parametr *velikost2* je volitelný: pokud je zadán tento parametr, příkaz vytvoří dvojrozměrný array. V tomto případě Velikost udává počet řádků a velikost2 počet sloupců v každé array. Každý řádek může být brán jako prvek nebo samostatný array. To znamená, že pokud pracujete s první úrovní array, můžete použít další příkazy array k předání nebo vymazání vnitřních array v dvojrozměrném array.

Použití příkazu ARRAY PICTURE na existující array:

• Pokud budete array zvětšovat, exitující prvky zůstanou nezměněné a nové jsou nastaveny na prázdný obrázek To znamená, že při zjišťování velikosti tohoto obrázku, se vám navrátí 0.
• Pokud array zmenšujete, budou vymazány poslední prvky.

Příklady

1. Tento příklad vytvoří procesový array se 100 obrázkovými prvky:

```
ARRAY PICTURE(agHodnoty;100)
```

2. Tento příklad vytvoří místní array se 100 řádky s 50 obrázkovými prvky:

```
ARRAY PICTURE ($agHodnoty;100;50)
```

3. Tento příklad vytvoří meziprocesní array s obrázkovými prvky a načte obrázky do prvků array. Velikost array je stejná jako počet "PICT" zdrojů dostupných v databázi. Název zdroje array začíná na "User Intf/"

```
RESOURCE LIST("PICT";$aiResIDs;$asResNames)
ARRAY PICTURE (<>agValues;Size of array($aiResIDs))
$vlPictElem:=0
For ($vlElem;1;Size of array(<>agValues))
   If ($asResNames="User Intf/@")
      $vlPictElem:=vlPictElem+1
      GET PICTURE RESOURCE("PICT";$aiResIDs{$vlElem};$vgPicture)
      <>agValues{$vlPictElem}:=$vgPicture
   End if
End for
ARRAY PICTURE (<>agValues;$vlPictElem)
```

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ARRAY POINTER

Příkazy a odkazy pro Array

Verze 3

ARRAY POINTER (NázevArray; velikost {;velikost2})

Parametr	Typ		Popis
NázevArray	Array	→	Název array
Velikost	Číslo	→	Počet prvků v array nebo počet řádků, pokud je zadána velikost2
velikost2	Číslo	→	Počet sloupců v dvojrozměrném array

Popis

Příkaz ARRAY POINTER vytvoří nebo upraví array ukazatelů.

• Parametr *NázevArray* je název array
• Parametr *velikost* udává počet prvků v array
• Parametr *velikost2* je volitelný: pokud je zadán tento parametr, příkaz vytvoří dvojrozměrný array. V tomto případě Velikost udává počet řádků a velikost2 počet sloupců v každém array. Každý řádek může být brán jako prvek nebo samostatný array. To znamená, že pokud pracujete s první úrovní array, můžete použít další příkazy array k předání nebo vymazání vnitřních array v dvojrozměrném array.

Použití příkazu ARRAY POINTER na existující array:

• Pokud budete array zvětšovat, exitující prvky zůstanou nezměněné a nové jsou nastaveny na nulový ukazatel. To znamená, že Nil použitý na jednu z těchto položek vrátí Pravda.
• Pokud array zmenšujete, budou vymazány poslední prvky.

Příklady

1. Tento příklad vytvoří procesový array se 100 ukazateli:

```
ARRAY POINTER(apHodnoty;100)
```

2. Tento příklad vytvoří místní array se 100 řádky s 50 ukazateli:

```
ARRAY POINTER ($apHodnoty;100;50)
```

3. Tento příklad vytvoří meziprocesní array ukazatelů a každý prvek nastaví k tabulce souhlasící s číslem prvku. Velikost array je stejná jako počet tabulek:

```
ARRAY POINTER (<>apHodnoty;Count tables)
For ($vlElem;1;Size of array(<>apHodnoty))
   <> apHodnoty $vlElem:=Table($vlElem)
End for
```

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

Size of array (Velikost array)

Příkazy a odkazy pro Array

Verze 3

Size of array (array) číslo

Parametr	Typ		Popis
array	array	→	Array pro který chcete velikost
Výsledek funkce	číslo	←	Vrátí počet prvků array

Popis

Příkaz Size of array vrátí počet prvků v array.

Příklad

1. Následující příklad vrátí velikost array anArray:

```
vlVelikost:=Size of array(anArray)    ` vlVelikost obsahuje velikost anArray
```

2. Následující příklad vrátí počet řad ve dvojrozměrném array:

```
vlŘada:=Size of array(a2dArray)    ` vlŘada obsahuje počet řad v array
```

3. Následující příklad vrátí počet sloupců v řadě dvojrozměrného array:

```
vlSloupce:=Size of array(a2dArray(10))    ` vlSloupce obsahuje počet sloupců v řadě
```

Dále si přečtěte

DELETE ELEMENT, INSERT ELEMENT.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SORT ARRAY (TŘÍDIT ARRAY)

Příkazy a odkazy pro Array

Verze 3

SORT ARRAY (array {;array2; ...; arrayN} {; > nebo <})

Parametr	Typ		Popis
array	Array	→	Array k třídění
> nebo <		→	> třídit ve vzestupném pořadí < třídit v sestupném pořadí Pokud nezadáte žádný znak, bude pořadí vzestupné

Popis

Příkaz SORT ARRAY třídí jeden nebo více array ve vzestupném nebo sestupném pořadí.

Poznámka: Nemůžete třídit array ukazatelů nebo obrázků. Můžete třídit prvky dvojrozměrných array, ale nemůžete třídit celý dvojrozměrný array.

Poslední parametr definovaný u třídění je zadání vzestupného nebo sestupnémho třídění. Symbol "větší než" (>) zadává vzestupné třídění a znak "menší než" (<) udává sestupné třídění. Pokud nezadáte ani jeden z těchto znaků, bude se array třídit vzestupně.

Pokud je zadán více než jeden array k třídění, jsou array tříděny stejně jako první array: není zde prováděno víceúrovňové třídění. Tato vlastnost je zejména použitelná ve skupině array ve formuláři: příkaz SORT ARRAY zachovává synchronizaci prvků ve skupině posuvných oblastí.

Příklady

1. Následující příklad vytvoří dvě array a pak je setřídí podle firmy:

```
ALL RECORDS ([Lidé])
SELECTION TO ARRAY ([Lidé]Jméno;asJméno;[Lidé]Firma;asFirma)
SORT ARRAY (asFirma; asJméno;>)
```

Protože SORT ARRAY neprovádí víceúrovňové třídění, skončí třídění se jmény v náhodném pořadí v každé firmě. K třídění lidí podle jména pro každou firmu, musíte napsat:

```
ALL RECORDS ([Lidé])
ORDER BY ([Lidé];[Lidé]Firma;>;[Lidé]Jméno;>)
SELECTION TO ARRAY ([Lidé]Jméno;asJméno;[Lidé]Firma;asFirma)
```

2. Zobrazíte jména z tabulky [Lidé] v plovoucím okně. Při klepnutí na tlačítko v okně, můžete třídit jména od A do Z nebo od Z do A. Více lidí ale může mít stejné jméno, použijete ještě pole [Lidé]ID, které je jedinečné indexované. Jakmile klepnete na seznam jmen, dostanete záznam který patří k tomuto jménu. Vzhledem k zachování synchronizace a skrytému array ID čísel si můžete být jisti že záznam který se otevře je správný:

```
    ` metoda objektu asNames
Case of
  ÷ (Form event=On Load)
    ALL RECORDS([Lidé])
    SELECTION TO ARRAY([Lidé]Jméno;asJméno;[Lidé]ID;alIDs)
    SORT ARRAY(asJméno;alIDs;>)
```

```
    ÷ (Form event=On Unload)
        CLEAR VARIABLE(asJméno)
        CLEAR VARIABLE(alIDs)
    ÷ (Form event=On Clicked)
        If (asJméno#0)
                ` Použít array alIDs k uchování pravého záznamu
            QUERY([Lidé];[Lidé]ID=alIDs{asJméno})
                ` Udělat něco se záznamem
        End if
End case
        ` metoda objektu bA2Z tlačítko
        ` Třídit array ve vzestupném pořadí a uchová je synchronizované
SORT ARRAY(asJméno;alIDs;>)
        ` metoda objektu bZ2A tlačítko
        ` Třídit array v sestupném pořadí a uchová je synchronizované
SORT ARRAY(asJméno;alIDs;<)
```

Dále si přečtěte

ORDER BY, SELECTION TO ARRAY.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Find in array (Nalézt v array)

Příkazy a odkazy pro Array

Verze 3

Find in array (array; hodnota {; start}) → číslo

Parametr	Typ		Popis
array	array	→	Array ve kterém se bude hledat
hodnota	Výraz	→	Hodnota stejného typu k vyhlednání v array
start	číslo	→	Prvek na kterém se má začít
Výsledek funkce	číslo	←	číslo prvního prvku v array který odpovídá v hodnotě

Popis

Příkaz Find in array vrátí číslo prvního prvku v *array*, který odpovídá hledané hodnotě.

Find in array může být použit na Text, Řetězec, Číselné, Datumové, Ukazatele a logické array. Parametry *array* a *hodnota* musí být stejného typu.

Pokud není nalezen žádný prvek, Find in array vrátí -1.

Pokud je definován parametr *Start*, příkaz započne hledání na položce jejíž číslo odpovídá zadanému parametru start. Pokud tento parametr není zadán, hledání začne na začátku.

Příklady

1. Následující metoda projektu vymaže všechny prázdné položky textového nebo řetězcového array jehož ukazatel je přiřazen jako parametr:

```
    ` CLEAN UP ARRAY metoda rojektu
    ` CLEAN UP ARRAY ( Ukazatel )
    ` CLEAN UP ARRAY ( → Text nebo Řetězec array )
C_POINTER ($1)
Repeat
   $vlElem:=Find in array ($1→;"")
   If ($vlElem>0)
      DELETE ELEMENT ($1→;$vlElem)
   End if
Until ($vlElem<0)
```

Po té co předáte tuto tuto metodu do databáze, můžete napsat:

```
ARRAY STRING(asHodnoty;....)
   ` ...
   ` Udělejte něco s array
   ` ...
   ` Vymazat prázdné prvky array
CLEAN UP ARRAY (→asHodnoty)
```

2. Následující metoda projektu označí první prvek v array jehož ukazatel je přiřazen jako první parametr, který je hodnotou pole nebo proměnné jejíž ukazatel je přiřazen jako parametr:

```
    ` SELECT ELEMENT metoda projektu
```

```
` SELECT ELEMENT ( Ukazatel ; Ukazatel)
` SELECT ELEMENT ( → Textový nebo řetězcový array ; → Textová n. řetězcová proměnná nebo pole )
$1→:=Find in array ($1→;$2→)
If ($1→=-1)
   $1→:=0 ` Pokud nebyl nalezen žádný prvek, nastaví array na neoznačený prvek
End if
```

Po té co předáte tuto metodu do databáze, můžete napsat:

```
   ` metoda objektu asGender - rozevírací nabídka
Case of
   ÷ (Form event=On Load)
      SELECT ELEMENT (→ asGender; →[Lidé]Gender)
End case
```

Dále si přečtěte

DELETE ELEMENT, INSERT ELEMENT, Size of array.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

INSERT ELEMENT (VLOŽIT PRVEK)

Verze 3

INSERT ELEMENT (array; kam {; kolik})

Parametr	Typ		Popis
array	array	→	Název array
kam	číslo	→	Kam vložit prvek
kolik	číslo	→	Počet prvků k předání 1 pokud není definováno

Popis

Příkaz INSERT ELEMENT vloží jeden nebo více prvků do *array*. Prvek je předán před prvek definovaný parametrem *Kam* a je inicializován na prázdnou hodnotu stejného typu jako *array*. Prvky které jsou za předáným prvkem se posunou a přečíslují se o počet definovaný o *kolik*, pokud je tento parametr nedefinován tak se posunou o jeden prvek.

Pokud je parametr *kam* větší než velikost *array* je prvek předán na konec *array*.

Parametr *kolik* je počet prvků k předání. Pokud není tento parametr definován, je předán jeden prvek. Velikost *array* se zvětší o *kolik*.

Příklady

1. Následující řádek vloží 5 prvků začínaje na prvku 10:

```
INSERT ELEMENT (anArray;10;5)
```

2. Následující příklad přidá prvky do array:

```
$vlElem:=Size of array(anArray) + 1
INSERT ELEMENT (anArray;$vlElem)
anArray:=...
```

Dále si přečtěte

DELETE ELEMENT, Size of array.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DELETE ELEMENT (VYMAZAT PRVEK)

Verze 3

DELETE ELEMENT (array; kde{; kolik})

Parametr	Typ		Popis
array	array	→	Array ze kterého vymazat prvek
kde	číslo	→	Prvek na kterém začít mazání
kolik	číslo	→	Počet prvků k vymazání 1 pokud není definováno

Popis

Příkaz **DELETE ELEMENT** vymaže jeden nebo více prvků z *array*. Položky se začnou mazat od položky definované parametrem *kde*.

Parametr *kolik* je počet prvků které se mají vymazat. Pokud není definováno, vymaže se jeden prvek. Array se zmenší o *kolik*.

Příklady

1. Následující řádek vymaže tři prvky a začne na prvku 5:

```
DELETE ELEMENT(ANARRAY; 5; 3)
```

2. Následující příklad vymaže poslední prvek z array, pokud existuje:

```
$vlElem:=Site of array (anArray)
If ($vlElem>0)
   DELETE ELEMENT(anArray; $vlElem)
End if
```

Dále si přečtěte

INSERT ELEMENT, Size of array.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

COPY ARRAY (KOPÍROVAT ARRAY)

Příkazy a odkazy pro Array

Verze 3

COPY ARRAY (zdroj; cíl)

Parametr	Typ		Popis
zdroj	array	→	Array ze kterého se bude kopírovat
cíl	array	→	Array do kterého se bude kopírovat

Popis

Příkaz COPY ARRAY vytvoří nebo přepíše array *cíl* a nahradí jeho obsah, velikost a typ array *zdroj*.

Cílový i zdrojový array mohou být místní, procesové nebo meziprocesové. Při kopírování array nezáleží na jeho velikosti.

Příklady

Následující příklad naplní array C. Dále vytvoří array D se stejnou velikostí, obsahem a typem jako array C:

```
ALL RECORDS ([Lidé]) ` Vybere všechny záznamy z tabulky [Lidé]
SELECTION TO ARRAY ([Lidé]Firma; C) ` Přesune data firem do array C
COPY ARRAY (C; D) ` Zkopíruje array C do array D
```

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

LIST TO ARRAY (SEZNAM DO ARRAY)

Příkazy a odkazy pro Array

Verze 3

Poznámka kompatibility

Vzhledem k předání nového Výběrového seznamu, nemusí být kompatibilita tohoto příkazu plně podporována. Ve verzi 6 doporučujeme spíše používat příkaz Load List k práci s hierarchickými seznamy definovanými v prostředí návrháře.

LIST TO ARRAY (seznam; array {; OdkPol})

Parametr	Typ		Popis
seznam	řetězec	→	Seznam ze kterého kopírovat první úroveň položek
array	Array	←	Array do kterého kopírovat seznam položek
OdkPol	Array	←	Seznam čísel odkazů položek.

Popis

Příkaz LIST TO ARRAY vytvoří nebo přepíše *array* první úrovní položek ze seznamu *seznam*.

Pokud nemáte definovaný array jako textový nebo řetězcový, LIST TO ARRAY vytvoří textový array jako výchozí nastavení.

Volitelný parametr *OdkPol* (číselný array) vrací seznam čísel odkazů položek.

Poznámka kompatibility: V předchozí verzi 4D byl tento array naplněn názvy jakéhokoli přiřazeného seznamu. Pokud měl prvek seznamu přiřazený seznam, název přiřazeného seznamu byl předán do prvku array se stejným číslem jako číslo prvku v seznamu. Pokud nebyl přiřazený žádný seznam, pak byl prvek prázdný řetězec. Druhý array byl nastaven na stejnou velikost jako array. Mohli jste použít názvy v těchto array k přístupu k přiřazeným seznamům.

Můžete dále používat příkazu LIST TO ARRAY k vytvoření array z první úrovně hierarchického seznamu. Příkaz vám však neumožní přístup k položkám potomků, pokud nějaké jsou. K práci s hierarchickým seznamem použijte a přečtěte si nové příkazy Hierarchického seznamu použité ve verzi 6.

Příklad

Následující příklad zkopíruje položky seznamu nazvaného Regiony do array nazvaného atRegiony:

```
LIST TO ARRAY ("Regiony";atRegiony)
```

Dále di přečtěte

ARRAY TO LIST, Load list, SAVE LIST.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ARRAY TO LIST (ARRAY DO SEZNAMU)

Verze 3

Poznámka kompatibility

Vzhledem k novému předání výběrového seznamu, nemůže být zajištěna plná kompatibilita tohoto příkazu. Ve verzi 6 doporučujeme použít příkaz SAVE LIST k práci s hierarchickými seznamy vytvořenými v prostředí návrháře.

ARRAY TO LIST (array; Seznam {; OdkPol})

Parametr	Typ		Popis
array	Array	→	Array ze kterého kopírovat prvky
Seznam	Řetězec	→	Seznam do kterého kopírovat prvky array
OdkPol	Array	→	Číselný array odkazů na položky

Popis

Příkaz ARRAY TO LIST vytvoří nebo nahradí seznam "*Seznam*" (jak je definovaný v prostředí návrháře v Editoru seznamů) s použitím prvků array "*array*".

Tento příkaz vám umožní vytvořit pouze první úroveň seznamu.

Volitelný parametr *OdkPol*, pokud je definován, musí být číselný array synchronizovaný s array "*array*". Každý prvek pak obsahuje číslo odkazu na položky seznamu odpovídající prvku *array*. Pokud vynecháte tento parametr, 4D automaticky nastaví čísla odkazů položek v seznamu na 1, 2, ...N.

Poznámka kompatibility: V předchozí verzi 4D byl tento parametr použit k přiřazení jiných seznamů k prvkům array. Pokud byl název přiřazeného prvku v array shodný s názvem nějakého seznamu, byl tento seznam přiřazen k prvku.

Příkaz LIST TO ARRAY můžete dále používat k vytvoření seznamu založeném na prvcích array. Tento příkaz však neumí zacházet s položkami potomků. K práci s hierarchickými seznamy použijte a přčtěte si nové příkazy Hierarchického seznamu předáné do verze 6.

Příklad

Následující příklad kopíruje array atRegiony do seznamu nazvaného "Regiony:"

```
ARRAY TO LIST (atRegiony; "Regiony")
```

Ovládání chyb

Chyba -9957 je vygenerována pokud příkaz ARRAY TO LIST bude použit na seznam který je právě upravován v prostředí návrháře. Můžete tuto chybu zachytit s použitím metody projektu ON ERR CALL.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

SELECTION TO ARRAY (VÝBĚR DO ARRAY)

Příkazy a odkazy pro Array

Verze 3

SELECTION TO ARRAY (pole | tabulka; array {;pole2 | tabulka2;....;poleN | tabulkaN; arrayN})

Parametr	Typ		Popis
pole | tabulka	Pole | tabulka	→	Pole použité pro získání dat nebo tabulka použitá pro získání číslel záznamů
array	array	←	Array pro přijetí dat nebo čísel záznamů

Popis

Příkaz SELECTION TO ARRAY vytvoří jeden nebo více *array* a zkopíruje data v polích nebo čísla záznamů z aktuálního výběru do array.

Příkaz SELECTION TO ARRAY je použit na tabulku definovanou v prvním parametru. SELECTION TO ARRAY může provádět následující:

• Načíst hodnoty z jednoho nebo více polí
• Načíst čísla záznamů s použitím zápisu ...;[tabulka];array;...
• Načíst hodnoty ze vztažených polí v případě že používáte automatický vztah Jedince ke Skupině nebo jste před použitím tohoto příkazu použili příkaz AUTOMATIC RELATION k přepnutí ručního vztahu Jednice ke Skupině do automatického. V obou případech jsou data načtena z tabulky spojené nějakým typem vztahu.

Každý array musí být typu jako je typ pole. Jsou dvě výjimky:
• Textové pole může být kopírováno do Řetězcových array.
• Časové pole může být kopírováno do Long Integer array.

Poznámka: Nemůžete pro kopírování určit podtabulku nebo podpole.

Pokud načítáte čísla záznamů, jsou tyto zkopírovány do Long Integer array.

4D Server: Příkaz SELECTION TO ARRAY je optimalizován pro 4D Server. Každý array je vytvářen na 4D Server a je posílán případně, podle rozsahu platnosti array, na stroj klienta.

Upozornění: Příkaz SELECTION TO ARRAY může vytvořit velké array podle jeho velikosti, kterou definujete a podle typu a velikosti dat, která načítáte. Array je umístěno v paměti a proto je dobré po použití tohoto příkazu otestovat výsledek. Proveďte to tak, že otestujete velikost každého array nebo hlídejte provádění příkazu pomocí ON ERR CALL.

Poznámka: Po použití příkazu SELECTION TO ARRAY je platný výběr i platný záznam stále stejný, ale platný záznam není načtený do paměti. Jestliže potřebujete použít hodnoty polí z platného záznamu, použijte LOAD RECORD po příkazu SELECTION TO ARRAY.

Příklady

1. V následujícím příkazu má tabulka [Lidé] automatický vztah k tabulce [Firmy]. Dvě array asPříjmení a asAdrSpolečnost jsou změněny podle počtu záznamů v tabulce [Lidé] a obsahují informace z obou tabulek:

```
SELECTION TO ARRAY([Lidé]Příjmení; asPříjmení;[Firmy]Adresy; asAdrSpolečnost)
```

2. Následující příklad vrací čísla záznamů z tabulky [Zákazníci] do array alCislaZaznamu a hodnotu pole [Zákazníci]Příjmení do array asPříjmení:

```
SELECTION TO ARRAY([Zákazníci];alCislaZaznamu;[ Zákazníci]Příjmení; asPříjmení)
```

Dále si přečtěte

ARRAY TO SELECTION, AUTOMATIC RELATIONS, ON ERR CALL, SELECTION TO ARRAY.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SELECTION RANGE TO ARRAY (ROZSAH VÝBĚRU DO ARRAY)

Příkazy a odkazy pro Array

Verze 3.5.3

SELECTION RANGE TO ARRAY(start; konec; pole | tabulka; array {;pole2 | tabulka2; array2;....; poleN | tabulkaN; arrayN})

Parametr	**Typ**		**Popis**
start	Číslo	→	Číslo vybraného záznamu kde začít přenos dat
konec	Číslo	→	Číslo vybraného záznamu kde skončit přenos dat
pole | tabulka	Pole n.Tabulka	→	Pole k použití pro přenos hodnot polí nebo tabulka pro vytvoření array z čísel záznamů
array	Array	←	Array do kterého uložit data

Popis

Příkaz SELECTION RANGE TO ARRAY vytvoří jeden nebo více array *array* a zkopíruje do něj hodnoty *pole* nebo čísla záznamů z platného výběru tabulky *tabulka*.

Na rozdíl od příkazu SELECTION TO ARRAY který používá celý výběr, příkaz SELECTION RANGE TO ARRAY používá pouze rozsah zadaný parametry *start* a *konec*.

Příkaz očekává, že použijete parametry *start* a *konec* tak, aby vyhověli podínce 1 <= start <= konec <= Records in selection ([...]).

Pokud předáte parametry 1 <= start = konec < Records in selection ([...]) načtete pole nebo dostanete číslo záznamu ze záznamu jehož číslo záznamu ve výběru (**Selected record number**) je start = konec.

Pokud předáte špatné číslo záznamu, příkaz provede následující:
• Jestliže *konec* > Records in selection ([...]), budou vráceny hodnoty polí definované od parametru start až do posledního záznamu ve výběru.
• Jestliže start > konec, vrátí pouze hodnoty polí ze záznamu definovaného parametrem start.
• Jestliže jsou oba parametry nekonzistentní s velikostí výběru, bude navrácen prázdný array.

Stejně jako SELECTION TO ARRAY, je i příkaz SELECTION RANGE TO ARRAY používán na výběr tabulky definované v prvním parametru *tabulka*.

Také jako SELECTION TO ARRAY, SELECTION RANGE TO ARRAY může provádět následující:
• Načíst hodnoty z jednoho nebo více polí,
• Načítat čísla záznamů s použitím zápisu ...;[Tabulka];array;...
• Načíst hodnoty ze vztažených polí v případě že používáte automatický vztah Jednice ke Skupině nebo jste před použitím tohoto příkazu použili příkaz AUTOMATIC RELATIONS k změně ručního vztahu Jednice ke Skupině na automatický. V obou případech jsou data načtena z tabulky spojené nějakým typem vztahu.

Každý array musí být stejného typu jako pole. Jsou dvě výjimky:
• Textové pole může být kopírováno do Řetězcových array.
• Časové pole může být kopírováno do LongInteger array.

Poznámka: Nemůžete kopírovat podtabulku nebo podpole.

Pokud načítáte čísla záznamů musí být tyto kopírovány do LongInteger array.

4D Server: Příkaz SELECTION TO ARRAY je optimalizován pro 4D Server. Každý array je vytvářen na 4D Server a je posílán případně, podle rozsahu platnosti array, na stroj klienta.

Upozornění: Příkaz SELECTION TO ARRAY může vytvořit velké array podle velikosti array, jakou definujete a podle typu a velikosti dat která načítáte. Array jsou umístěny v paměti a proto je dobré při použití tohoto příkazu otestovat výsledek. Provedete to tak, že otestujete velikost každého array nebo hlídejte provádění příkazu pomocí ON ERR CALL.

Pokud je příkaz dokončen je velikost každého výsledného array rovna (start-konec)+ 1, pokud je konec definován správně a ne za koncem výběru. Ve tomto případě obsahuje array (Records in selection([...])-start) + 1 prvků.

Příklady

Následující kód bude pracovat s prvními 50-ti záznamy aktuálního výběru v tabulce [Faktury]. Načte hodnoty z pole [Faktury]ID_Faktury a ze vztaženého pole [Zákazníci]ID_Zákazníka.

```
SELECTION RANGE TO ARRAY(1;50;[Faktury]ID_Faktury;alFaID;
                         [Zákazníci]ID_Zákazníka;alZakID)
```

2. Následující kód pracuje s posledními 50-ti záznamy z aktuálního výběru pro tabulku [Faktury]. Načte čísla záznamů z tabulky [Faktury] a ze vztažené tabulky [Zákazníci]:

```
lVelVyb := Records in selection ([Faktury])
SELECTION RANGE TO ARRAY (lVelVyb -49; lVelVyb;[Faktury];
                         alFaRecN;[Zákazníci];alZakRecN)
```

3. Následující kód pracuje v postupných krocích po 1000 záznamech s výběrem který by nemohl být jako celek načten do array:

```
lMaxStr := 1000
lVelVyb := Records in selection ([Telefonni_Seznam])
For ($lStr ; 1; 1+(( lVelVyb -1)) )
      ` Načíst hodnoty a/nebo čísla záznamů
   SELECTION RANGE TO ARRAY (1+( lMaxStr*($lStr-1)); lMaxStr*$lStr;...;...;...;...;...;...)
      ` Provést něco s array
End for
```

Dále si přečtěte

AUTOMATIC RELATIONS, ON ERR CALL, SELECTION TO ARRAY.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ARRAY TO SELECTION (ARRAY DO VÝBĚRU)

Příkazy a odkazy pro Array

Verze 3

ARRAY TO SELECTION (array; pole {;array2; pole2; .....; arrayN;poleN})

Parametr	Typ		Popis
array	Array	→	Array které se bude kopírovat do výběru
pole	Pole	←	Pole do kterého se načtou data

Popis

Příkaz ARRAY TO SELECTION zkopíruje jeden nebo více array do výběru záznamů. Všechna pole musí patřit do jedné tabulky.

Jestliže výběr v době volání existuje, prvky array jsou předány do záznamů podle pořadí v array a pořadí záznamů. Pokud existuje více prvků array než záznamů, jsou vytvořeny nové záznamy. Záznamy, ať už nové nebo existující jsou automaticky uloženy.

Pokud jsou array různých velikostí, je počet kopírovaných prvků nastaven podle délky prvního array. Všechny další array jsou zkopírovány do polí které následují jejich název.

Tento příkaz je opakem příkazu SELECTION TO ARRAY. Příkaz ARRAY TO SELECTION však nemůže pracovat s různými tabulkami, včetně vztažených, dokonce i když existuje automatický vztah.

UPOZORNĚNÍ: Příkaz ARRAY TO SELECTION používejte pozorně, protože přepíše aktuální hodnoty v polích. Pokud je nějaký záznam zamčený jiným procesem během provádění příkazu ARRAY TO SELECTION tento záznam nebude změněn. Všechny zamčené záznamy jsou přesunuty do sady LockedSet. Po provedení příkazu ARRAY TO SELECTION můžete vyzkoušet tuto sadu, jestli byly změněny všechny záznamy.

4D Server: Příkaz je optimalizován pro 4D Server. Všechny array jsou poslány, je-li potřeba ze stroje klienta na server a záznamy jsou upraveny a vytvořeny na serveru. Protože je tento příkaz synchronní, musí klient počkat na dokončení akce. V prostředí více uživatelů nebo více procesů nebudou záznamy které jsou zamčeny upraveny.

Příklad

V následujícím příkladu budou do tabulky [Lidé] doplněna data z array asPřijmeni a asFirmy. Hodnoty z array asPřijmeni budou předána do pole [Lidé]Příjmení a hodnoty z array asFirmy budou předány do pole [Lidé]Firma:

```
ARRAY TO SELECTION (asPřijmeni;[Lidé]Příjmení;asFirmy;[Lidé]Firma)
```

Dále si přečtěte

SELECTION TO ARRAY.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DISTINCT VALUES (ODLIŠNÉ HODNOTY)

Příkazy a odkazy pro Array

Verze 6.0 (Upraveno)

DISTINCT VALUES (pole; array)

Parametr	Typ		Popis
pole	Pole nebo podpole	→	Pole nebo podpole s daty k vytvoření array
array	Array	←	Array k naplnění jedinečnými hodnotami z pole

Popis

Příkaz DISTINCT VALUES vytvoří a naplní array "*array*" neopakovanými (jedinečnými) hodnotami z pole "*pole*" ze záznamů v platném výběru tabulky, ke které pole nebo podpole patří.

Do DISTINCT VALUES můžete předat libovolné **indexovatelné** pole nebo podpole, t.j ta jež mohou být indexována, aniž by jste je museli indexovat.
Je však zřejmé, že použití na neindexované pole bude pomalejší. Je nutno si pamatovat, že v tomto případě bude rovněž ztracen ukazatel na platný záznam.

Poznámka: Nyní je možno používat tento příkaz na indexovaná i neindexovaná pole. Jeho způsob provádění je možno nastavit pomocí příkazu SET DATABASE PARAMETER.

Pokud předáte pole z tabulky, DISTINCT VALUES prohlédne a uloží jedinečné hodnoty pouze z platného výběru. Při použití tohoto příkazu na podpole, prohlédne se každý podzáznam každého záznamu z platného výběru záznamů.

Poznámka: Od verze 4D 6.5, když je DISTINCT VALUES voláno během **transakce**, a tato není ještě ukončena, budou záznamy vytvořené během transakce **vzaty do úvahy.**

Pokud nejdříve vytvoříte array, příkaz DISTINCT VALUES očekává, že jeho typ bude kompatibilní s polem nebo podpolem které předáte. Při nesouhlasu ve zkompilované verzi bude generována chyba. V nezkompilované verzi, DISTINCT VALUES změní typ array na array stejného typu. Pokud je pole nebo podpole typu Čas, příkaz vytvoří array LongInteger.

Po dokončení příkazu je array stejně velká jako počet jedinečných hodnot nalezených ve výběru. Příkaz nemění nikdy platný výběr ani platný záznam (je-li pole indexováno). Příkaz DISTINCT VALUES používá index pole a tak budou prvky array seřazeny ve vzestupném pořadí. Pokud je to pořadí které potřebujete, nemusíte již používat příkaz SORT ARRAY po provedení příkazu DISTINCT VALUES.

UPOZORNĚNÍ: DISTINCT VALUES může vytvořit velký array podle velikosti výběru a počtu jedinečných hodnot. Array je umístěno v paměti, proto je dobré testovat výsledek po dokončení příkazu. Proveďre tak, že otestujete velikost každého array nebo hlídejte provádění příkazu pomocí ON ERR CALL.

4D Server: Příkaz je optimalizován pro 4D Server. Každý array je vytvářen nebo hodnoty jsou vypočítány na serveru; array je potom poslán, jeli potřeba dle rozsahu platnosti, zpět na stroj klienta.

Příklady

1. Následující příklad vytvoří seznam měst z platného výběru a řekne uživateli počet měst ve kterých jsou vybrané firmy:

```
ALL RECORDS([Obchody]) ` Vytvoří výběr záznamů
DISTINCT VALUES([Obchody]Města;asMěsta)
ALERT("Firmy sídlí v " +String(Size of array(asMěsta))+" městech.")
```

2. Následující příklad uloží do asKlíče všechna klíčová slova která jsou přiřazená (s použitím podtabulky) k dokumentům 4D Write uloženým v tabulce [Dokumentace] a jejichž téma je "Ekonomie":

```
QUERY ([Dokumentace];[ Dokumentace]Téma="Ekonomie")
DISTINCT VALUES([Dokumentace]Klíče'KlíčováSlova;asKlíče)
```

Po té co byl tento array vytvořen jej můžete použít k nalezení všech dokumentů označených určitým klíčovým slovem:

```
QUERY ([Dokumentace];[ Dokumentace] Klíče'KlíčováSlova = asKlíče {asKlíče})
SELECTION TO ARRAY ([Dokumentace]Předmět;asPředmět)
   ` ...
```

Dále si přečtěte

ON ERR CALL, SELECTION TO ARRAY, SELECTION RANGE TO ARRAY, SET DATABASE PARAMETER.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

LONGINT ARRAY FROM SELECTION (LONGINT ARRAY Z VÝBĚRU)

Příkazy a odkazy pro Array

Verze 6.5

LONGINT ARRAY FROM SELECTION ({tabulka;} záznamArray {; výběr})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka platného výběru nebo platná tabulka pokud vynecháno
záznamArray	LongInt Array	←	Array čísel záznamů
výběr	Řetězec	→	Název pojmenovaného výběru nebo platný výběr, pokud vynecháno

Popis

Příkaz LONGINT ARRAY FROM SELECTION naplní array *záznamArray* čísly (absolutními) záznamů, které jsou ve výběru.

Pokud nepředáte parametr *výběr*, příkaz použije platný výběr tabulky. Pouze v tomto případě je potřeba použít parametr tabulka.

Poznámka: Prvek 0 array je nastaven na hodnotu -1.

Dále si přečtěte

CREATE SELECTION FROM ARRAY.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

BOOLEAN ARRAY FROM SET (LOGICKÉ ARRAY ZE SADY)

Verze 6.5

BOOLEAN ARRAY FROM SET (logickéArray {; sada})

Parametr	Typ		Popis
logickéArray	Logické Array	←	Array k označení jestli je záznam v sadě
sada	Řetězec	→	Název sady nebo UserSet pokud je vynecháno.

Popis

Příkaz BOOLEAN ARRAY FROM SET naplní array logických hodnot, které označují jestli záznam je nebo není v sadě. V array *logickéArray* je tolik prvků, kolik je záznamů v sadě. Prvky v sadě jsou setříděny v pořadí ve kterém byly záznamy do tabulky pořizovány.

Každý prvek array má hodotu:
- True, pokud odpovídající záznam je v sadě.
- False pokud odpovídající záznam není v sadě.

Pokud N je počet záznamů v tabulce, prvek 0 odpovídá záznamu 0, prvek 1 odpovídá záznamu 1, atd.

Pokud nepoužijete parametr *sada*, bude použita sada UserSet.

Dále si přečtěte

CREATE SET FROM ARRAY.

Příkazy a odkazy pro Array

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Příkazy BLOB

Příkazy a odkazy pro BLOB

Verze 6.0

Definice

4th Dimension v6 obsahuje nový datový typ BLOB (**B**inary **L**arge **Ob**jects):

Můžete definovat pole a proměnné BLOB:
• K vytvoření pole BLOB vyberte BLOB z rozevírací nabídky **Typ pole** v okně **Vlastnosti pole**.
• K vytvoření BLOB proměnné použijte příkaz **C_BLOB**. Můžete vytvořit místní, procesovou nebo meziprocesovou proměnnou BLOB.

Poznámka: Array typu BLOB neexistuje.

Ve 4th Dimension je BLOB postupná série proměnné délky, která může být brána jako jeden objekt nebo jejíž byty mohou být adresovány jednotlivě. BLOB může být prázdný (nulová délka) nebo může obsahovat kolem 2.147.483.647 bytů (2 GB).

BLOBy a paměť

BLOB může být načten do paměti jako celek. Proměnná BLOB je držena a existuje pouze v paměti. BLOB pole je načítáno do paměti při načtení záznamu, ke kterému patří.

Jako ostatní typy polí, které mohou obsahovat velký počet dat (obrázkové a podatbulka pole), nejsou BLOB duplikovány v paměti při úpravě dat a zavádění záznamu do paměti. Výsledek vracený příkazy **Old** a **Modified** nejsou směrodatné pokud jsou použity na BLOB pole.

Zobrazení BLOB

BLOB může obsahovat jakýkoli typ dat, protože nemá určené žádné výchozí zobrazení. Pokud zobrazíte ve formuláři BLOB pole nebo proměnnou, vždy se zobrazí prázdné, ať už mají jakýkoli obsah.

BLOB pole

BLOB pole můžete použít k uložení všech typů dat do velikosti 2 GB. Na BLOB pole nemůžete použít indexování a tak musíte použít výraz k vyhledávání podle hodnot uložených v BLOB poli. BLOB pole není dobré používat pro hodnoty, které chcete vybavovat rychle. Například neukládejte klíčová slova do BLOB polí; na tento účel použijte podtabulku, ve které můžete indexovat podpole klíčových slov.

Předávání parametrů, ukazatele a výsledky funkcí

BLOB ve 4th Dimension může být předán jako parametr příkazům 4D nebo rutinám které povolují BLOB parametry. Na druhou stránku nemohou být použity jako parametr v metodě uživatele. BLOB nemůže být vrácen jako výsledek funkce.

K předání BLOB vaší vlastní metodě, definujte ukazatel na BLOB a přiřaďte tento ukazatel jako parametr.

Příklad:

```
   ` Vytvoření proměnné typu BLOB
C_BLOB (anyBlobVar)
   ` BLOB je předán jako parametr k příkazu 4D
SET BLOB SIZE (anyBlobVar;1024*1024)
```

```
    ` BLOB je předán jako parametr k externí rutině
$errCode:= Udělat něco s   BLOB (anyBlobVar)
    ` Ukazatel na BLOB je předán jako paramer k metodě uživatele
COMPUTE BLOB (→anyBlobVar )
    ` vytvoření proměnné typu Ukazatel
C_POINTER (aPointer)
    ` Definice ukazateli k BLOB
aPointer := →anyBlobVar
    ` Ukazatel na BLOB je předán jako parametr k metodě uživatele
COMPUTE BLOB (aPointer)
```

Poznámka pro vývojáře 4D Extension

BLOB parametr je vytvořen "&O" (písmeno "O" ne číslice "0").

Přiřazení

Jeden BLOB můžete přiřadit k jinému.

Příklad:

```
    ` Vytvoření dvou proměnných typu BLOB
C_BLOB (vBlobA;vBlobB)
    ` Nastavení velikosti prvního BLOB na 10K
SET BLOB SIZE (vBlobA;10*1024)
    ` Přiřazení prvního BLOB k druhému
vBlobB:=vBlobA
```

Žádný operátor nemůže být použit na BLOBy; neexistuje výraz typu BLOB.

Adresování obsahu BLOB

Každý byte v BLOBu můžete adresovat jednotlivě s použitím složených závorek {....} Uvnitř BLOB jsou byty číslovány od 0 do N-1 kde N je velikost BLOB.

Příklad:

```
    ` Vytvoření proměnné typu BLOB
C_BLOB (vBlob)
    ` Nastavení velikosti BLOB na 256 bytů
SET BLOB SIZE (vBlob;256)
    ` Smyčka která nastaví všech 256 bytů BLOBu na nula
For ( vByte ; 0 ; BLOB size (vBlob)-1)
   vBlobvByte:=0
End for
```

Protože můžete jednotlivé byty BLOBu adresovat jednotlivě, můžete do něj ukládat jakoukoli proměnnou nebo pole.

Příkazy pro BLOB ve 4th Dimension

4th Dimension obsahuje následující příkazy pro práci s BLOB:

• SET BLOB SIZE změní velikost BLOBu

• BLOBsize navrátí velikost BLOBu.

• DOCUMENT TO BLOB a BLOB TO DOCUMENT vám umožní načíst a zapsat jakéhokoli dokument z nebo do BLOBu (volitelně do dat a zdrojů (resource) na Macintoshi).

• VARIABLE TO BLOB a BLOB TO VARIABLE stejně jako LIST TO BLOB a BLOB to list vám umožní uložit a vybavit proměnné 4D do/z BLOB.

• COMPRESS BLOB, EXPAND BLOB a BLOB PROPERTIES vám umožní pracovat s komprimovanými BLOBy.

• Příkazy BLOB to integer, BLOB to longint, BLOB to real, BLOB to text, INTEGER TO BLOB, LONGINT TO BLOB, REAL TO BLOB a TEXT TO BLOB vám umožní pracovat s jakkoli řazenými daty z disku, zdrojů, OS, atd.

• DELETE FROM BLOB, INSERT IN BLOB a COPY BLOB umožní rychlé ovládání velkých částí dat v BLOBu.

Tyto příkazy jsou popsány v této kapitole.

Dále:

• C_BLOB vytvoří proměnnou typu BLOB. Více informací najdete v kapitole Compiler.

• GET CLIPBOARD a APPEND TO CLIPBOARD umožní práci s jakýmikoliv daty ve Schránce. V kapitole Schránka najdete více informací.

• GET RESOURCE a SET RESOURCE umožní práci s daty uloženými ve zdrojích. V kapitole Zdroje najdete více informací.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET BLOB SIZE (NASTAVIT VELIKOST BLOB)

Příkazy a odkazy pro BLOB

Verze 6.0

SET BLOB SIZE (blob; velikost {; vyplnit})

Parametr	Typ		Popis
blob	BLOB	→	BLOB pole nebo proměnná
velikost	Číslo	→	Velikost BLOBu
vyplnit	Číslo	→	ASCII kód znaku k vyplnění

Popis

SET BLOB SIZE změní velikost BLOBu podle zadaného parametru.

Jako výchozí jsou nově přidané byty (pokud nějaké) do BLOBu nastaveny na hodnotu 0x00. Pokud chcete mít tyto byty nastaveny na jinou hodnotu, předejte tuto hodnotu (0...255) volitelným parametrem *vyplnit*.

Příklady

1. Jestliže jste ukončili práci s procesním nebo meziprocesním BLOBem, je dobré uvolnit paměť kterou používal. Uděláte to následovně:

```
SET BLOB SIZE(aProcessBLOB;0)
SET BLOB SIZE(<>anInterprocessBLOB;0)
```

2. Následující příklad vytvoří BLOB velikosti 16K vyplněný 0xFF:

```
C_BLOB(vxData)
SET BLOB SIZE(vxData;16*1024;0xFF)
```

Dále si přečtěte

BLOB size.

Ovládání chyb

Jestliže nemůžete upravit velikost BLOB kvůli nedostatku paměti, objeví se chyba -108. Můžete tuto chybu zachytit pomocí ON ERR CALL.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

BLOB size (Velikost BLOB)

Verze 6.0

BLOB size (blob) → Číslo

Parametr	Typ		Popis
blob	BLOB	→	BLOB pole nebo proměnná
Výsledek funkce	Číslo	←	Velikost BLOBu v bytech

Popis

BLOB size vrací velikost BLOBu v bytech.

Příklady

Tento kód vloží 100 bytů do BLOBu myBlob:

```
SET BLOB SIZE (BLOB size(myBlob)+100)
```

Dále si přečtěte

SET BLOB SIZE.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

COMPRESS BLOB (ZABALIT BLOB)

Příkazy a odkazy pro BLOB

Verze 6.0

COMPRESS BLOB (blob {; komprese})

Parametr	Typ		Popis
blob	BLOB	→	BLOB ke kompresi
komprese	Číslo	→	Pokud není vynechán: 1, komprimovat tolik kolik je to možné 2, komprimovat tak rychle jak je možné

Popis

Příkaz COMPRESS BLOB zkomprimuje BLOB *blob* s použitím vnitřního algoritmu 4D. Tento příkaz komprimuje pouze BLOB který je větší než 255 bytů.

Volitený parametr komprese vám umožňuje zvolit způsob jak bude BLOB komprimován:
• Pokud předáte 1, bude BLOB komprimován tolik, kolik je to možné, bez ohledu na rychlosti kompresních a dekompresních operací.
• Pokud předáte 2, bude BLOB komprimován tak rychle, jak je to možné (a bude rozbalen tak rychle jak to bude možné), bez ohledu na velikost jaká vznikne (zabalený BLOB bude větší).
• Pokud předáte jiné číslo nebo tento parametr vynecháte, bude BLOB komprimován tak jak je to možné s použitím módu 1.

4th Dimension nabízí následující předdefinované konstanty:

Konstanta	*Typ*	*Hodnota*
Silná komprese	*Long Integer*	*1*
Rychlá komprese	*Long Integer*	*2*

Po volání je nastavena proměnná OK na 1, pokud byl BLOB úspěšně komprimován. Pokud tato akce nemůže být provedena, je nastavena systémová proměnná OK na 0; například pokud není dost paměti.

Po té, co byl BLOB zabalen, jej můžete rozbalit příkazem EXPAND BLOB.

Aby jste zjistili jestli je BLOB zabalený, použijte příkaz BLOB PROPERTIES.

UPOZORNĚNÍ: Zabalený BLOB je stále BLOB a nic vám nebrání měnit jeho obsah. Ale pokud to uděláte, nebude příkaz EXPAND BLOB schopen BLOB rozbalit.

Příklady

1. Tento příklad testuje, jestli je BLOB vxMyBlob zabalený, a jestli ne, tak jej zabalí:

```
BLOB PROPERTIES (vxMyBlob;$vlCompressed;$vlExpandedSize;$vlCurrentSize)
If ($vlCompressed=Is notcompressed)
   COMPRESS BLOB (vxMyBlob)
End if
```

Zapamatujte si, že pokud použijete příkaz COMPRESS BLOB na zabalený BLOB, příkaz to pozná a neprovede nic.

2. Tento příklad vám umožní vybrat dokument a pak jej zabalit:

```
$vhDokumRef := Open document ("")
If (OK=1)
   CLOSE DOCUMENT ($vhDokumRef)
   DOCUMENT TO BLOB (Document;vxBlob)
   If (OK=1)
      COMPRESS BLOB (vxBlob)
      If (OK=1)
         BLOB TO DOCUMENT (Document;vxBlob)
      End if
   End if
End if
```

Dále si přečtěte

BLOB PROPERTIES, EXPAND BLOB.

Systémové proměné nebo Sady

Systémová proměnná OK je nastavena na 1, pokud byl BLOB úspěšně zabalen; pokud ne je nastavena na 0.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

EXPAND BLOB (ROZBALIT BLOB)

Příkazy a odkazy pro BLOB

Verze 6.0

EXPAND BLOB(blob)

Parametr	Typ		Popis
blob	BLOB	→	BLOB k rozbalení

Popis

Příkaz EXPAND BLOB rozbalí BLOB *blob*, který byl zabalen pomocí příkazu COMPRESS BLOB.

Po volání je proměnná OK nastavena na 1, pokud byl BLOB zabalen (nebo nebyl původně vůbec zabalen). Jestliže se rozbalení nepovede, je OK nastavena na 0; například pokud není dost paměti.

Aby jste zjistili jestli byl BLOB zabalený použijte příkaz BLOB PROPERTIES.

Příklady

1. Tento příklad zkontroluje jestli je BLOB vxMyBlob zabalený a pokud ano, rozbalí jej:

```
BLOB PROPERTIES (vxMyBlob;$vlCompressed;$vlExpandedSize;$vlCurrentSize)
If ($vlCompressed#Is not compressed)
   EXPAND BLOB (vxMyBlob)
End if
```

Pokud použijete příkaz EXPAND BLOB na BLOB který není zabalený, příkaz to pozná a neprovede nic.

2. Tento příklad vám umožní označit dokument a rozbalit jej pokud je zabalený:

```
$vhDokumRef := Open document ("")
If (OK=1)
   CLOSE DOCUMENT ($vhDokumRef)
   DOCUMENT TO BLOB (Document;vxBlob)
   If (OK=1)
      BLOB PROPERTIES (vxBlob;$vlCompressed;$vlExpandedSize;$vlCurrentSize)
      If ($vlCompressed#Is not compressed)
         EXPAND BLOB (vxBlob)
         If (OK=1)
            BLOB TO DOCUMENT (Document;vxBlob)
         End if
      End if
   End if
```

End if

Dále si přečtěte

BLOB PROPERTIES, COMPRESS BLOB.

Systémové proměnné a Sady

Systémová proměnná OK je nastavena na 1 pokud byl BLOB správně rozbalen a na 0 pokud operace nebyla dokončena.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

BLOB PROPERTIES (VLASTNOSTI BLOB)

Příkazy a odkazy pro BLOB

Verze 6.0

BLOB PROPERTIES (blob; zabaleno {; rozbalenáVel {; AktVelikost}})

Parametr	Typ		Popis
blob	BLOB	→	BLOB o kterém získate informace
zabaleno	Číslo	←	0 = BLOB není zabalený
			1 = BLOB je zabalený kompaktně
			2 = BLOB je zabalený rychle
rozbalenáVel	Číslo	←	Velikost BLOBu (v bytech) pokud není zabalený
AktVelikost	Číslo	←	Aktuální velikost BLOBu (v bytech)

Popis

Příkaz BLOB PROPERTIES vrací informace o BLOBu blob.

• Parametr *zabaleno* vám řekne, jestli a jak je BLOB zabalený a vrátí jednu z následujících hodnot.

Poznámka: 4D nabízí předdefinované konstanty:

Konstanta	*Typ*	*Hodnota*
Není zabalený	*Long Integer*	*0*
Zabalený kompaktně	*Long Integer*	*1*
Zabalený rychle	*Long Integer*	*2*

• Ať už je metoda zabalení jakákoli, do parametru *rozbalenáVel* je vždy dosazena velikost BLOBu před zabalením.
• Parametr *AktVelikost* vrací aktuální velikost BLOBu. Pokud je BLOB zabalený bude většinou AktVelikost menší než rozbalenáVel. Pokud není BLOB zabalený, bude hodnota těchto parametrů shodná.

Příklady

1. Podívejte se na příklady u příkazů COMPRESS BLOB a EXPAND BLOB.

2. Po té co byl BLOB zabalený, následující metoda projektu zjistí procenta místa ušetřeného zabalením:

```
` Metoda projektu Místo ušetřené zabalením
` Místo ušetřené zabalením (Ukazatel { ; Ukazatel } ) → Délka
` Místo ušetřené zabalením ( → BLOB {; → ušetřenéByty } ) → Procenta
C_POINTER ($1;$2)
C_LONGINT ($0;$vlCompressed;$vlExpandedSize;$vlCurrentSize)
BLOB PROPERTIES ($1→;$vlCompressed;$vlExpandedSize;$vlCurrentSize)
If ($vlExpandedSize=0)
   $0:=0
   If (Count parameters>=2)
      $2→:=0
   End if
Else
   $0:=100-(($vlCurrentSize/$vlExpandedSize)*100)
   If (Count parameters>=2)
      $2→:=$vlExpandedSize-$vlCurrentSize
   End if
End if
```

Po uložení této metody do vaší aplikace ji můžete použít následovně:

```
` ...
COMPRESS BLOB (vxBlob)
$vlPercent:= Místo ušetřené zabalením (→vxBlob;→vlBlobSize)
ALERT ("Zabalení ušetřilo "+String (vlBlobSize)+" bytů, "+String ($vlPercent;"#0%")+" místa.")
```

Dále si přečtěte

COMPRESS BLOB, EXPAND BLOB.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DOCUMENT TO BLOB (DOKUMENT DO BLOBU)

Příkazy a odkazy pro BLOB

Verze 6.0

DOCUMENT TO BLOB (dokument; blob {; *})

Parametr	Typ		Popis
dokument	Řetězec	→	Název dokumentu
blob	BLOB	→	Proměnná nebo pole BLOB k přijetí obsahu dokumentu
		←	Dokument v BLOBu
*	*	→	Pouze na Macintoshi: Zdrojová část je načtená pokud je předáno * a pokud ne je načten Datová část.

Popis

DOCUMENT TO BLOB načte čistý obsah dokumentu do BLOBu. Musíte vložit název existujícího dokumentu který není ještě otevřený, jinak se objeví chyba. Aby jste umožnili uživateli načíst dokument do BLOB, použijte příkaz Open document a proměnnou procesu odkazu na dokument (podívejte se na Příklad).

Poznámka pro Macintosh: Na Macintoshi mohou být dokumenty složeny ze dvou částí: Data a Zdroje. Jako výchozí načte příkaz Open document Datovou část dokumentu. Pro načtení Zdrojů, použijte volitelný parametr *. Na Windows je tento parametr ignorován. Všimněte si, že prostředí 4D nabízí obdobu části souboru Zdroje MacOs na Windows. Například u struktury jsou Data uložena v souboru s koncovkou .4DB; zdroje jsou uloženy v souboru se stejným názvem ale koncovkou .RSR. Pokud na Windows vytváříte alikaci s částí dat a zdrojů uloženou v BLOBech, potřebujete pouze přístup k souborům korespondujícím s částí se kterou pracujete.

Příklad

Píšete Informační systém který umožní rychlé načítání a ukládání dokumentů. Ve vstupním formuláři vytvoříte tlačítko které umožní načíst dokument do BLOBu. Metoda pro tlačítko může vypadat takto:

```
$vhDokumRef:=Open document("") ` Vyberte dokument
If (OK=1) ` Pokud byl dokument vybrán
   CLOSE DOCUMENT($vhDokumRef) ` Nepotřebujeme jej mít otevřený
   DOCUMENT TO BLOB (Document;[VašeTabulka]PoleBLOB)
   If (OK=0)
      ` Zachycení chyby
   End if
End if
```

Dále si přečtěte

BLOB TO DOCUMENT, Open document.

Systémové proměnné

Proměnná OK je nastavena na 1, pokud byl dokument správně načten, jinak je nastavena na 0 a je generována chyba.

Ovládání chyb

• Pokud se pokusíte načíst (do BLOBu) dokument, který neexistuje, nebo je otevřený, je generována chyba File Manageru.
• I/O chyba je generována pokud je dokument zamčen, umístěn na zamčeném disku a nebo jsou problémy s jeho načtením.

• Pokud není dostatek paměti k načtení dokumentu je genegována chyba -108.
V každém případě můžete chyby zachytit pomocí metody ON ERR CALL.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

BLOB TO DOCUMENT (BLOB DO DOKUMENTU)

Příkazy a odkazy pro BLOB

Verze 6.0

BLOB TO DOCUMENT (dokument; blob {;*})

Parametr	Typ		Popis
dokument	Řetězec	→	Název dokumentu
blob	BLOB	→	Nový obsah dokumentu
*	*	→	Pouze na Macintoshi Je zapsán Zdroj pokud je předána * jinak jsou zapsána Data.

Popis

BLOB TO DOCUMENT přepíše celý obsah dokumentu *dokument* daty z blobu. Musíte zadat název dokumentu který existuje a není otevřený, jinak vznikne chyba. Pokud chcete nechat uživatele vybrat svůj vlastní dokument, použijte příkazy Open document nebo Create document a procesovou proměnnou odkazu na dokument (přečtěte si Příklad).

Poznámka pro Macintosh: Na Macintoshi mohou být dokumenty složeny ze dvou částí: Data a Zdroje. Jako výchozí nastavení příkaz BLOB TO DOCUMENT přepíše Datovou část dokumentu. Pro přepsání Zdrojů, použijte volitelný parametr *. Na Windows je tento parametr ignorován. Všimněte si, že prostředí 4D obsahuje obdobu části souboru Zdroje MacOs i na Windows. Například pro strukturu část Data je uložena v souboru s koncovkou .4DB; zdroje jsou uloženy v souboru se stejným názvem ale koncovkou .RSR. Pokud na Windows vytváříte aplikaci s ukládáním dat a zdrojů v BLOBech, potřebujete pouze přístup k souboru korespondujícím s částí se kterou pracujete.

Příklad

Píšete Informační systém který umožní rychlé načítání a ukládání dokumentů. Ve vstupním formuláři vytvoříte tlačítko které umožní načíst dokument z BLOBu. Metoda pro tlačítko může vypadat takto:

```
$vhDokumRef:=Create document("") ` Uložit dokument
If (OK=1) ` Pokud byl dokument vytvořen
   CLOSE DOCUMENT($vhDokumRef) ` Nepotřebujeme jej nechat otevřený
   BLOB TO DOCUMENT (Document;[VašeTabulka]PoleBLOB)
   If (OK=0)
      ` Zachytit chyby
   End if
End if
```

Dále si přečtěte

Create document, DOCUMENT TO BLOB, Open document.

Systémové proměnné

OK je nastavena na 1 pokud bylo do dokumentu správně zapsáno, pokud ne je OK nastavena na 0 a je generována chyba.

Ovládání chyb

- Pokud se pokusíte zapsat do dokumentu, který neexistuje nebo je již otevřen jinou aplikací nebo procesem, objeví se chyba File Manageru.
- Disk může být plný pro zapsání dokumentu.
- I/O chyby se mohou objevit při psaní do dokumentu.

Ve všech případech můžete chyby zachytit pomocí metody ON ERR CALL.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

VARIABLE TO BLOB (PROMĚNNÁ DO BLOBU)

Příkazy a odkazy pro BLOB

Verze 6.0

VARIABLE TO BLOB (proměnná; blob {; posun|*})

Parametr	Typ		Popis
proměnná	Proměnná	→	Proměnná k uložení do BLOBu
blob	BLOB	→	BLOB do kterého se proměnná uloží
posun|*	Znak číslo	→	* k přidání hodnoty do BLOB nebo posun uvnitř BLOB v bytech
		←	nový posun, je-li uveden, není-li uvedena *

Popis

Příkaz VARIABLE TO BLOB uloží proměnnou *proměnná* do BLOBu *blob*.

Pokud zadáte volitelný parametr *, proměnná se připojí nakonec k BLOBu a jeho velikost přiměřeně vzroste. S použitím tohoto parametru můžete postupně přidávat do BLOBu čísla proměnných nebo seznamy (přečtěte si další příkazy pro BLOB) tak dlouho, dokud se BLOB vejde do paměti.

Pokud nepoužijete ani volitelný parametr *, ani proměnný parametr *posun*, bude proměnná přidána na začátek BLOBu a přepíše jeho obsah; velikost se podle toho změní.

Pokud předáte proměnný parametr *posun*, bude proměnná zapsána do BLOBu začínaje od bytu *posun* (počátek BLOB je nula) a existující byty budou přepsány. Velikost BLOBu se patřičně změní podle toho kam zapíšete proměnnou tj.podle umístění proměnné (pokud je to za současný konec posun plus velikost proměnné *proměnná*). Nově předáné byty (např. před začátkem zápisu), na rozdíl od těch které zapisujete, jsou nastaveny na nula.

Po provedení příkazu je proměnná *posun* vrácena, zvětšená o množství bytů které jste zapsali. Proto můžete znovu použít stejnou proměnnou *posun* s jiným příkazem zápisu do BLOBu (např. VARIABLE TO BLOB) k zapsání jiné proměnné nebo zapsání seznamu.

VARIABLE TO BLOB přijímá všechny typy proměnných (včetně BLOBu) až na tyto výjimky:
- Ukazatele
- Array ukazatelů
- Dvojrozměrné array

Uložíte-li Long Integer, který je odkaz na hierarchický seznam (ListRef). VARIABLE TO BLOB uloží proměnnou Long Integer a ne seznam. K uložení a načtení hierarchického seznamu použijte příkazy LIST TO BLOB a BLOB to list.

UPOZORNĚNÍ: Pokud použijete BLOB k uložení proměnných musíte později použít příkaz BLOB TO VARIABLE pro přečtení obsahu BLOBu, protože proměnné jsou uloženy v BLOBu s použitím vnitřního formátu 4D.

Po dokončení příkazu, a pokud byla proměnná správně uložena, je proměnná OK nastavena na 1. Pokud nemohla být operace provedena, je OK nastavena na 0; například nebyl dostatek paměti.

Poznámka k nezávislosti na Platformě

VARIABLE TO BLOB a BLOB TO VARIABLE používají vnitřní formát 4D k ovládání proměnných uložených v BLOBu. Z toho důvodu se nemusíte starat o přenášení mezi platformami. Jinými slovy BLOB vytvořený na Windows s použitím těchto příkazů, je použitelný i na Macintosh a naopak.

Příklady

1. Dvě následující metody projektu vám umožní rychlé uložení a načtení array z dokumentu na disku:

```
    ` Metoda projektu ULOŽIT ARRAY
    ` ULOŽIT ARRAY ( řetězec ; Ukazatel )
    ` ULOŽIT ARRAY ( Dokument ; → Array )
C_STRING (255;$1)
C_POINTER ($2)
C_BLOB ($vxArrayData)
VARIABLE TO BLOB ($2→;$vxArrayData) ` Uložit array do BLOBu
COMPRESS BLOB ($vxArrayData) ` Zabalit BLOB
BLOB TO DOCUMENT ($1;$vxArrayData) ` Uložit BLOB na disk
    ` Metoda projektu NAČÍST ARRAY
    ` NAČÍST ARRAY ( Řetězec ; Ukazatel )
    ` NAČÍST ARRAY ( Dokument ; → Array )
C_STRING (255;$1)
C_POINTER ($2)
C_BLOB ($vxArrayData)
DOCUMENT TO BLOB ($1;$vxArrayData) ` Načíst BLOB z disku
EXPAND BLOB ($vxArrayData) ` Rozbalit BLOB
BLOB TO VARIABLE ($vxArrayData;$2→) ` Načíst array z BLOBu
```

Po té, co jste vložili tyto metody do vaší aplikace, můžete napsat:

```
ARRAY STRING (...;asArray;...)
    ` ...
ULOŽIT ARRAY ( $vsNazev;→asAnyArray)
    ` ...
NAČÍST ARRAY ( $vsNazev;→asAnyArray)
```

2. Následjící dvě metody projektu vám umožní uložit a načíst sadu proměnných do BLOBu:

```
` Metoda projektu ULOŽIT PROM DO BLOB
` ULOŽIT PROM DO BLOB ( Ukazatel   {; Ukazatel ...   {; Ukazetel }}   )
` ULOŽIT PROM DO BLOB ( BLOB   {; Prom1 ...   {; Prom2}}   )
C_POINTER (${1})
C_LONGINT ($vlParam)
SET BLOB SIZE ($1→;0)
For ($vlParam;2;Count parameters)
   VARIABLE TO BLOB (${$vlParam}→;$1→;*)
End for

   ` Metoda projektu NAČÍST PROM Z BLOB
   ` NAČÍST PROM Z BLOB ( Ukazatel {; Ukazatel... {; Ukazatel}})
   ` NAČÍST PROM Z BLOB ( BLOB   {; Prom1 ...   {; Prom2 }}   )
C_POINTER (${1})
C_LONGINT ($vlParam;$vlPosun)
$vlPosun:=0
For ($vlParam;2;Count parameters)
   BLOB TO VARIABLE ($1→;${$vlParam}→;$vlPosun)
End for
```

Po předání těchto dvou metod do vaší aplikace můžete napsat:

ULOŽIT PROM DO BLOB (→vxBLOB;→vgPicture;→asAnArray;→alAnotherArray)
` ...
NAČÍST PROM Z BLOB (→vxBLOB;→vgPicture;→asAnArray;→alAnotherArray)

Dále si přečtěte

BLOB to list, BLOB TO VARIABLE, LIST TO BLOB.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

BLOB TO VARIABLE (BLOB DO PROMĚNNÉ)

Verze 6.0

BLOB TO VARIABLE (blob; proměnná {; posun})

Parametr	Typ		Popis
blob	BLOB	→	BLOB obsahující proměnné 4D
proměnná	Proměnná	→	Proměnná k zapsání z BLOBu
posun	Číslo	→	Umístění proměnné v BLOB
		←	Umístění následující proměnné v BLOB

Popis

Příkaz BLOB TO VARIABLE přepíše obsah proměnné *proměnná* z BLOBu blob začínající na bytu posun (číslování začíná od nuly) definovaného parametrem *posun*.

Data uložená v předchozím do BLOBu musí být konzistentní s cílovou proměnnou. Většinou budete používat BLOB který jste naplnili s použitím příkazu VARIABLE TO BLOB.

Pokud nepoužijete volitelný parametr posun, bude se proměnná číst ze začátku BLOBu. Pokud pracujete s BLOBem ve kterém je uloženo více proměnných, musíte použít parametr *posun* a navíc vložit číselnou proměnnou. Před provedením příkazu nastavte tuto číselnou proměnnou na patřičný posun. Po provedení příkazu vrátí ta samá proměnná umístění další proměnné v BLOBu.

Po provedení příkazu a pokud byla proměnná správně přepsána, je nastavena proměnná OK na 1. Pokud nemohla být operace správně provedena je OK nastavena na 0; například při nedostatku paměti.

Poznámka k nezávislosti na Platformě

BLOB TO VARIABLE a VARIABLE TO BLOB používají vnitřní formát 4D k ovládání proměnných uložených v BLOBu. Z toho důvodu se nemusíte bát o přenos mezi platformami. Jinými slovy BLOB vytvořený na Windows s použitím těchto příkazů, je použitelný i na Macintoshi a naopak.

Příklad

Podívejte se na příklady u příkazu VARIABLE TO BLOB.

Dále si přečtěte

VARIABLE TO BLOB.

Systémové proměnné a Sady

Proměnná OK je nastavena na 1, pokud byla proměnná správně přepsána, pokud ne je nastavena na 0.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

LIST TO BLOB (SEZNAM DO BLOBU)

Příkazy a odkazy pro BLOB

Verze 6.0

LIST TO BLOB (seznam; blob {; *})

Parametr	Typ		Popis
seznam	ListRef	→	Hierarchcký seznam k uložení do BLOBu
blob	BLOB	→	BLOB k uložení Hierarchického seznamu
*	*	→	* k připojení hodnoty do BLOB

Popis

Příkaz LIST TO BLOB uloží hierarchický seznam adresovaný odkazem *seznam* do BLOBu *blob*.

Pokud nastavíte volitelný parametr * připojí se hierarchický seznam na konec BLOBu a jeho velikost se patřičně zvětší. S použitím parametru * můžete do BLOBu postupně uložit tolik proměnných nebo seznamů (přečtěte si další příkazy k BLOBu) dokud se BLOB vejde do paměti.

Pokud nepoužijete parametr * bude hierarchický seznam předán na začátek BLOBu a přepíše jeho obsah; velikost se patřičně upraví.

UPOZORNĚNÍ: Pokud do BLOBu ukládáte seznamy, musíte k jejich zpětnému načtení použít příkaz BLOB to list, protože seznamy jsou v BLOBu uloženy podle vnitřního formátu 4D.

Po provedení příkazu, pokud byl seznam správně uložen, je proměnná OK nastavena na 1. Jestliže nemohla být operace provedena, je OK nastavena na 0; například pokud není dostatek paměti.

Poznámka k nezávislosti na Platformě

LIST TO BLOB a BLOB to list používají vnitřní formát 4D k ovládání proměnných uložených v BLOBu. Z toho důvodu se nemusíte bát o přenos mezi platformami. Jinými slovy BLOB vytvořený na Windows s použitím těchto příkazů, je použitelný i na Macintosh a naopak.

Příklady

Přečtěte si příklady k příkazu BLOB to list.

Dále si přečtěte

BLOB to list, BLOB TO VARIABLE, VARIABLE TO BLOB.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

BLOB to list (BLOB do seznamu)

Příkazy a odkazy pro BLOB

Verze 6.0

BLOB to list (blob {; posun}) → OdkSeznam

Parametr	Typ		Popis
blob	BLOB	→	BLOB obsahující hierarchický seznam
posun	Číslo	→	Posun v BLOBu (v bytech)
		←	Nový posun po přečtení (posun+délka)
Výsledek funkce	OdkSeznam	←	Odkaz na nově vytvořený seznam

Popis

Příkaz BLOB to list vytvoří nový hierarchický seznam s obsahem z BLOBu *blob* počínající na bytu *posun* (číslování začíná na nule) definovaného parametrem *posun* a vrací *OdkSeznam,* číslo odkazu na nový seznam.

Data v BLOBu musí pro tento příkaz být konzistentní se seznmem. Většinou použijete příkaz LIST TO BLOB k naplnění BLOBu.

Pokud nedefinujete parametr *posun*, budou se data číst od začátku BLOBu. Pokud pracujete s BLOBem, ve kterém je uloženo více proměnných nebo seznamů, musíte použít pro parametr *posun* číselnou proměnnou a naplnit ji číselnou hodnotou posunu. Před provedením příkazu nastavte tuto číselnou proměnnou na patřičný posun. Po provedení příkazu vrátí ta samá proměnná posun (umístění) další proměnné či seznamu v BLOBu (Zvětší se o délku bytů seznamu v BLOB).

Po provedení příkazu, pokud byl seznam správně vytvořen, je proměnná OK nastavena na 1. Jestliže nemohla být operace provedena, je OK nastavena na 0; například pokud není dostatek paměti.

Poznámka k nezávislosti na Platformě

BLOB to list a LIST TO BLOB používají vnitřní formát 4D k ovládání proměnných uložených v BLOBu. Z toho důvodu se nemusíte bát o přenos mezi platformami. Jinými slovy BLOB vytvořený na Windows s použitím těchto příkazů, je použitelný i na Macintosh a naopak.

Příklad

V tomto příkladu při otevření formuláře načte metoda formuláře pro vstup dat seznam z BLOBu a při uzavření formuláře jej opět do BLOBu uloží:

```
      ` Metoda formuláře [Věc k provedení];"Vstup"
   Case of
      ÷ (Form event=On Load)
         hList:=BLOB to list([Věc k provedení]Další šílené nápady)
         If (OK=0)
            hList:=New list
         End if
      ÷ (Form event=On Unload)
         CLEAR LIST(hList;*)
      ÷ (bValidate=1)
         LIST TO BLOB(hList;[Věc k provedení]Další šílené nápady)
   End case
```

Dále si přečtěte

LIST TO BLOB.

Systémové proměnné a Sady

Proměnná OK je nastavena na 1 pokud byl seznam správně vytvořen, pokud ne je nastavena na 0.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

INTEGER TO BLOB (INTEGER DO BLOBU)

Příkazy a odkazy pro BLOB

Verze 6.0

INTEGER TO BLOB (integer; blob; PořadíBytů {; posun | *})

Parametr	Typ		Popis
Integer	Číslo	→	Hodnota Integeru pro zapsání do BLOBu
blob	BLOB	→	BLOB k zapsání Integeru
PořadíBytů	Číslo	→	0 Přirozené pořadí dle stroje
			1 Pořadí Macintosh
			2 Pořadí Windows
posun | *	Číslo | *	→	Posun začátku zápisu nebo * pro zápis na konec
		←	Nový posun, je-li uveden, a jestliže není *

Popis

Příkaz INTEGER TO BLOB zapíše 2 byty hodnoty proměnné *Integer* do BLOBu *blob*.

Parametr *PořadíBytů* stanoví pařadí zapsání jednotlivých bytů proměnné *Integer*. Musíte předat jednu z předdefinovaných konstant 4D (dle patformy, Mac, PC pořadí):

Konstanta	***Typ***	***Hodnota***
Native byte ordering	*Long Integer*	*0*
Macintosh byte ordering	*Long Integer*	*1*
PC byte ordering	*Long Integer*	*2*

Poznámka k nezávislosti na Platformě

Pokud používáte BLOB mezi platformami Macintosh a PC, je důležité při použití tohoto příkazu pořadí bytů řídit.

Pokud použijete parametr * je hodnota z proměnné *Integer* přidána na konec BLOBu a jeho velikost patřičně vzroste. S použitím tohoto parametru můžete postupně přidávat tolik hodnot Integer, Long Integer, Real nebo Text (přečtěte si další příkazy k BLOBu) dokud se BLOB vejde do paměti.

Pokud nepřidáte parametr * nebo *posun*, zapíše se 2 bytová hodnota Integer na začátek BLOBu a přepíše jeho obsah; velikost se patřičně změní či nezmění.

Pokud předáte proměnný parametr *posun*, 2-bytová hodnota Integer se v BLOBu zapíše od pozice posun (číslování začíná od nuly). Velikost BLOBu se patřičně změní podle toho kam zapíšete proměnnou tj.podle umístění dle proměnné *posun* (pokud je to za současný konec velikost bude *posun* plus 2 byty). Nově předáné byty (např. před začátkem zápisu), jsou nastaveny na nula.

Po provedení příkazu je navrácena proměnná *posun* zvětšená o počet bytů které jste přidali. Proto můžete použít tuto proměnnou dále s jiným příkazem pro BLOB k zapsání jiné hodnoty.

Příklady

1. Po spuštění tohoto kódu:

```
INTEGER TO BLOB (0x0206;vxBlob;Native byte ordering)
```

- Velikost vxBlob je 2 byty
- Na Macintoshi vxBlob{0} = $02 a vxBlob{1} = $06
- Na PC vxBlob{0} = $06 a vxBlob{1} = $02

2. Po spuštění tohoto kódu:

```
INTEGER TO BLOB (0x0206;vxBlob;Macintosh byte ordering)
```

• Velikost vxBlob je 2 byty
• Na všech platformách je vxBlob{0} = $02 a vxBlob{1} = $06

3. Po spuštění tohoto kódu:

```
INTEGER TO BLOB (0x0206;vxBlob;PC byte ordering)
```

• Velikost vxBlob je 2 byty
• Na všech platformách je vxBlob{0} = $06 a vxBlob{1} = $02

4. Po spuštění tohoto kódu:

```
SET BLOB SIZE (vxBlob;100)
INTEGER TO BLOB (0x0206;vxBlob;PC byte ordering;*)
```

• Velikost vxBlob je 102 bytů
• Na všech platformách je vxBlob{0} = $06 a vxBlob{1} = $02
• Ostatní byty BLOBu zůstanou nezměněné

5. Po spuštění tohoto kódu:

```
SET BLOB SIZE (vxBlob;100)
vlPosun:=50
INTEGER TO BLOB (518;vxBlob;Macintosh byte ordering;vlPosun)
```

• Velikost vxBlob je 100 bytů
• Na všech platformách je vxBlob{0} = $02 a vxBlob{1} = $06
• Ostatní byty BLOBu zůstanou nezměněné
• Proměnná vlPosun se zvětší o 2 byty (nyní je 52)

Dále si přečtěte

BLOB to integer, BLOB to longint, BLOB to real, BLOB to text, LONGINT TO BLOB, REAL TO BLOB, TEXT TO BLOB.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

LONGINT TO BLOB (LONGINT DO BLOBU)

Příkazy a odkazy pro BLOB

Verze 6.0

LONGINT TO BLOB (longInt; blob; PořadíBytů {; posun | *})

Parametr	Typ		Popis
longInt	Číslo	→	Long Integer hodnota k zapsání do BLOBu
blob	BLOB	→	BLOB k uložení Long Integer hodnoty
PořadíBytů	Číslo	→	0 Přirozené pořadí dle stroje
			1 Pořadí Macintosh
			2 Pořadí Windows
posun | *	Číslo | *	→	Posun v BLOBu nebo * pro připojení na konec
		←	Nový posun po zapsání, pokud je uveden a není *

Popis

Příkaz LONGINT TO BLOB zapíše 4 bytovou hodnotu Long Integer z proměnné *LongInt* do BLOBu *blob*.

Parametr *PořadíBytů* stanoví pařadí zapsání bytů hodnoty Long Integer. Musíte vložit jednu z předdefinovaných konstant 4D (pořadí dle patformy, Mac, PC) :

Konstanta	***Typ***	***Hodnota***
Native byte ordering	*Long Integer*	*0*
Macintosh byte ordering	*Long Integer*	*1*
PC byte ordering	*Long Integer*	*2*

Poznámka k nezávislosti na Platformě

Pokud používáte BLOB mezi platformami Macintosh a PC, je důležité bytů při použití tohoto příkazu řídit pořadí zápisu bytů.

Pokud použijete parametr * je hodnota Long Integer přidána na konec BLOBu a jeho velikost patřičně vzroste. S použitím tohoto parametru můžete postupně přidávat na konec BLOB tolik hodnot Integer, Long Integer, Real nebo Text (přečtěte si další příkazy k BLOBu) dokud se BLOB vejde do paměti.

Pokud nepřidáte parametr * nebo posun, zapíše se 4 bytová hodnota Long Integer na začátek BLOBu a přepíše jeho obsah; velikost se patřičně změní.

Pokud předáte proměnný parametr *posun*, 4-bytová hodnota Long Integer se v BLOBu zapíše od pozice *posun* (číslování začíná od nuly). Velikost BLOBu se patřičně změní podle toho kam zapíšete proměnnou tj.podle umístění dle proměnné *posun* (pokud je to za současný konec velikost bude *posun* plus 4 byty). Nově předáné byty (např. před začátkem zápisu), jsou nastaveny na nula.

Po provedení příkazu je navrácena proměnná *posun* zvětšená o počet bytů které jste přidali. Proto můžete použít tuto proměnnou s jiným příkazem pro BLOB k zapsání jiné hodnoty.

Příklady

1. Po spuštění tohoto kódu:

```
LONGINT TO BLOB (0x01020304;vxBlob;Native byte ordering)
```

- Velikost vxBlob je 4 byty
- Na Macintoshi vxBLOB{0}=$01, vxBLOB{1}=$02, vxBLOB{2}=$03, vxBLOB{3}=$04
- Na PC vxBLOB{0}=$04, vxBLOB{1}=$03, vxBLOB{2}=$02, vxBLOB{3}=$01

2. Po spuštění tohoto kódu:

```
LONGINT TO BLOB (0x01020304;vxBlob;Macintosh byte ordering)
```

• Velikost vxBlob je 4 byty
• Na všech Platformách vxBLOB{0}=$01, vxBLOB{1}=$02, vxBLOB{2}=$03, vxBLOB{3}=$04

3. Po spuštění tohoto kódu:

```
LONGINT TO BLOB (0x01020304;vxBlob;PC byte ordering)
```

• Velikost vxBlob je 4 byty
• Na všech Platformách je vxBLOB{0}=$04, vxBLOB{1}=$03, vxBLOB{2}=$02, vxBLOB{3}=$01

4. Po spuštění tohoto kódu:

```
SET BLOB SIZE (vxBlob;100)
LONGINT TO BLOB (0x01020304;vxBlob;PC byte ordering;*)
```

• Velikost vxBlob je 104 byty
• Na všech platformách vxBLOB{100}=$04, vxBLOB{101}=$03, vxBLOB{102}=$02, vxBLOB{103}=$01
• Ostatní byty BLOBu jsou nezměněné

5. Po spuštění tohoto kódu:

```
SET BLOB SIZE (vxBlob;100)
vlPosun:=50
LONGINT TO BLOB (0x01020304;vxBlob;Macintosh byte ordering;vlPosun)
```

• Velikost vxBlob je 100 bytů
• Na všech platformách vxBLOB{0}=$01, vxBLOB{1}=$02, vxBLOB{2}=$03, vxBLOB{3}=$04
• Ostatní byty BLOBu jsou nezměněné
• Proměnná vlPosun se zvětší o 4 (nyní je jeho velikost 54)

Dále si přečtěte

BLOB to integer, BLOB to longint, BLOB to real, BLOB to text, INTEGER TO BLOB, REAL TO BLOB, TEXT TO BLOB.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

REAL TO BLOB (REAL DO BLOBU)

Příkazy a odkazy pro BLOB

Verze 6.0

REAL TO BLOB (real; blob; FormátReal {; posun | *})

Parametr	Typ		Popis
real	Číslo	→	Hodnota Real k zapsání do BLOBu
blob	BLOB	→	BLOB k uložení hodnoty
FormátReal	Číslo	→	0 Přirozené pořadí dle stroje
			1 Pořadí real Macintosh
			2 Pořadí dvojitý real Macintosh
			3 Pořadí dvojitý real Windows
posun | *	Číslo | *	→	Posun v BLOBu nebo * pro připojení na konec
		←	Nový posun po zapsání pokud není *

Popis

Příkaz REAL TO BLOB zapíše Real hodnotu proměnné *real* do BLOBu *blob*.

Parametr *FormátReal* definuje vnitřní formát a pořadí bytů pro Real hodnotu. Musíte vložit jednu z předdefinovaných konstant 4D (pořadí dle patformy, Mac, Mac dvojitý, PC):

Konstanta	***Typ***	***Hodnota***
Native real format	*Long Integer*	*0*
Macintosh real format	*Long Integer*	*1*
Macintosh Double real format	*Long Integer*	*2*
Windows Double real format	*Long Integer*	*3*

Poznámka k nezávislosti na Platformě

Pokud používáte BLOB mezi platformami Macintosh a PC, je důležité při použití tohoto příkazu řídit pořadí bytů.

Pokud použijete parametr * je hodnota Real přidána na konec BLOBu a jeho velikost patřičně vzroste. S použitím tohoto parametru můžete postupně přidávat tolik hodnot Integer, Long Integer, Real nebo Text (přečtěte si další příkazy k BLOBu) dokud se BLOB vejde do paměti.

Pokud nepřidáte parametr * nebo *posun*, zapíše se hodnota Real na začátek BLOBu a přepíše jeho obsah; velikost se patřičně změní.

Pokud předáte proměnný parametr *posun*, hodnota Real se v BLOBu zapíše od pozice posun (číslování začíná od nuly). Velikost BLOBu se patřičně změní podle toho kam zapíšete proměnnou tj.podle umístění dle proměnné *posun* (pokud je to za současný konec velikost BLOB bude *posun* plus 8 nebo 10 bytů). Nově předáné byty (např. před začátkem zápisu), jsou nastaveny na nula.

Po provedení příkazu je navrácena proměnná *posun* zvětšená o počet bytů které jste přidali. Proto můžete použít tuto proměnnou s jiným příkazem pro BLOB k zapsání jiné hodnoty.

Příklady

1. Po spuštění tohoto kódu:

```
C_REAL (vrValue)
vrValue := ...
REAL TO BLOB (vrValue;vxBlob;Native real format)
```

- Na PC a Power Macintoshi, velikost vxBlob je 8 bytů
- Na Macintoshi 68K, velikost vxBlob je 10 bytů

2. Po spuštění tohoto kódu:

```
C_REAL (vrValue)
vrValue := ...
REAL TO BLOB (vrValue;vxBlob;Extended real format)
```

• Na všech platformách, velikost vxBlob je 10 bytů

3. Po spuštění tohoto kódu:

```
C_REAL (vrValue)
vrValue := ...
REAL TO BLOB (vrValue;vxBlob;Macintosh Double real format) ` nebo Windows double real format
```

• Na všech platformách, velikost vxBlob je 8 bytů

4. Po spuštění tohoto kódu:

```
SET BLOB SIZE (vxBlob;100)
C_REAL (vrValue)
vrValue := ...
INTEGER TO BLOB (vrValue;vxBlob;Windows Double real format) ` n. Macintosh double real format
```

• Na všech platformách, velikost vxBlob je 8 bytů

5. Po spuštění tohoto kódu:

```
SET BLOB SIZE (vxBlob;100)
REAL TO BLOB (vrValue;vxBlob;Extended real format;*)
```

• Na všech platformách, velikost vxBlob je 110 bytů
• Na všech platformách, real hodnota je uložena v bytech #100 až #109
• Ostatní byty BLOBu jsou nezměněné

6. Po spuštění tohoto kódu:

```
SET BLOB SIZE (vxBlob;100)
C_REAL (vrValue)
vrValue := ...
vlPosun:=50
REAL TO BLOB (vrValue;vxBlob;Windows Double real format;vlPosun)`n.Macintosh double real format
```

• Na všech platformách, velikost of vxBlob je 100 bytů
• Na všech platformách, real hodnota je uložena v bytech #50 až #57
• Ostatní byty BLOBu jsou nezměněné
• Proměnná vlPosun se zvětšila o 8 (nyní je 58)

Dále si přečtete

BLOB to integer, BLOB to longint, BLOB to real, BLOB to text, INTEGER TO BLOB, LONGINT TO BLOB, TEXT TO BLOB.

Příkazy a odkazy pro BLOB

TEXT TO BLOB (TEXT DO BLOBU)

Příkazy a odkazy pro BLOB

Verze 6.0

TEXT TO BLOB (text; blob; FormátText {; posun | *})

Parametr	Typ		Popis
text	Řetězec	→	Text k zapsání do BLOBu
blob	BLOB	→	BLOB k uložení textu
FormátText	Číslo	→	0 C String 1 Pascal String 2 Text s délkou 3 Text bez délky
posun | *	Číslo | *	→	Posun v BLOBu nebo * pro připojení na konec
		←	Nový posun po zapsání pokud není *

Popis

Příkaz TEXT TO BLOB zapíše textovou hodnotu z proměnné *text* do BLOBu *blob*.

Parametr *FormátText* definuje vnitřní formát a pořadí bytů pro textovou hodnotu. Musíte vložit jednu z předdefinovaných konstant 4D(řetězec C, Pascal, text s délkou, text bez délky) :

Konstanta	***Typ***	***Hodnota***
C String	*Long Integer*	*0*
Pascal String	*Long Integer*	*1*
Text with length	*Long Integer*	*2*
Text without length	*Long Integer*	*3*

Následující tabulka popisuje každý z těchto fotmátů:

Textový formát	Popis a Příklady
C string	Text je ukončený znakem nula (ASCII kód $00) "" → $00 "Hello World!" → $48 65 6C 6C 6F 20 57 6F 72 6C 64 21 00
Pascal string	Text předchází 1 byte s uvedením délky "" → $00 "Hello World!" → $0C 48 65 6C 6C 6F 20 57 6F 72 6C 64 21
Text with length	Textu předchází 2 byty s uvedením délky "" → $00 00 "Hello World!" → $00 0C 48 65 6C 6C 6F 20 57 6F 72 6C 64 21
Text without length	Text je složen pouze z vlastních znaků "" → Žádná data "Hello World!" → $48 65 6C 6C 6F 20 57 6F 72 6C 64 21

Poznámka: Příkaz přijímá jak Text (definovaný C_TEXT) tak Řetězec (definovaný C_STRING). Nezapomeňte že Text může být dlouhý až 32000 znaků a Řetězec může obsahovat počet znaků definovaný při vytvoření, ale maximálně 255 znaků.

Pokud předáte parametr * je hodnota Text přidána na konec BLOBu a jeho velikost patřičně vzroste. S použitím tohoto parametru můžete postupně přidávat tolik hodnot Integer, Long Integer, Real nebo Text (přečtěte si další příkazy k BLOBu) dokud se BLOB vejde do paměti.

Pokud nepředáte parametr * nebo posun, zapíše se hodnota Text na začátek BLOBu a přepíše jeho obsah; velikost se patřičně změní.

Pokud předáte proměnný parametr posun, hodnota Text se v BLOBu zapíše na pozici posun (číslování začíná od nuly). Velikost BLOBu se patřičně změní podle toho kam zapíšete proměnnou tj.podle umístění dle proměnné *posun* (pokud je to za současný konec velikost bude *posun* plus velikost textu v bytech). Nově předáné byty (např. před začátkem zápisu), jsou nastaveny na nula.

Po provedení příkazu je navrácena proměnná posun zvětšená o počet bytů které jste přidali. Proto můžete použít tuto proměnnou s jiným příkazem pro BLOB k zapsání jiné hodnoty.

Příklady

Po spuštění tohoto kódu:

```
SET BLOB SIZE (vxBlob;0)
C_TEXT (vtValue)
vtHodnota := "Nazdar světe!" ` Délka vtHodnota je 12 bytů
TEXT TO BLOB (vtValue;vxBlob;C string) ` Velikost BLOBu je 13 bytů
TEXT TO BLOB (vtValue;vxBlob;Pascal string) ` Velikost BLOBu je 13 bytů
TEXT TO BLOB (vtValue;vxBlob;Text with length) ` Velikost BLOBu je 14 bytů
TEXT TO BLOB (vtValue;vxBlob;Text without length) ` Velikost BLOBu je 12 bytů
```

Dále si přečtěte

BLOB to integer, BLOB to longint, BLOB to real, BLOB to text, INTEGER TO BLOB, LONGINT TO BLOB, REAL TO BLOB.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

BLOB to integer (BLOB do integer)

Příkazy a odkazy pro BLOB

Verze 6.0

BLOB to integer (blob; pořadíBytů {; posun }) → Číslo

Parametr	Typ		Popis
blob	BLOB	→	BLOB ze kterého načíst Integer hodnotu
pořadíBytů	Číslo	→	0 Přirozené pořadí dle stroje
			1 Pořadí Macintosh
			2 Pořadí Windows
posun	Proměnná	→	Posun v BLOBu (zadaný v bytech)
		←	Nový posun po provedení
Výsledek funkce	Číslo	←	2-bytová integer hodnota

Popis

Příkaz BLOB to integer vrací 2 bytovou Integer hodnotu přečtenou z BLOBu *blob*.

Parametr *pořadíBytů* nastaví pořadí čtení bytů pro Integer hodnotu. Předáte jednu z předdefinovaných konstant 4D:

Konstanta	***Typ***	***Hodnota***
Native byte ordering	*Long Integer*	*0*
Macintosh byte ordering	*Long Integer*	*1*
PC byte ordering	*Long Integer*	*2*

Poznámka k nezávislosti na Platformě

Pokud používáte BLOB mezi platformami Macintosh a PC, je důležité při použití tohoto příkazu řídit pořadí bytů.

Pokud předáte proměnný parametr *posun*, 2-bytová hodnota Integer se z BLOBu přečte od pozice posun (číslování začíná od nuly). Pokud tento parametr nepoužijete, přečtou se první 2 byty z BLOBu.

Poznámka: Hodnotu posun můžete použít od 0 do Velikost BLOBu mínus 2. Pokud to neuděláte, bude generována chyba -111.

Po provedení příkazu je navrácena proměnná *posun* zvětšená o počet bytů které jste přidali. Proto můžete použít tuto proměnnou dále s jiným příkazem pro BLOB k přečtení jiné hodnoty.

Příklady

Následující příklad přečte 20 hodnot Integer z BLOBu a začne na posunu 0x200:

```
$vlPosun:=0x200
For ($viSmyčka;0;19)
   $viHodnota:=BLOB to integer(vxUrčitýBlob;PC byte ordering;$vlPosun)
      ` Udělat něco s $viHodnota
End for
```

Dále si přečtěte

BLOB to longint, BLOB to real, BLOB to text, INTEGER TO BLOB, LONGINT TO BLOB, REAL TO BLOB, TEXT TO BLOB.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

BLOB to longint (BLOB do longint)

Příkazy a odkazy pro BLOB

Verze 6.0

BLOB to longint (blob; pařadíBytů {; posun }) → Číslo

Parametr	Typ		Popis
blob	BLOB	→	BLOB ze kterého vzít Long Integer hodnotu
pořadíBytů	Číslo	→	0 Přirozené pořadí dle stroje
			1 Pořadí Macintosh
			2 Pořadí Windows
posun	Proměnná	→	Posun v BLOBu (zadaný v bytech)
		←	Nový posun po provedení
Výsledek funkce	Číslo	←	4-bytová Long Integer hodnota

Popis

Příkaz BLOB to longint vrací 4 bytovou Long Integer hodnotu přečtenou z BLOBu *blob*.

Parametr *pořadíBytů* nastaví řazení bytů pro Long Integer hodnotu. Předáte jednu z předdefinovaných konstant 4D:

Konstanta	***Typ***	***Hodnota***
Native byte ordering	*Long Integer*	*0*
Macintosh byte ordering	*Long Integer*	*1*
PC byte ordering	*Long Integer*	*2*

Poznámka k nezávislosti na Platformě

Pokud používáte BLOB mezi platformami Macintosh a PC, je důležité při použití tohoto příkazu řídit pořadí bytů.

Pokud předáte proměnný parametr *posun*, 4-bytová hodnota Long Integer se z BLOBu přečte od pozice posun (číslování začíná od nuly). Pokud tento parametr nepoužijete, přečtou se první 4 byty z BLOBu.

Poznámka: Hodnotu pro *posun* můžete použít od 0 do Velikost BLOBu mínus 4. Pokud to neuděláte, bude generována chyba -111.

Po provedení příkazu je navrácena proměnná posun zvětšená o počet bytů které jste přidali. Proto můžete použít tuto proměnnou s jiným příkazem pro BLOB k přečtení jiné hodnoty.

Příklad

Následující příklad přečte 20 Long Integer hodnot z BLOBu a začne na posunu 0x200:

```
$vlPosun:=0x200
For ($viSmyčka;0;19)
   $vlHodnota:=BLOB to longint(vxUrčitýBlob;PC byte ordering;$vlPosun)
      ` Udělat něco s $vlHodnota
End for
```

Dále si přečtěte

BLOB to integer, BLOB to real, BLOB to text, INTEGER TO BLOB, LONGINT TO BLOB, REAL TO BLOB, TEXT TO BLOB.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

BLOB to real (BLOB do real)

Příkazy a odkazy pro BLOB

Verze 6.0

BLOB to real (blob; FormátReal {; posun }) → Číslo

Parametr	Typ		Popis
blob	BLOB	→	BLOB ze kterého se bude číst Real hodnota
FormátReal	Číslo	→	0 Přirozené pořadí dle stroje
			1 Pořadí real Macintosh
			2 Pořadí dvojitý real Macintosh
			3 Pořadí dvojitý real Windows
posun	Proměnná	→	Posun v BLOBu (značený v bytech)
		←	Nový posun po přečtení
Výsledek funkce	Číslo	←	8/10-bytová Real hodnota

Popis

Příkaz BLOB to real přečte Real hodnotu z BLOBu *blob*.

Parametr *FormátReal* definuje vnitřní formát pořadí čtení bytů pro Real hodnotu. Musíte vložit jednu z předdefinovaných konstant 4D:

Konstanta	***Typ***	***Hodnota***
Native real format	*Long Integer*	*0*
Macintosh real format	*Long Integer*	*1*
Macintosh Double real format	*Long Integer*	*2*
Windows Double real format	*Long Integer*	*3*

Poznámka k nezávislosti na Platformě

Pokud používáte BLOB mezi platformami Macintosh a PC, je důležité při použití tohoto příkazu řídit pořadí bytů.

Pokud předáte proměnný parametr *posun*, hodnota Real se z BLOBu přečte od pozice posun (číslování začíná od nuly). Pokud tento parametr nepoužijete, přečte se prvních 8 až 10 bytů z BLOBu.

Poznámka: Hodnotu posun můžete použít od 0 do Velikost BLOBu mínus 8 nebo 10. Pokud to neuděláte, bude generována chyba -111.

Po provedení příkazu je navrácena proměnná *posun* zvětšená o počet bytů které jste přečetli. Proto můžete použít tuto proměnnou dále s jiným příkazem pro BLOB k přečtení jiné hodnoty.

Příklad

Následující příklad přečte 20 Real hodnot z BLOBu a začne na posunu 0x200:

```
$vlPosun:=0x200
For ($viSmyčka;0;19)
   $vrHodnota:=BLOB to real(vxUrčitýBlob;PC byte ordering;$vlPosun)
      ` Udělat něco s $vrHodnota
End for
```

Dále si přečtěte

BLOB to integer, BLOB to longint, BLOB to text, INTEGER TO BLOB, LONGINT TO BLOB, REAL TO BLOB, TEXT TO BLOB.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

BLOB to text (BLOB do textu)

Příkazy a odkazy pro BLOB

Verze 6.0

BLOB to text (blob; FormátText {; posun {; DélkaTextu}}) → Text

Parametr	Typ		Popis
blob	BLOB	→	BLOB ze kterého vzít hodnotu textu
FormátText	Číslo	→	0 C String 1 Pascal String 2 Text s délkou 3 Text bez délky
posun	Proměnná	→	Posun v BLOBu (zadaný v bytech)
		←	Nový posun po přečtení
DélkaTextu	Číslo	→	Počet znaků k přečtení
Výsledek funkce	Číslo	→	Textová hodnota

Popis

Příkaz BLOB to text vrací textovou hodnotu přečtenou z BLOBu *blob*.

Parametr *FormátText* definuje vnitřní formát a pořadí čtení bytů pro textovou hodnotu. Musíte vložit jednu z předdefinovaných konstant 4D:

Konstanta	***Typ***	***Hodnota***
C String	*Long Integer*	*0*
Pascal String	*Long Integer*	*1*
Text with length	*Long Integer*	*2*
Text without length	*Long Integer*	*3*

Následující tabulka popisuje každý z těchto fotmátů:

Textový formát	**Popis a Příklady**
C String	Text je ukončený znakem nula (ASCII kód $00) "" → $00 "Hello World!" → $48 65 6C 6C 6F 20 57 6F 72 6C 64 21 00
Pascal String	Text předchází 1 byte s údajem o délce "" → $00 "Hello World!" → $0C 48 65 6C 6C 6F 20 57 6F 72 6C 64 21
Text with length	Textu předchází 2 byty s údajem o délce "" → $00 00 "Hello World!" → $00 0C 48 65 6C 6C 6F 20 57 6F 72 6C 64 21
Text without length	Text je složen pouze z vlastních znaků "" → Žádná data "Hello World!" → $48 65 6C *6C 6F 20 57 6F 72 6C 64 21*

UPOZORNĚNÍ: Počet znaků, které se budou číst je ovlivněn parametrem *FormátText*, KROMĚ formátu Text bez délky, pro který MUSÍTE definovat počet znaků pro přečtení v parametru *DélkaTextu*. Pro ostatní formáty textu je *DélkaTextu* ignorována a můžete jí vynechat.

Nezapomeňte, že textová proměnná může obsahovat až 32000 znaků a Řetězcová proměnná může obsahovat předem definovaný počet znaků, ale maximálně 255. Pokud se pokusíte načíst více než je možnost proměnné, 4D ořízne čtený text.

Pokud předáte proměnný parametr *posun*, hodnota Real se z BLOBu přečte od pozice posun (číslování začíná od nuly). Pokud tento parametr nepoužijete, čte od začáteku BLOBu podle formátu který zadáte. Nezapomeňte že musíte zadat parametr *posun* pokud používáte formát Text bez délky.

Poznámka: Pro parametr *posun* můžete zadat hodnotu mezi nula a velikosti BLOBu mínus velikost textu k přečtení. Pokud to neuděláte, výsledek funkce není zaručen.

Po provedení příkazu je navrácena proměnná *posun* zvětšená o počet bytů které jste přečetli. Proto můžete použít tuto proměnnou dále s jiným příkazem pro BLOB k přečtení jiné hodnoty.

Příklad

Tento příklad přečte hypotetický zdroj MacOs jehož vnitřní formát je stejný jako "STR#" zdroje:

```
GET RESOURCE ("ABCD";viResID;vxResData;viMyResFile)
vlVelikost:=BLOB Size(vxResData)
If (vlVelikost >0)
      ` Zdroj začíná 2-bytovým integerem definujícím počet řetězců
   vlPosun:=0
   viNbVstupů:=BLOB to integer(vxResData;Macintosh Byte Ordering;vlPosun)
      ` Pak zdroj obsahuje text ve formátu Pascal řetězce
   For (viVstup;1; viNbVstupů)
      If (vlPosun< vlVelikost)
         vsVstup:=BLOB to text(vxResData;Pascal string;vlPosun)
            ` Udělat něco s vsVstup
      Else
            ` Zdroj dat je špatný, ukončit smyčku
         viVstup:= viNbVstupů +1
      End if
   End for
End if
```

Dále si přečtěte

BLOB to integer, BLOB to longint, BLOB to real, INTEGER TO BLOB, LONGINT TO BLOB, REAL TO BLOB, TEXT TO BLOB.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

INSERT IN BLOB (VLOŽIT DO BLOBU)

Verze 6.0

INSERT IN BLOB (blob; posun; délka {; vyplnit})

Parametr	Typ		Popis
blob	BLOB	→	BLOB do kterého se bude vkládat
posun	Proměnná	→	Kde v BLOBu začít s vkládáním
délka	Číslo	→	Počet bytů které se vloží
vyplnit	Číslo	→	Výchozí hodnota bytů (0x00..0xFF) pokud vynecháno tak 0x00

Popis

Příkaz INSERT IN BLOB přidá do BLOBu *blob* počet bytů definovaný parametrem *délka* na pozici definovanou parametrm *posun*. BLOB se zvětší o počet bytů *délka*.

Pokud nedefinujete parametr *vyplnit*, budou byty předáné do BLOBu nastaveny na 0x00. Jinak můžete definovat jakýkoli ASCII znak zadáním jeho číslo (0...255).

Před provedením příkazu zadáte do *posun* pozici pro předání a po dokončení příkazu vám *posun* vrátí pozici za předánými byty.

Dále si přečtěte

DELETE FROM BLOB.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DELETE FROM BLOB (VYMAZAT Z BLOBU)

Verze 6.0

DELETE FROM BLOB (blob; posun; délka)

Parametr	Délka		Popis
blob	BLOB	→	BLOB ze kterého se bude mazat
posun	Číslo	→	Odkud začít mazat
délka	Číslo	→	Počet bytů k vymazání

Popis

Příkaz DELETE FROM BLOB vymaže z BLOBu *blob* počet bytů definovaných parametrem *délka* a začne na pozici definované *posun*. BLOB se zmenší o *délka* bytů.

Dále si přečtěte

INSERT IN BLOB.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

COPY BLOB (KOPÍROVAT BLOB)

Příkazy a odkazy pro BLOB

Verze 6.0

COPY BLOB (zdrBLOB; cílBLOB; zdrPosun; cílPosun; délka)

Parametr	Typ		Popis
zdrBLOB	BLOB	→	Zdrojový BLOB
cílBLOB	BLOB	→	Cílový BLOB
zdrPosun	Proměnná	→	Výchozí pozice pro kopírování
cílPosun	Proměnná	→	Cílová pozice pro kopírování
délka	Číslo	→	Počet bytů ke zkopírování

Popis

Příkaz COPY BLOB zkopíruje určitý počet bytů definovaný parametrem *délka* z BLOBu *zdrBLOB* do BLOBu *cílBLOB*.

Kopírování začne na pozici definované *zdrPosun* (ve zdrojovém BLOBu) a umístí se na pozici definovanou *cílPosun* (v cílovém BLOBu).

Poznámka

U cílového BLOBu se může změnit velikost.

Po provedení příkazu vrátí proměnné *zdrPosun* a *cílPosun* novou pozici v cílovém a zdrojovém BLOBu.

Dále si přečtěte

DELETE FROM BLOB, INSERT IN BLOB.

Příkazy a odkazy pro BLOB

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Logické příkazy

Příkazy a odkazy pro Logické

Verze 6.0

4D obsahuje Logické funkce pro Logické výpočty

True (Pravda)

False (Nepravda)

Not (Ne)

Příklady

Tento příkaz nastaví logickou proměnnou založenou na hodnotě tlačítka. Nastaví proměnnou maLogicka na True pokud bylo na tlačítko meTlacitko klepnuto a False pokud na tlačítko nebylo klepnuto. Pokud je klepnuto na tlačítko, proměnná tlačítka je nastavena na 1.

```
If (meTlacitko=1) ` Pokud bylo na tlačítko klepnuto
   maLogicka:=True ` maLogicka je nastavena na True
Else ` Pokud na tlačítko nebylo klepnuto,
   maLogicka:=False ` maLogicka je nastavena na False
End if
```

Předchozí příklad může být zjednodušen do jednoho řádku:

```
maLogicka:=(meTlacitko =1)
```

Dále si přečtěte

False, Logické operátory, Not, True.

Následující příkazy 4D vrací logický výsledek: Activated, After, Before, Before selection, Before subselection, Caps lock down, Compiled application, Deactivated, During, End selection, End subselection, In break, In footer, In header, In transaction, Is a list, Is a variable, Is in set, Is user deleted, Locked, Macintosh command down, Macintosh control down, Macintosh option down, Modified, Modified record, Nil, Outside call, Read only state, Semaphore, Shift down, True, Undefined, User in group, Windows Alt down, Windows Ctrl down.

Příkazy a odkazy pro Logické

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

True (Pravda)

Příkazy a odkazy pro Logické

Verze 3

True → Logické

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

True vrací logickou hodnotu True.

Příklad

Následující příklad nastaví proměnnou vbOption na Pravda.

```
vbOption := True
```

Dále si přečtěte

False, Not.

Příkazy a odkazy pro Logické

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

False (Nepravda)

Příkazy a odkazy pro Logické

Verze 3

False → Logické

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

False vrací logickou hodnotu False.

Příklad

Následující příklad nastaví proměnnou vbOption na Nepravda:

```
vbOption := False
```

Dále si přečtěte

Not, True.

Příkazy a odkazy pro Logické

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Not (Opak)

Příkazy a odkazy pro Logické

Verze 3

Not (Logické) → Logické

Parametr	Typ	Popis
Logické	Logické	Logická hodnota k obrácení

Popis

Funkce Not vrací negaci logické hodnoty, mění True na False a False na True.

Příklad

Tento příklad nejdříve přiřadí True k proměnné pak změní hodnotu proměnné na False a pak zpět na True:

```
vResult:=True ` vResult je nastaven na True
vResult:=Not(vResult) ` vResult je nastaven na False
vResult:=Not(vResult) ` vResult je nastaven na True
```

Příkazy a odkazy pro Logické

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

APPEND TO CLIPBOARD (PŘIPOJIT DO SCHRÁNKY)

Příkazy a odkazy pro Schránka

Verze 6.0

APPEND TO CLIPBOARD (TypDat; data)

Parametr	Typ		Popis
TypDat	Řetězec	→	4 znaky dlouhý řetězec typu
data	BLOB	→	Data k připojení do schránky

Popis

Příkaz APPEND TO CLIPBOARD připojí do Schránky data obsažená v BLOBu *data* a označí je datovým typem definovaným pomocí *TypDat*.

UPOZORNĚNÍ: Hodnota předáná do *TypDat* je citlivá na velikost písmen, tzn. že "abcd" není to samé jako "ABCD."

Pokud je BLOB *data* správně připojen do schránky je proměnná OK nastavena na 1. Jinak je proměnná OK nastavena na 0 a může být generována chyba.

Většinou použijete příkaz APPEND TO CLIPBOARD k připojení více částí jednoho typu nebo k připojení jiných datových typů než je TEXT nebo PICT. K připojení nových dat do Schránky, musíte nejdříve vyčisit Schránku použitím příkazu CLEAR CLIPBOARD.

Pokud chcete vymazat a připojit:
• text do Schránky, použijte příkaz SET TEXT TO CLIPBOARD.
• obrázek do Schránky, použijte příkaz SET PICTURE TO CLIPBOARD.

Jestliže však BLOB obsahuje nějaký text nebo obrázek, můžete použít příkaz APPEND TO CLIPBOARD k přidání textu nebo obrázku do Schránky.

Příklad

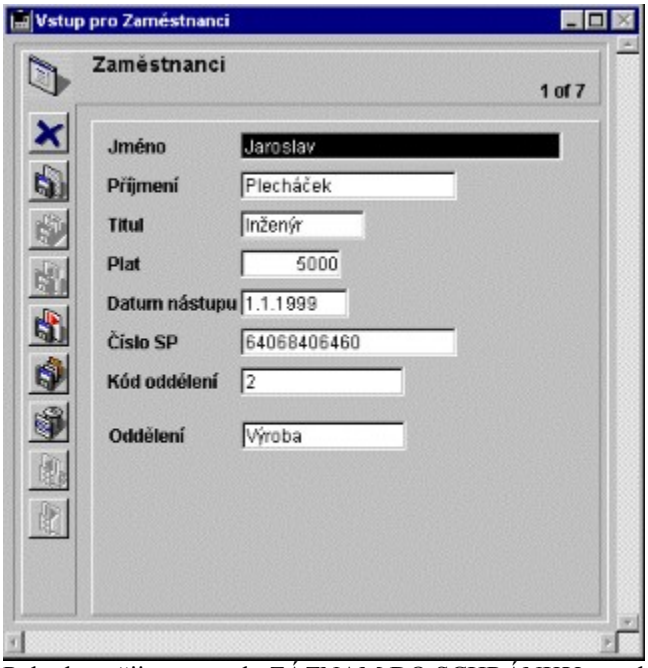
Pokud potřebujete prcovat spíše se strukturovanými daty než s jednotlivými kusy dat, můžete použitím příkazů pro Schránku a BLOBů, vytvořit složité schéma Vyjmout/Kopírovat/Vložit. V následujícím příkladu vám dvě metody projektu ZÁZNAM DO SCHRÁNKY a ZÁZNAM ZE SCHRÁNKY umožní kopírovat celé záznamy z a do Schránky.

```
    ` Metoda projektu ZÁZNAM DO SCHRÁNKY
    ` ZÁZNAM DO SCHRÁNKY ( číslo )
    ` ZÁZNAM DO SCHRÁNKY ( Číslo tabulky )
C_LONGINT($1;$vlPole;$vlPoleTyp)
C_POINTER($vpTable;$vpPole)
C_STRING(255;$vsNazev)
C_TEXT($vtDataZáznam;$vtPoleData)
C_BLOB($vxDataZáznam)
    ` Vymazat Schránku (Zůstane prázdná pokud tam není nějaký aktuální záznam)
CLEAR CLIPBOARD
    ` Vytvoří se ukazatel na tabulce, jejíž číslo bylo zadáno jako parametr
$vpTable:=Table($1)
    ` Jestliže je nějaký aktuální záznam
```

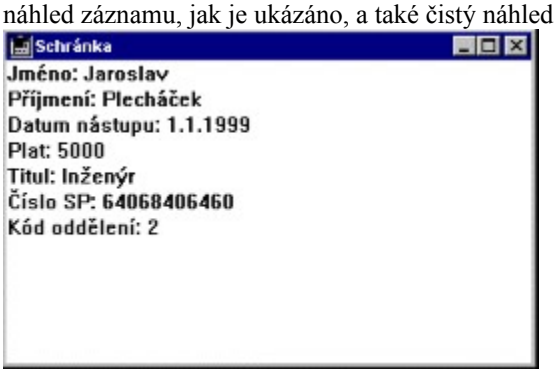
```
If ((Record number($vpTable→)>=0) | (Record number($vpTable→)=-3))
      ` Nastaví textovou proměnnou která bude obsahovat textový náhled záznamu
   $vtDataZáznam:=""
      ` Pro každé pole v záznamu:
   For ($vlPole;1;Count fields($1))
         ` Vezme typ pole
      GET FIELD PROPERTIES($1;$vlPole;$vlPoleTyp)
         ` Vytvoří ukazatel na poli
      $vpPole:=Field($1;$vlPole)
         ` Podle typu pole, zkopíruje (nebo ne) jeho data
         ` v patřičném pořadí
      Case of
         ÷ (($vlPoleTyp=Is Alpha field) | ($vlPoleTyp=Is Text ))
            $vtPoleData:=$vpPole→
         ÷ (($vlPoleTyp=Is Real ) | ($vlPoleTyp=Is Integer )
            | ($vlPoleTyp=Is LongInt ) | ($vlPoleTyp=Is Date) | ($vlPoleTyp=Is Time))
            $vtPoleData:=String($vpPole→)
         ÷ ($vlPoleTyp=Is Boolean )
            $vtPoleData:=String(Num($vpPole→);"Ano;;Ne")
      Else
            ` Přeskočit a ignorovat ostatní datové typy
         $vtPoleData:=""
      End case
         ` Nashromáždit data polí do textové proměnné
         ` která obsahuje textový náhled záznamu
      $vtDataZáznam:=$vtDataZáznam+Field name($1;$vlPole)+":"
                        +Char(9)+$vtPoleData+Carriage return
         ` Poznámka: metoda Carriage return vrací Char(13) na Macintoshi
         ` a Char(13)+ Char(10) na Windows
   End for
      ` Umístit textový náhled záznamu do Schránky
   SET TEXT TO CLIPBOARD($vtDataZáznam)
      ` Název scrap souboru v Temporary složce
   $vsNazev:= Temporary folder+"Scrap"+String(1+( Random%99))
      ` Vymaže scrap soubor jestliže existuje (může zde být testována chyba)
   DELETE DOCUMENT($vsNazev)
      ` Vytvoří scrap soubor
   SET CHANNEL(10;$vsNazev)
      ` Pošle celý záznam do scrap souboru
   SEND RECORD($vpTable→)
      ` Uzavřít scrap soubor
   SET CHANNEL(11)
      ` Načte scrap soubor do BLOBu
   DOCUMENT TO BLOB($vsNazev;$vxDataZáznam)
      ` Dále již nepotřebujeme scrap soubor
   DELETE DOCUMENT($vsNazev)
      ` Přiřadit plný náhled záznamu do Schránky
      ` Poznámka: Použijeme námi zvolený řetězec "4Drc" jako datový typ
   APPEND TO CLIPBOARD("4Drc";$vxDataZáznam)
      ` V tomto případě Schránka obsahuje:
      ` (1) Textový náhled záznamu (jak je ukázáno v následujícím obrázku)
      ` (2) Celkový náhled záznamu (obsahující Obrázek, Podpole a BLOB pole)
```

End if

Při zadání následujícího záznamu:



Pokud použijete metodu ZÁZNAM DO SCHRÁNKY na tabulku [Zaměstnanci], Schránka bude obsahovat textový náhled záznamu, jak je ukázáno, a také čistý náhled záznamu.



Můžete tento náhled záznamu vložit do jiného záznamu s použitím metody ZÁZNAM ZE SCHRÁNKY:

```
    ` Metoda ZÁZNAM ZE SCHRÁNKY
    ` ZÁZNAM ZE SCHRÁNKY ( Číslo )
    ` ZÁZNAM ZE SCHRÁNKY ( Číslo tabulky )
C_LONGINT($1;$vlPole;$vlPoleTyp;$vlPosCR;$vlPosColon)
C_POINTER($vpTable;$vpPole)
C_STRING(255;$vsNazev)
C_BLOB($vxClipboardData)
C_TEXT($vtClipboardData;$vtPoleData)
    ` Vytvořit ukazatel na tabulce jejíž číslo bylo předáno jako parametr
$vpTable:=Table($1)
    ` Pokud není žádný platný záznam
If ((Record number($vpTable→)>=0) | (Record number($vpTable→)=-3))
    Case of
```

```
        ` Obsahuje schránka úplný náhled záznamu?
÷ (Test clipboard("4Drc")>0)
        ` Pokud ano, načíst obsah Schráky
    GET CLIPBOARD("4Drc";$vxClipboardData)
        ` Název scrap souboru v Temporary složce
    $vsNazev:= Temporary folder+"Scrap"+String(1+( Random%99))
        ` Vymazat scrap soubor pokud existuje (zde může být testována chyba)
    DELETE DOCUMENT($vsNazev)
        ` Uložit BLOB do scrap souboru
    BLOB TO DOCUMENT($vsNazev;$vxClipboardData)
        ` Otevřít scrap soubor
    SET CHANNEL(10;$vsNazev)
        ` Načíst celý záznam ze scrap souboru
    RECEIVE RECORD($vpTable→)
        ` Uzavřít scrap soubor
    SET CHANNEL(11)
        ` Již nepotřebujeme scrap soubor
    DELETE DOCUMENT($vsNazev)
        ` Obsahuje schránka TEXT?
÷ (Test clipboard("TEXT")>0)
        ` Načíst text ze Schránky
    $vtClipboardData:=Get text from clipboard
        ` Nastavit čísla polí aby mohla narůstat
    $vlPole:=0
    Repeat
            ` Podívat se na další řádek pole v textu
        $vlPosCR:=Position(CR ;$vtClipboardData)
        If ($vlPosCR>0)
                ` Načíst řádek pole
            $vtPoleData:=Substring($vtClipboardData;1;$vlPosCR-1)
                ` Pokud je tam dvojtečka ":"
            $vlPosColon:=Position(":";$vtPoleData)
            If ($vlPosColon>0)
                ` Vzít pouze data pole (Odstraní název pole)
                $vtPoleData:=Substring($vtPoleData;$vlPosColon+2)
            End if
                ` Zvětšení čísla pole
            $vlPole:=$vlPole+1
                ` Schránka může obsahovat více dat než potřebujeme...
            If ($vlPole<=Count fields($vpTable))
                    ` Vzít typ pole
                GET FIELD PROPERTIES($1;$vlPole;$vlPoleTyp)
                    ` Vzít ukazatel na poli
                $vpPole:=Field($1;$vlPole)
                    ` V závisosti na typu pole,
                    ` kopíruje (nebo ne) text patřičným způsobem
                Case of
                    ÷ (($vlPoleTyp=Is Alpha field ) | ($vlPoleTyp=Is Text ))
                        $vpPole→:=$vtPoleData
                    ÷ (($vlPoleTyp=Is Real ) | ($vlPoleTyp=Is Integer ) | ($vlPoleTyp=Is LongInt ))
                        $vpPole→:=Num($vtPoleData)
                    ÷ ($vlPoleTyp=Is Date )
```

```
                        $vpPole→:=Date($vtPoleData)
                    ÷ ($vlPoleTyp=Is Time )
                        $vpPole→:=Time($vtPoleData)
                    ÷ ($vlPoleTyp=Is Boolean )
                        $vpPole→:=($vtPoleData="Ano")
                Else
                    ` Přeskočit a ignorovat ostatní datové typy
                End case
            Else
                    ` Všechna pole byla přiřazena, můžeme opustit smyčku
                $vtClipboardData:=""
            End if
                    ` Oddělit text, který už byl přečten
                $vtClipboardData:=Substring($vtClipboardData; $vlPosCR+Length(CR))
        Else
                    ` Nebyl nalezen oddělovač, opustíme smyčku
            $vtClipboardData:=""
        End if
            ` Opakovat dokud jsou nějaká data
    Until (Length($vtClipboardData)=0)
Else
    ALERT("Schránka neobsahuje žádná data, která se dají přečíst jako záznam.")
End case
End if
```

Dále si přečtěte

CLEAR CLIPBOARD, SET PICTURE TO CLIPBOARD, SET TEXT TO CLIPBOARD.

Systémové proměnné

Pokud jsou data správně předána do schránky, je OK nastavena na 1; jinak je nastavena na 0 a může být generována chyba.

Ovládání chyb

Pokud není dostatek paměti k připojení dat do Schránky, je generována chyba -108.

Příkazy a odkazy pro Schránka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CLEAR CLIPBOARD (VYMAZAT SCHRÁNKU)

Verze 6.0

CLEAR CLIPBOARD

Parametr Typ Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz CLEAR CLIPBOARD vymaže obsah Schránky. Pokud Schránka obsahuje více výskytů stejných dat, budou vymazány všechny výskyty. Po provedení tohoto příkazu zůstane Schránka prázdná.

Příkaz CLEAR CLIPBOARD musíte použít pokaždé když přidáváte nová data do Schránky pomocí příkazu APPEND TO CLIPBOARD, protože tento příkaz si nevymaže obsah Schránky, ale připojí nová data k již existujícím.

Jedno provedení CLEAR CLIPBOARD a pak několikeré provedení příkazu APPEND TO CLIPBOARD vám umožní Vyjmout nebo Kopírovat stejná data pod různými formáty.

Na druhou stránku příkazy SET TEXT TO CLIPBOARD a SET PICTURE TO CLIPBOARD automaticky vymažou obsah Schránky před předáním nového obsahu.

Příklad

1. Následující kód nejdříve vymaže a pak vloží data do schránky:

```
CLEAR CLIPBOARD ` Make sure the clipboard becomes empty
APPEND TO CLIPBOARD('XWKZ';$vxSomeData) ` Připojit nějaká data typu 'XWKZ'
APPEND TO CLIPBOARD('SYLK';$vxSylkData) ` Připojit nějaká data, ale ve formátu Sylk
```

2. Podívejte se na příklad u příkazu APPEND TO CLIPBOARD.

Dále si přečtěte

APPEND TO CLIPBOARD.

Příkazy a odkazy pro Schránka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GET CLIPBOARD (ZÍSKAT ZE SCHRÁNKY)

Příkazy a odkazy pro Schránka

Verze 6.0

GET CLIPBOARD (typDat; data)

Parametr	Typ		Popis
typDat	Řetězec	→	4 znaky dlouhý typ dat
data	BLOB	←	Požadovaná data vytažená ze schránky

Popis

Příkaz GET CLIPBOARD vrací do pole nebo do proměnné BLOB, předaného parametrem *data*, data uložená ve Schránce, jejichž typ je definovaný v parametru *typDat*.

UPOZORNĚNÍ: Hodnota předáná do *typDat* je citlivá na velikost písmen, tzn. že "abcd" není to samé jako "ABCD."

Pokud jsou data správně použita ze Schránky, je systémová proměnná OK nastavena na 1. Pokud je Schránka prázdná nebo neobsahuje data zadaného typu a příkaz vrátí prázdný BLOB, nastaví OK na 0 a generuje chybu -102. Pokud není dostatek paměti k vyjmutí dat ze Schránky, příkaz nastaví proměnnou OK na 0 a generuje chybu -108.

Příklad

Následujícící metody objektu pro dvě tlačítka kopírují data z a do array *asMoznosti* (rozevírací nabídka, rozevírací seznam, ...) umístěném ve formuláři:

```
    ` metoda objektu bKopirujMoznosti
If (Size of array(asMoznosti)>0) ` Je co kopírovat?
      ` Přesunout položky array do BLOBu
   VARIABLE TO BLOB (asMoznosti;$vxClipData)
   CLEAR CLIPBOARD ` Vyprázdní Schránku
   APPEND TO CLIPBOARD ("artx";asMoznosti) ` Všiměte si vybraného datového typu
End if
   ` metoda objektu bVlozMoznosti
If (Test clipboard ("artx")>0) ` Je nějaký typ dat "artx" ve Schránce?
   GET CLIPBOARD ("artx";$vxClipData) ` Vyjmout data ze Schránky
      ` Umístit array do BLOBu
   BLOB TO VARIABLE ($vxClipData;asMoznosti)
   asMoznosti:=0 ` Odznačit vybranou položku pro array
End if
```

Dále si přečtěte

APPEND TO CLIPBOARD, GET PICTURE FROM CLIPBOARD, Get text from clipboard.

Systémové proměnné

Pokud jsou data správně načtena, je proměnná OK nastavena na 1; jinak je nastavena na 0 a je generována chyba.

Ovládání chyb

• Pokud není dostatek paměti k načtení dat, je generována chyba -108.

• Pokud nejsou žádná data zadaného typu, je generována chyba -102.

Příkazy a odkazy pro Schránka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GET PICTURE FROM CLIPBOARD (ZÍSKAT OBRÁZEK ZE SCHRÁNKY)

Verze 6.0

GET PICTURE FROM CLIPBOARD (obrázek)

Parametr	Typ		Popis
obrázek	Obrázek	←	Obrázek načtený ze Schránky

Popis

GET PICTURE FROM CLIPBOARD vrací obrázek uložený ve Schránce do obrázkového pole nebo proměnné *obrázek*.

Pokud je obrázek správně načten ze Schránky, příkaz nastaví do proměnné OK 1. Jestliže je Schránka prázdná nebo neobsahuje obrázek, příkaz vrátí prázdný obrázek a nastaví proměnnou OK na 0 a generuje chybu -102. Pokud není dostatek paměti k načtení obrázku ze Schránky, příkaz nastaví proměnnou OK na 0 a generuje chybu -108.

Příklady

Následující metoda objektu tlačítka přiřadí obrázek ze Schránky (pokud nějaký je) do pole [Zaměstnanci]Fotka:

```
If (Test clipboard ("PICT")>0)
   GET PICTURE FROM CLIPBOARD ([Zaměstnanci]Fotka)
Else
   ALERT ("Schránka neobsahuje žádný obrázek.")
End if
```

Dále si přečtěte

GET CLIPBOARD, Get text from clipboard, Test clipboard.

Příkazy a odkazy pro Schránka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Get text from clipboard (Získat text ze schránky)

Příkazy a odkazy pro Schránka

Verze 6.0

Get text from clipboard → Řetězec

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Řetězec	←	Vrací text (pokud je nějaký) načtený ze Schránky

Popis

Get text from clipboard vrací text uložený ve Schránce.

Pokud je text správně načten ze Schránky, příkaz nastaví do proměnné OK 1. Jestliže je Schránka prázdná nebo neobsahuje text, příkaz vrátí prázdný řetězec a nastaví proměnnou OK na 0 a generuje chybu -102. Pokud není dostatek paměti k načtení obrázku ze Schránky, příkaz nastaví proměnnou OK na 0 a generuje chybu -108.

Textová pole a textové proměnné 4D mohou obsahovat až 32000 znaků. Pokud je ve Schránce více než 32000 znaků, příkaz při vkládání do proměnné nebo pole ořízne výsledný text ze Schránky. Pokud pracujete s velkými daty ve Schránce, použijte nejdříve příkaz Test clipboard a pokud bude Schránka obsahovat více než 32000 znaků, použijte příkaz GET CLIPBOARD místo Get text from clipboard.

Příklady

Následující příklad testuje obsah Schránky a pak podle velikosti načte data ze Schránky buď do textové proměnné nebo do BLOBu:

```
$vlVelikost:=Test clipboard ("TEXT")
Case of
   ÷ ($vlVelikost<=0)
      ALERT ("Ve Schránce není žádný text.")
   ÷ ($vlVelikost<=32000)
      $vtClipData:=Get text from clipboard
      If (OK=1)
         ` Udělat něco s textem
      End if
   ÷ ($vlVelikost>32000)
      GET CLIPBOARD ("TEXT";$vxClipData)
      If (OK=1)
         ` Udělat něco s BLOBem
      End if
End case
```

Dále si přečtěte

GET CLIPBOARD, GET PICTURE FROM CLIPBOARD, Test clipboard.

Systémové proměnné

Pokud je text správně načte je proměnná OK nastavena na 1, jinak je nastavena na 0 a je generována chyba.

Ovládání chyb

• Pokud není dostatek paměti k načtení textu, je generována chyba -108.
• Pokud ve Schránce není text, bude generována chyba -102.

Příkazy a odkazy pro Schránka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET PICTURE TO CLIPBOARD (VLOŽIT OBRÁZEK DO SCHRÁNKY)

Příkazy a odkazy pro Schránka

Verze 6.0

SET PICTURE TO CLIPBOARD (obrázek)

Parametr	Typ		Popis
obrázek	Obrázek	←	Obrázek jehož kopie bude umístěna do Schránky

Popis

SET PICTURE TO CLIPBOARD vymaže obsah Schránky a umístí do ní kopii obrázku *obrázek*.

Po předání obrázku do Schránky jej můžete zpět načíst příkazem GET PICTURE FROM CLIPBOARD nebo GET CLIPBOARD ("PICT";...).

Pokud je obrázek správně uložen do Schránky, je proměnná OK nastavena na 1. Pokud není dostatek paměti ke zkopírování obrázku do Schránky, je proměnná OK nastavena na 0 a je generována chyba.

Příklad

S použitím plovoucího okna zobrazíte formulář který obsahuje array asPrijmZamestn, který zobrazuje seznam jmen zaměstnanců z tabulky [Zaměstnanci]. Pokaždé když klepnete na jméno, chcete zkopírovat fotografii zaměstnance do Schránky. Do metody objektu pro array napíšete následující:

```
If (asPrijmZamestn#0)
   QUERY ([Zaměstnanci];[Zaměstnanci]Příjmení=asPrijmZamestn{asPrijmZamestn})
   If (Picture size ([Zaměstnanci]Fotka)>0)
      SET PICTURE TO CLIPBOARD ([Zaměstnanci]Fotka) ` kopírovat fotografii zaměstnance
   Else
      CLEAR CLIPBOARD ` Nebyl nalezen žádný záznam nebo fortografie
   End if
End if
```

Dále si přečtěte

APPEND TO CLIPBOARD, GET PICTURE FROM CLIPBOARD.

Systémové proměnné a Sady

Pokud je kopie obrázku správně umístěna do Schránky, je proměnná OK nastavena na 1.

Příkazy a odkazy pro Schránka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET TEXT TO CLIPBOARD (VLOŽIT TEXT DO SCHRÁNKY)

Verze 6.0

SET TEXT TO CLIPBOARD (text)

Parametr	Typ		Popis
text	Řetězec	→	Text jehož kopie bude umístěna do Schránky

Popis

SET TEXT TO CLIPBOARD vymaže obsah Schránky a vloží do ní kopii textu *text*.

Po předání textu do Schránky jej můžete zpět načíst příkazem Get text from clipboard nebo GET CLIPBOARD ("TEXT";...).

Pokud je text správně uložen do Schránky, je proměnná OK nastavena na 1. Pokud není dostatek paměti ke zkopírování textu do Schránky, je proměnná OK nastavena na 0 a je generována chyba.

Text 4th Dimension může obsahovat maximálně 32000 znaků. K práci s většími textovými bloky nejdříve tento blok přesuňte do BLOBu, proveďte CLEAR CLIPBOARD a pak APPEND TO CLIPBOARD ("TEXT";...).

Příklad

Přečtěte si příklady u příkazu APPEND TO CLIPBOARD.

Dále si přečtěte

APPEND TO CLIPBOARD, Get text from clipboard.

Systémové proměnné a Sady

Pokud je kopie textu správně přesunuta do Schránky je proměnná OK nastavena na 1.

Příkazy a odkazy pro Schránka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Test clipboard (Testovat schránku)

Příkazy a odkazy pro Schránka

Verze 6.0

Tet clipboard (typDat) → Číslo

Parametr	Typ		Popis
typDat	Řetězec	→	typu dat , 4 znaky dlouhý řetězec
Výsledek funkce	Číslo	←	Velikost dat (v bytech), uložených ve Schránce, nebo kód chyby

Popis

Příkaz Test clipboard vám umožní testovat jestli Schránka obsahuje typ dat charakterizovaný řetězcem předávaným v parametru *typDat* a velikost těchto dat.

UPOZORNĚNÍ: Hodnota předáná do typDat je citlivá na velikost písmen, tzn. že "abcd" není to samé jako "ABCD."

Pokud je Schránka prázdná nebo neobsahuje data zadaného typu, příkaz vrátí chybu -102 (podívejte se na tabulku předdefinovaných konstant). Pokud Schránka obsahuje data vybraného typu, příkaz vrátí velikost těchto dat vyjádřenou v bytech.

Po té, co zjistíte že Schránka obsahuje data vybraného typu, můžete je ze Schránky načíst pomocí jednoho z následujících příkazů:

• Pokud Schránka obsahuje data typu TEXT, můžete je načíst příkazem Get text from clipboard který vrátí textovou hodnotu nebo příkazem GET CLIPBOARD který vrátí text do BLOBu.
• Pokud Schránka obsahuje data txpu PICT, můžete je načíst příkazem GET PICTURE FROM CLIPBOARD který vrátí obrázek do obrázkové proměnné nebo pole nebo příkazem GET CLIPBOARD který vrací obrázek do BLOBu.
• Pro všechny další typy dat použijte příkaz GET CLIPBOARD který načte data do BLOBu.

4th Dimension obsahuje následující předdefinované kostanty:

Konstanta	***Typ***	***Popis***
No such data in clipboard	*Long Integer*	*-102*
Text data	*Řetězec*	*TEXT*
Picture data	*Řetězec*	*PICT*

Příklady

1. Následující kód testuje jestli Schránka obsahuje obrázek a pokud ano, zkopíruje tento obrázek do proměnné 4D:

```
If (Test clipboard (Picture data) > 0) ` Je ve Schránce obrázek?
   GET PICTURE FROM CLIPBOARD ($vPicPromenna) ` Pokud ano, načíst obrázek ze Schránky
Else
   ALERT("Ve Schránce není žádný obrázek.")
End if
```

2. Aplikace obvykle ze Schránky vyjmou a kopírují data typu TEXT a PICT, protože většina aplikací rozeznává pouze dva základní typy dat. Aplikace však může připojit do Schránky více výskytů jedněch dat v různých formátech. Například pokaždé když vyjmete nebo kopírujete různé části tabulky, tabulková aplikace může připojit data do Schránky pod hypotetickým formátem 'TBLK' stejně jako SYLK a TEXT. Formát 'TBLK' může obsahovat data ve specifické struktuře aplikace. Formát SYLK může obsahovat stejná data, ale s použitím formátu SYLK pak může tato data rozpoznat více aplikací. Konečně formát TEXT bude obsahovat stejná data bez zvláštních informací obsažených ve formátech SYLK a TBLK. Z tohoto důvodu při psaní rutin pro Vyjmout/Kopírovat/Vložit mezi 4th Dimension a jinými aplikacemi,by jste měli napsat podobný kód:

```
Case of
      ` Nejdříve otestujte jestli Schránka obsahuje nějaká data
      ` z hypotetické tabulkové aplikace
   ÷ (Test clipboard ('TBLK') > 0)
      ` ...
      ` Pak otestujte, jestli obsahuje nějaká SYLK data
   ÷ (Test clipboard ('SYLK') > 0)
      ` ...
      ` Nakonec otestujte, jestli obsahuje nějaká textová data
   ÷ (Test clipboard ('TEXT') > 0)
      ` ...
End case
```

Jinými slovy vyzkoušíte načíst ze Schránky data, která obsahují nejvíce původních informací.

3. Podívejte se na příklad u příkazu APPEND TO CLIPBOARD.

Dále si přečtěte

GET CLIPBOARD, GET PICTURE FROM CLIPBOARD, Get text from clipboard.

Příkazy a odkazy pro Schránka

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET CHANNEL (NASTAVIT KANÁL)

Příkazy a odkazy pro Komunikace

Verze 3

SET CHANNEL (port | operace; nastavení | dokument)

Parametr	Typ		Popis
port | operace	Číslo	→	Číslo sériového portu nebo operace dokumentu
nastavení | dokument	Číslo | Řetězec	→	Nastavení sériového portu nebo Název dokumentu

Popis

Příkaz SET CHANNEL otevře sériový port nebo dokument. Najednou můžete pomocí tohoto příkazu otevřít pouze jeden port nebo jeden dokument.

Historická poznámka

Tento příkaz byl první příkaz ve 4th Dimension pro práci se sériovými porty a dokumenty na disku. Od té doby byli přidány další příkazy. Dnes můžete pracovat s dokumenty na disku pomocí příkazů Open document, Create document a Append document. S těmito příkazy můžete číst a psát z a do dokumentů s použitím SEND PACKET nebo RECEIVE PACKET (tyto příkazy také pracují spolu se SET CHANNEL). Pokud budete chtít použít příkazy SEND VARIABLE, RECEIVE VARIABLE, SEND RECORD a RECEIVE RECORD, musíte použít příkaz SET CHANNEL k zpřístupnění dokumentu na disku.

Popis SET CHANNEL je složen ze dvou částí:

- Práce se sériovým portem.
- Práce s dokumenty.

Práce se sériovým portem - SET CHANNEL (port; nastavení)

První způsob použití SET CHANNEL je otevření sériového portu, nastavení protokolu a dalších informací o portu. Data mohou být poslána pomocí SEND PACKET, SEND RECORD nebo SEND VARIABLE, a načtena pomocí RECEIVE BUFFER, RECEIVE PACKET, RECEIVE RECORD nebo RECEIVE VARIABLE.

Parametry portu

První parametr, *port*, vybere port a protokol.

Na Windows:

Můžete adresovat až 99 portů (postupně). Následující tabulka zobrazuje hodnoty pro porty:

Rozsah	***Popis***
101 až 199	*Sériová komunikace bez protokolu*
201 až 299	*Sériová komunikace se softwarovým protokolem jako XON/XOFF*
301 až 399	*Sériová komunikace s hardwarovým protokolem jako RTS/CTS*

Důležité: Hodnota kterou předáte do parametru port se musí odkazovat k existujícímu dodatečnému COM portu rozpoznatelnému z Windows. Pokud budete například chtít použít hodnoty 101, 103 a 125, musí být porty COM1, COM3 a COM25 správně nastaveny (vzorec je protokol tj. 100,200,300 + číslo portu COM).

Na Macintosh:

Definujete hodnotu pro port předáním hodnoty sériového portu a protokolu jak je zobrazeno v následující tabulce v tím jak fungují na standardní COM1 a COM2 na Windows.

	Hodnoty pro součet Protokol + Port	***Popis***
Sériový port	*0*	*Port tiskárny na Macintoshi (Windows COM2)*
	1	*Modemový port na Macintoshi (Windows COM1)*
Protokol	*0*	*Žádný*
	20	*XON/XOFF*
	30	*DTR*

Například pro použití XON/XOFF s portem modemu, předáte 20 + 1 = 21. Pro parametr port použijete tedy číslo 21. Pro kompatibilitu kódu přes platformy jsu hodnoty portu použité na Macintoshi následovně použitelné na Windows:

Hodnota portu	***Popis***
0	*COM2*
1	*COM1*
20	*COM2 (se softwarovým protokolem jako XON/XOFF)*
21	*COM1 (se softwarovým protokolem jako XON/XOFF)*
30	*COM2 (s hardwarovým protokolem jako RTS/CTS)*
31	*COM1 (s hardwarovým protokolem jako RTS/CTS)*

Nastavení parametrů

Parametrem se nastavuje rychlost, počet datových bitů, počet stop bitů a paritu. Hodnotu parametru definujete nastavením hodnot pro rychlost, datové bity, stop bity a paritu, tak jak jsou zobrazeny v následující tabulce. Například pro nastavení rychlosti 12000 baudů, 8 datových bitů, 1 stop bit a žádnou paritu, předáte 94 + 3072 + 16384 + 0 = 19550. V parametru se pak předá číslo 19550.

	Do součtu do parametru nastavení	***Skutečná hodnota***
Rychlost (baud)	*380*	*300*
	189	*600*
	94	*1200*
	62	*1800*
	46	*2400*
	30	*3600*
	22	*4800*
	14	*7200*
	10	*9600*
	4	*19200*
	0	*57600*
	1022	*115200*
	1021	*230400*
Data bity	*0*	*5*
	2048	*6*
	1024	*7*
	3072	*8*
Stop bity	*16384*	*1*
	-32768	*1.5*
	-16384	*2*
Parita	*0*	*Žádná*
	4096	*Lichá*
	12288	*Sudá*

Tip: Různé číselné hodnoty k nastavení portu a parametru nastavení (neobsahující ale hodnoty pro přídavné COM1...COM99) jsou dostupné jako předdefinované konstanty v okně Průzkumníka v Prostředí návrháře. Pro COM1...COM99 použijte přímo číselné hodnoty.

Příklad :

Chcete-li nastavit port tiskárny/COM2 a žádný protokol, použijte jednu ze syntaxí:

SET CHANNEL (0;param)

nebo

SET CHANNEL (102;param)

Práce s Dokumenty na disku - SET CHANNEL (operace; dokument)

Druhý formát příkazu SET CHANNEL umožní vytvořit, otevřít a zavřít dokument. Na rozdíl od Systémových příkazů pro dokumenty, může otevřít najednou pouze jeden dokument. Dokument může být čten nebo do něj zapisováno.

Parametr *operace* definuje akci, která má být provedena s dokumentem určeném parametrm *dokument*. Následující tabulka zobrazuje hodnoty *operací* a rozdíly mezi funkcemi při různých parametrech *dokument*. První sloupec zobrazuje hodnoty pro parametr *operace* a druhý sloupec zobrazuje hodnoty pro *dokument*. Třetí sloupec zobrazuje výslednou provedenou operaci.

Například k zobrazení okna pro otevření dokumentu můžete použít následující řádek:

SET CHANNEL (13; "")

Operace	***Dokument***	***Výsledek***
10	*Řetězec*	*Otevře dokument definovaný pomocí řetězce. Jestliže dokument neexistuje, dokument je vytvořen a otevřen.*
10	*"" (prázdný řetězec)*	*Zobrazí okno pro otevření souboru. Jsou zobrazeny všechny typy.*
11	*nic*	*Zavře otevřený soubor.*
12	*"" (prázdný řetězec)*	*Zobrazí dialogové okno Uložit soubor k vytvoření nového souboru.*
13	*"" (prázdný řetězec)*	*Zobrazí okno pro otevření souboru. Jsou zobrazeny pouze textové dokumenty.*

Všechny operace v této tabulce nastavují, pokud je třeba, systémovou proměnnou Document. Také nastaví proměnnou OK na 1 pokud požadovaná akce proběhne správně, jinak je nastavena OK na 0.

Příklady

Podívejte se na příklady pro příkazy RECEIVE BUFFER, SET TIMEOUT a RECEIVE RECORD.

Dále si přečtěte

Append document, Create document, Open document, RECEIVE BUFFER, RECEIVE PACKET, RECEIVE RECORD, RECEIVE VARIABLE, SEND PACKET, SEND RECORD, SEND VARIABLE, SET TIMEOUT.

Příkazy a odkazy pro Komunikace

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET TIMEOUT (NASTAVIT ČASOVÝ LIMIT)

Příkazy a odkazy pro Komunikace

Verze 3

SET TIMEOUT (sekundy)

Parametr	Typ		Popis
sekundy	Číslo	→	Sekund do konce časového limitu

Popis

Příkaz SET TIMEOUT definuje kolik času má příkaz sériového portu k dokončení. Pokud nebude příkaz sériového portu dokončen v zadaném limitu, bude tento příkaz zrušen a generována chyba -9990 a proměnná OK bude nastavena na 0. Chybu můžete zachytit pomocí metody s použitím ON ERR CALL.

Všimněte si, že čas je celkový čas pro provedení příkazu a ne čas mezi přijetím znaků. Ke zrušení předchozího nastavení a zastavení sledování komunikace portu, použijte nastavení 0 pro parametr sekundy.

Příkazy které jsou ovlivněny nastavením časového limitu jsou:

- RECEIVE PACKET
- RECEIVE RECORD
- RECEIVE VARIABLE

Příklad

Následující příklad nastaví sériový port pro přijetí dat. Pak nastaví časový limit. Data jsou přijata pomocí RECEIVE PACKET. Pokud nejsou přijata v zadaném čase, je generována chyba.

```
    SET CHANNEL (MacOS Serial Port; Speed 9600 + Data Bits 8 + Stop Bits One + Parity None) ` Otevřít Sériový Port
    SET TIMEOUT (10) ` Nastaví čas na 10 sekund
    ON ERR CALL ("ZACHYTIT COM CHYBY") ` Nenechat metodu přerušit
    RECEIVE PACKET (vtBuffer; Char (13)) ` Číst až po znak Carriage Return, Nový řádek.
    If (OK=0)
       ALERT ("Chyba při přijetí dat.")
    Else
       [Lidé]Název:=vtBuffer ` Poslat přijatá data do pole
    End if
    ON ERR CALL("")
```

Dále si přečtěte

ON ERR CALL, RECEIVE BUFFER, RECEIVE PACKET, RECEIVE RECORD, RECEIVE VARIABLE.

Příkazy a odkazy pro Komunikace

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

USE ASCII MAP (POUŽÍT ASCII MAPU)

Příkazy a odkazy pro Komunikace

Verze 3

USE ASCII MAP (mapa | * {; mapInOut})

Parametr	Typ		Popis
mapa | *	Řetězec | *	→	Název dokumentu mapy která se použije, nebo * ke zrušení výchozí ASCII mapy.
mapVstVýst	Číslo	→	0 = Výstupní mapa 1 = Vstupní mapa Pokud vynecháno, pak výstupní mapa

Popis

Příkaz USE ASCII MAP má dva formáty. První je načíst ASCII mapu z disku a použít jí. Jestliže je *mapVstVýstt* 0, je tato mapa načtena jako výstupní a pokud je 1 je načtena jako vstupní.

ASCII mapa musí být dopředu vytvořena v dialogovém okně ASCII mapa v Prostředí uživatele. Jakmile je mapa načtena, 4th Dimension ji použije při přenosu dat mezi databází a dokumenty nebo sériovým portem. Tento přenos dat je Import a export textových (ASCII), DIF a SYLK souborů. ASCII mapa pracuje také při přenosech při příkazech SEND PACKET, RECEIVE PACKET a RECEIVE BUFFER. Nemá ale žádný efekt na data přenášená pomocí SEND RECORD, SEND VARIABLE, RECEIVE RECORD a RECEIVE VARIABLE.

Pokud předáte prázdný řetězec do parametru mapa, USE ASCII MAP zobrazí standardní okno Otevřít soubor a můžete vybrat ASCII mapu ručně. Když použijete příkaz USE ASCII MAP a mapa je správně načtena je proměnná OK nastavena na 1, pokud z nějakého důvodu mapa není načtena je OK nastavena na 0.

Druhý způsob použití příkazu USE ASCII MAP je s parametrem * místo mapy. Pokud použijete tento způsob, 4th Dimension použije zpět původní ASCII mapu. Pokud je *mapVstVýst* 0, je přeřazena výstupní mapa a pokud je tento parametr 1, je přestavena vstupní mapa. Při použití výchozí ASCII mapy nebudou prováděny žádné převody znaků.

Příklad

Následující příklad použije speciální ASCII mapu a po exportu dat opět nastaví výchozí ASCII mapu.

```
USE ASCII MAP ("MacdoPC"; 0) ` Načte alternativní ASCII mapu
EXPORT TEXT ([MaTabulka]; "MujText") ` Exportuje data přes mapu
USE ASCII MAP (*; 0) ` Nastaví zpět výchozí ASCII mapu
```

Dále si přečtěte

EXPORT DIF, EXPORT SYLK, EXPORT TEXT, IMPORT DIF, IMPORT SYLK, IMPORT TEXT, Mac to Win, RECEIVE BUFFER, RECEIVE PACKET, SEND PACKET, Win to Mac.

Příkazy a odkazy pro Komunikace

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SEND PACKET (POSLAT DATOVÝ BALÍK)

Příkazy a odkazy pro Komunikace

Verze 3

SEND PACKET ({odkDok;} balík)

Parametr	Typ		Popis
odkDok	DocRef	→	Číslo odkazu k dokumentu nebo Platný kanál (sériový port nebo dokument)
balík	Řetězec	→	Řetězec nebo text k poslání

Popis

SEND PACKET pošle balík do sériového portu nebo dokumentu. Pokud je předán *okdDok*, je balík poslán do dokumentu k němuž je odkaz v *odkDok*. Pokud není tento parametr předán je balík poslán do portu nebo dokumentu, který byl otevřen pomocí SET CHANNEL. Balík je pouze kousek dat, většinou řetězec znaků.

Před tím, než použijete SEND PACKET, musíte otevřít port nebo dokument pomocí příkazu SET CHANNEL nebo pomocí jiného příkazu k otevření dokumentu.

Při psaní do dokumentu začne první požití SEND PACKET psát na začátek dokumentu, pokud nebyl dokument otevřen pomocí příkazu Append document. Dokud dokument nezavřete, jsou všechny následné balíky připojovány za již přidané.

Poznámka k Verzi 6: Tento příkaz je stále použitelný s příkazem SET CHANNEL. Jinak můžete pro dokumenty otevřené pomocí Open document, Create document a Append document k získání a změně umístění v dokumentu, kam bude text psán (SEND PACKET) a nebo čten (RECEIVE PACKET), použít nové příkazy Get document position a SET DOCUMENT POSITION .

Důležité: SEND PACKET zapisuje v MacOs ASCII mapě na Windows Macintosh. Každá z nich používá osm bitů. Standardní ASCII mapa používá pouze spodních 7 bitů. Mnoho zařízení nepoužívá osmý bit stejným způsobem jako Windows/Macintosh. Jestliže poslaný řetězec obsahuje osmý bit, ujstěte se, že jste vytvořili ASCII mapu k převodu ASCII znaků a před použitím SEND PACKET spustili příkaz USE ASCII MAP. Protokoly jako XON/XOFF používají nějaké nižší ASCII kódy k vytvoření spojení mezi počítači. Buďte proto opatrní aby jste neposílali znaky které by mohli kolodovat s protokolem nebo dokonce přerušit komunikaci.

Příklad

Následující příklad uloží data z polí do dokumentu. Zapíše pole jako pole s pevnou délkou. Pole s pevnou délkou mají vždy stejnou délku. Pokud pole má méně znaků, je místo doplněno mezerami. (tj. proto, aby bylo dosaženo požadované délky.) I přes to, že pole s pevnou délkou nejsou příliš účinný způsob ukládání dat, některé počítačové systémy a aplikace jej stále používají:

```
$vhDokumRef := Create document ("") ` Vytvoří dokument
If (OK=1) ` Byl dokument vytvořen?
   For ($vlZaznam; 1; Records in selection ([Lidé])) ` Jedna smyčka na záznam
         ` Poslat balík. Vytvoří balík z řetězce o 15 znacích
         ` obsahující pole jméno
      SEND PACKET ($vhDokumRef; Change string(15 * Char(Space); [Lidé]Jméno;1))
         ` Pošle druhý balík. Vytvoří balík z řetězce o 15 znacích
         ` obsahující pole příjmení
         ` Toto může být již v prvním SEND PACKET, ale je odděleno pro přehlednost
      SEND PACKET ($vhDokumRef; Change string (15 * Char(Space); [Lidé]Příjmení; 1))
      NEXT RECORD([Lidé])
```

```
    End for
        ` Poslat Char(26), který je použitý jako end-of-file oddělovač pro některé počítače
    SEND PACKET ($vhDokumRef; Char(SUB ASCII Code))
    CLOSE DOCUMENT ($vhDokumRef) ` Zavřít dokument
End if
```

Dále si přečtěte

Get document position, RECEIVE PACKET, SET DOCUMENT POSITION.

Příkazy a odkazy pro Komunikace

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

RECEIVE PACKET (NAČÍST DATOVÝ BALÍK)

Příkazy a odkazy pro Komunikace

Verze 3

RECEIVE PACKET ({odkDok;} dplProm; stopZnak | počZnak)

Parametr	**Typ**	**Popis**
okdDok	DocRef	Číslo odkazu k dokumentu nebo není-li pak Platný kanál (sériový port nebo dokument)
dplProm	Proměnná	Proměnná k načtení dat
stopZnak | počZnaků	Řetězec | Číslo	Znak na kterém zastavit načítání nebo počet znaků k přečtení

Popis

RECEIVE PACKET načte znaky z portu nebo z dokumentu.

Pokud je definován *odkDok*, tento příkaz načte znaky z dokumentu otevřeného pomocí Open document, Create document nebo a Append document. Pokud je tento parametr vynechán, přečte tento příkaz znaky ze sériového portu nebo dokumentu otevřeného pomocí SET CHANNEL.

Ať už je zdroj jakýkoli, jsou znaky načteny do *dplProm*, což musí být Textová nebo Řetězcová proměnná. Pokud chcete načíst pouze určitý počet znaků, zadejte tento počet do parametru *počZnaků*. Pokud chcete načítat dokud se neobjeví určitý znak, zadejte tento znak do proměnné *stopZnak* (tento znak není při čtení fyzicky přidán do proměnné *dplProm*).

Při čtení dokumentu, pokud nejsou definovány parametry *stopZnak* či *počZnaků*, bude příkaz načítat znaky dokud nedojde na konec dokumentu. Nezapomeňte, že zatímco proměnná typu Řetězec má pevnou délku, proměnná typu Text může obsahovat až 32000 znaků. Při čtení ze sériového portu bude RECEIVE PACKET čekat dokud nevyprší časový limit (pokud je nějaký - SET TIMEOUT) nebo dokud uživatel nepřeruší příjem (čtěte dále).

Během provádění RECEIVE PACKET může uživatel přerušit příjem stisknutím Ctrl-Alt-Shift (Windows) nebo Command-Option-Shift (Macintosh). Toto přerušení generuje chybu -9994 kterou můžete zachytit pomocí metody s ON ERR CALL. Obvykle přerušujete příjem pouze pokud se přeruší komunikace na sériovém portu.

Při čtení dokumentu, začne první příkaz RECEIVE PACKET číst na začátku dokumentu. Při každém dalším čtení z dokumentu začne příkaz číst na znaku který následuje za posledním přečteným.

Poznámka k Verzi 6: Tento příkaz je stále použitelný s příkazem SET CHANNEL. Jinak můžete pro dokumenty otevřené pomocí Open document, Create document a Append document k získání a změně umístění v dokumentu, kam bude text psán (SEND PACKET) a nebo čten (RECEIVE PACKET), použít nové příkazy Get document position a SET DOCUMENT POSITION.

Při pokusu o čtení za koncem souboru, RECEIVE PACKET vrátí koncový bod čtení těsně před znak konce souboru a nastaví proměnnou OK na 1. Následující RECEIVE PACKET pak vrátí prázdný řetězec a nastaví proměnnou OK na 0.

Příklady

1. Následující príklad načte prvních dvacet znaků ze sériového portu do proměnné vZískatDvacet:

```
RECEIVE PACKET (vZískatDvacet; 20)
```

2. Následující příklad načte do proměnné vData data z dokumentu odkaz na nějž je v proměnné myDoc. Bude načítat tak dlouho dokud nenarazí na znak Return (ascii 13):

```
RECEIVE PACKET (myDoc;vData;Char (Carriage Return))
```

3. Následující příklad načte data z dokumentu do pole. Data jsou uložena jako pole s pevnou délkou. Metoda vyvolá

podprogram k odstranění případných mezer (mezery na konci řetězce). Podprogram následuje za metodou:

```
$vhDokumRef := Open document ("";"TEXT") ` Otevřít TEXTový dokument
If (OK=1) ` Pokud byl dokument otevřen
   Repeat ` Provádět smyčku dokud budou nějaká data
      RECEIVE PACKET ($vhDokumRef; $Var1; 15) ` Načíst 15 znaků
      RECEIVE PACKET ($vhDokumRef; $Var2; 15) ` udělat to samé pro druhé pole
      If (OK = 1) ` Pokud nejsme na konci dokumentu
         CREATE RECORD([Lidé]) ` Vytvořit nový záznam
         [Lidé]Jméno := VynechatMez ($Var1) ` Uložit jméno
         [Lidé]Příjmení := VynechatMez ($Var2) ` Uložit příjmení
         SAVE RECORD([Lidé]) ` Uložit záznam
      End if
   Until (OK =0)
   CLOSE DOCUMENT ($vhDokumRef) ` Uzavřít dokument
End if
```

Mezery na konci dat jsou odstraněny následující metodou nazvanou VynechatMez:

```
For ($i; Length ($1); 1; -1) ` Smyčka z konce řetězce na začátek
   If ($1[[$i]] # " ") ` Pokud to není mezera…
      $i := -$i ` Posunout smyčku ke konci
   End if
End for
$0 := Delete string ($1; -$i; Length ($1)) ` Vymazat mezery
```

Dále si přečtěte

Get document position, RECEIVE PACKET, SEND PACKET, SET DOCUMENT POSITION, SET TIMEOUT.

Systémové proměnné

Po provedení příkazu je proměnná OK nastavena na 1 pokud byl balík správně načten, jinak je nastavena na 0.

Příkazy a odkazy pro Komunikace

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

RECEIVE BUFFER (PŘIJMOUT BUFFER)

Příkazy a odkazy pro Komunikace

Verze 3

RECEIVE BUFFER (doplProm)

Parametr	Typ		Popis
doplProm	Proměnná	→	Proměnná k uložení dat
		←	Přijatá data v proměnné

Popis

Příkaz RECEIVE BUFFER načte sériový port, který byl předtím otevřen pomocí SET CHANNEL. Sériový port má buffer který se vyplní znaky dokud příkaz tento buffer nepřečte. RECEIVE BUFFER načte znaky z bufferu sériového portu, uloží je do *doplProm* a pak buffer vymaže. Pokud v bufferu nejsou žádné znaky, pak nebude parametr *doplProm* obsahovat nic.

Na Windows:

Buffer sériového portu má omezenou velikost. To znamená, že buffer se může přeplnit. Pokud je plný a ještě stále dostává nové znaky, budou tyto znaky nahrazovat již existující. Nahrazené znaky budou ztraceny; proto je důležité při přijímání znaků do bufferu načítat znaky rychle.

Na Macintoshi

Buffer sériového portu má velikost 64 znaků. To znamená, že může obsahovat maximálně 64 znaků. Pokud je plný a ještě stále dostává nové znaky, budou tyto znaky nahrazovat již existující. Nahrazené znaky budou ztraceny; ; proto je důležité při přijímání znaků do bufferu načítat znaky rychle.

Poznámka: Existují plug-iny 4d které vám umožní zvětšit velikost bufferu.

RECEIVE BUFFER je odlišný od RECEIVE PACKET v tom, že vezme cokoli je v bufferu a ihned to přesune. RECEIVE PACKET čeká dokud nenajde zadaný znak, nebo dokud nenačte určitý počet znaků.

Během provádění RECEIVE BUFFER může uživatel přerušit příjem stisknutím Ctrl-Alt-Shift (Windows) nebo Command-Option-Shift (Macintosh). Toto přerušení generuje chybu -9994 kterou můžete zachytit pomocí metody s ON ERR CALL.

Příklad

Metoda projektu POSLOUCHAT SERIOVY PORT používá RECEIVE BUFFER k načtení textu ze sériového portu a ukládá jej do meziprocesové proměnné:

```
   ` POSLOUCHAT SERIOVY PORT
While (<>IP_Listen_Serial_Port)
   RECEIVE BUFFER($vtBuffer)
   If ((Length($vtBuffer)+Length(<>vtBuffer))>MAXTEXTLEN)
      <>vtBuffer:=""
   End if
   <>vtBuffer:=<>vtBuffer+$Buffer
End while
```

Tato metoda může být spuštěna jako metoda procesu pro místní proces:

```
   ` začít poslouchat sériový port
SET CHANNEL (201; Speed 9600 + Data Bits 8 + Stop Bits One+ Parity None)      ` Otevřít sériový port
```

```
<>IP_Listen_Serial_Port:=True
$vlSerialPID:=New process(" POSLOUCHAT SERIOVY PORT ";16*1024;"$Posluchac Serioveho")
```

V tomto okamžiku může jakýkoli jiný proces číst meziprocesovou proměnnou <>vtBuffer a pracovat s daty přicházejícími z portu.

K zastavení poslechu sériového portu stačí spustit:

```
` Zastavení poslechu sériového portu
<>IP_Listen_Serial_Port:=False
```

Všiměte si, že meziprocesová proměnná <>vtBuffer může být chráněna semaforem tak, že proces nebude konfliktní. Pro více informací se podívejte na příkaz Semaphore.

Dále si přečtěte

ON ERR CALL, RECEIVE PACKET, Semaphore, SET CHANNEL, Proměnné.

Příkazy a odkazy pro Komunikace

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SEND VARIABLE (POSLAT PROMĚNNOU)

Verze 3

SEND VARIABLE (proměnná)

Parametr	Typ		Popis
proměnná	Proměnná	→	Proměnná k poslání

Popis

Příkaz SEND VARIABLE pošle proměnnou do dokumentu nebo sériového portu, který byl otevřen pomocí SET CHANNEL. Proměnná je poslána ve speciálním formátu 4D a může být přečtena pouze pomocí příkazu RECEIVE VARIABLE. SEND VARIABLE pošle kompletní proměnnou (typ a hodnota).

Poznámky

1. Pokud pošlete proměnnou do dokumentu pomocí tohoto příkazu, musí být dokument otevřen pomocí příkazu SET CHANNEL. Nemůžete použít SEND VARIABLE pro dokumenty otevřené pomocí Open document, Append document nebo Create document.

2. Tento příkaz nepřijímá proměnné typů array. Pokud chcete poslat nebo načíst array, použijte nové příkazy pro BLOB obsažené ve verzi 6, nebo je odešlete po prvcích.

Příklad

Podívejte se na příklad u příkazu RECEIVE RECORD.

Dále si přečtěte

RECEIVE RECORD, RECEIVE VARIABLE, SEND RECORD, SET CHANNEL.

Příkazy a odkazy pro Komunikace

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

RECEIVE VARIABLE (NAČÍST PROMĚNNOU)

Příkazy a odkazy pro Komunikace

Verze 3

RECEIVE VARIABLE (proměnná)

Proměnná	Typ		Popis
proměnná	Proměnná	→	Proměnná do které se má ukládat

Popis

RECEIVE VARIABLE načte proměnnou, která byla před tím uložena pomocí SEND VARIABLE, z dokumentu nebo sériového portu, který byl otevřen příkazem SET CHANNEL.

Ve interpretačním módu, jestliže nebyla proměnná před voláním příkazu RECEIVE VARIABLE deklarována, je vytvořena, zapsána a je jí předána hodnota podle toho co je do ní načítáno. Ve zkompilované databázi musí být proměnná stejného typu jako data která jsou do ní načítána, obvykle řetězec. V obou případech je obsah proměnné nahrazen tím co je načteno.

Poznámky

1. Pokud načtete proměnnou z dokumentu pomocí tohoto příkazu, musí být dokument otevřen pomocí příkazu SET CHANNEL. Nemůžete použít RECEIVE VARIABLE pro dokumenty otevřené s Open document, Append document nebo Create document.

2. Tento příkaz nepřijímá proměnné typů array. Pokud chcete poslat nebo načíst array, použijte nové příkazy pro BLOB obsažené ve verzi 6.

3. Během provádění RECEIVE VARIABLE může uživatel přerušit příjem stisknutím Ctrl-Alt-Shift (Windows) nebo Command-Option-Shift (Macintosh). Toto přerušení generuje chybu -9994 kterou můžete zachytit pomocí metody s ON ERR CALL. Obvykle přerušujete příjem pouze pokud se přeruší komunikace na sériovém portu.

Příklad

Podívejte se na příklad u příkazu RECEIVE RECORD.

Dále si přečtěte

ON ERR CALL, RECEIVE RECORD, SEND RECORD, SEND VARIABLE.

Systémové proměnné a Sady

Pokud je proměnná správně načtena, je proměnná OK nastavena na 1, jinak je nastavena na 0.

Příkazy a odkazy pro Komunikace

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SEND RECORD (POSLAT ZÁZNAM)

Verze 3

SEND RECORD ({tabulka})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ze které poslat platný záznam. Pokud je vynechán, pak použít platnou tabulku.

Popis

SEND RECORD pošle platný záznam tabulky do sériového portu nebo dokumentu, který byl otevřen pomocí SET CHANNEL. Záznam je poslán ve specielním formátu 4D a může být přečten pouze pomocí příkazu RECEIVE RECORD. Pokud není žádný platný záznam, příkaz SEND RECORD neprovede nic.

Je poslán kompletní záznam. To znamená že budou poslány všechny podzáznamy, obrázky a BLOBy které jsou obsaženy v záznamu.

Důležité: Pokud jsou záznamy poslány a načteny pomocí SEND RECORD a RECEIVE RECORD, musí být cílová a zdrojová struktura tabulky kompatibilní. Pokud není, 4D převede hodnoty podle platné definice tabulky při provedení RECEIVE RECORD.

Poznámka: Pokud pošlete záznam do dokumentu pomocí tohoto příkazu, musí být dokument otevřen pomocí příkazu SET CHANNEL. Nemůžete tento příkaz použít na dokumenty otevřené pomocí Open document, Append document nebo Create document.

Příklad

Podívejte se na příklad u příkazu RECEIVE RECORD.

Dále si přečtěte

RECEIVE RECORD, RECEIVE VARIABLE, SEND VARIABLE.

Příkazy a odkazy pro Komunikace

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

RECEIVE RECORD (NAČÍST ZÁZNAM)

Příkazy a odkazy pro Komunikace

Verze 3

RECEIVE RECORD ({tabulka})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka do které se načte záznam. Pokud je vynechán, pak použít platnou tabulku.

Popis

RECEIVE RECORD načte záznam do tabulky ze sériového portu nebo dokumentu, který byl otevřen pomocí SET CHANNEL. Záznam musí být poslán příkazem SEND RECORD. Při spuštění příkazu RECEIVE RECORD se automaticky vytvoří nový záznam a po načtení záznamu musíte použít příkaz SAVE RECORD pro uložení nového záznamu.

Je načten kompletní záznam. To znamená že budou načteny všechny podzáznamy, obrázky a BLOBy které jsou obsaženy v záznamu.

Důležité: Pokud jsou záznamy poslány a načteny pomocí SEND RECORD a RECEIVE RECORD musí být cílová a zdrojová struktura tabulky kompatibilní. Pokud není, 4D převede hodnoty podle platné definice tabulky při provedení RECEIVE RECORD.

Poznámky

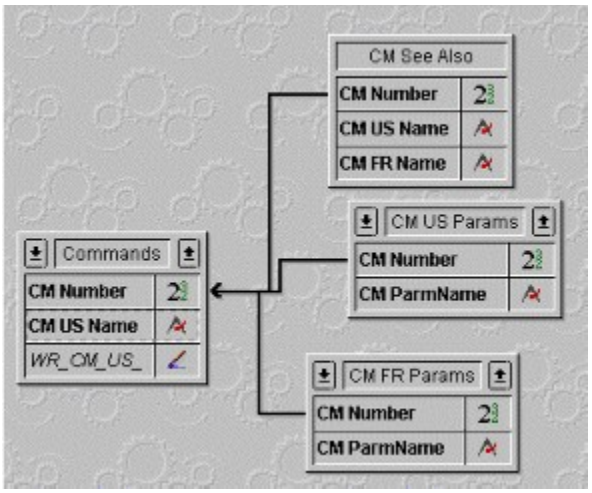
1. Pokud načtete záznam z dokumentu nebo sériového portu pomocí tohoto příkazu, musí být temto otevřen pomocí příkazu SET CHANNEL. Nemůžete tento příkaz použít na dokumenty otevřené pomocí Open document, Append document nebo Create document.

2. Během provádění RECEIVE RECORD může uživatel přerušit příjem stisknutím Ctrl-Alt-Shift (Windows) nebo Command-Option-Shift (Macintosh). Toto přerušení generuje chybu -9994 kterou můžete zachytit pomocí metody s ON ERR CALL. Obvykle přerušujete příjem pouze pokud se přeruší komunikace na sériovém portu.

Příklad

Kombinované použití příkazů SEND VARIABLE, SEND RECORD, RECEIVE VARIABLE a RECEIVE RECORD je ideální pro archivování dat nebo pro výměnu dat mezi jednouživatelskými databázemi používanými na různých místech. Data mezi databázemi 4D můžete přenášet pomocí export/import příkazů jako EXPORT TEXT a IMPORT TEXT. Jestliže vaše data obsahují grafiku a/nebo vztažené tabulky je použití RECEIVE RECORD a SEND RECORD daleko výhodnější.

Například dokumentace kterou čtete byla vytvořena pomocí 4D a 4D Write. Protože mnoho lidí jí psalo na mnoha místech na světě, potřebovali jsme jednoduchý způsob pro přenos dat mezi databázemi. Zde je jednoduchý náhled na strukturu databáze:



Tabulka [Příkazy] obsahuje popis každého příkazu nebo tématu. Tabulky [CM US Params] a [CM FR Params] obsahují seznam parametrů v Angličtině a Francouzštině. Tabulka [CM See Also] obsahuje příkazy zobrazené jako odkazy (sekce Dále si přečtěte) pro každý příkaz. Výměna dokumentace mezi databázemi je pouhé poslání záznamu [Příkazy] a jeho vztažených záznamů. Aby jsme to udělali, použijeme SEND RECORD a RECEIVE RECORD. Dále použijeme SEND VARIABLE a RECEIVE VARIABLE pro označení import/export dokumentu makrooddělovači a odkazy, tagy.

Zde je (zjednodušená) metoda projektu pro export dokumentace:

```
    ` Metoda projektu CM_EXPORT_SEL
    ` Tato metoda pracuje s platným výběrem tabulky [Příkazy]
SET CHANNEL (12;"") ` Umožnit uživateli otevřít dokument
If (OK=1)
        ` Označit dokument proměnnou, která udá jeho obsah
        ` Poznámka: proměnná procesu JAZYK_MOTORUznačí
        ` zda jsou posílána US (Anglická) nebo FR (Francouzská) data
   $vsTag:="4DV6COMMAND"+Jazyk_motoru
   SEND VARIABLE($vsTag)
      ` Poslat proměnnou, která označí, kolik záznamů [Příkazy] bylo posláno
   $vlNbCmd:=Records in selection([Příkazy])
   SEND VARIABLE($vlNbCmd)
   FIRST RECORD([Příkazy])
      ` Pro každý příkaz
   For ($vlCmd;1;$vlNbCmd)
         ` Poslat záznam [Příkazy]
      SEND RECORD([Příkazy])
         ` Vybrat všechny vztažené záznamy
      RELATE MANY([Příkazy])
         ` Podle jazyka, poslat proměnnou označující
         ` počet parametrů které budou následovat
      Case of
         ÷ (Jazyk_motoru="US")
            $vlNbParm:=Records in selection([CM US Params])
         ÷ (Jazyk_motoru="FR")
            $vlNbParm:=Records in selection([CM FR Params])
      End case
      SEND VARIABLE($vlNbParm)
         ` Poslat parametr záznamů (pokud nějaký)
      For ($vlParm;1;$vlNbParm)
```

```
            Case of
                ÷ (Jazyk_motoru="US")
                    SEND RECORD([CM US Params])
                    NEXT RECORD([CM US Params])
                ÷ (Jazyk_motoru="FR")
                    SEND RECORD([CM FR Params])
                    NEXT RECORD([CM FR Params])
            End case
        End for
            ` Poslat proměnnou označující kolik "Dále si přečtěte" bude následovat
        $vlNbSee:=Records in selection([CM See Also])
        SEND VARIABLE($vlNbSee)
            ` Poslat [See Also] záznamy (pokud nějaké)
        For ($vlSee;1;$vlNbSee)
            SEND RECORD([CM See Also])
            NEXT RECORD([CM See Also])
        End for
            ` Jít na další [Příkazy] záznam a pokračovat v exportu
        NEXT RECORD([Příkazy])
    End for
    SET CHANNEL(11) ` Zavřít dokument
End if
```

Zde je (zjednodušená) metoda projektu k importování dokumentace:

```
    ` Metoda projektu CM_IMPORT_SEL
SET CHANNEL(10;"") ` Umožnit uživateli otevřít existující dokument
If (OK=1) ` Pokud byl dokument otevřen
    RECEIVE VARIABLE($vsTag) ` Zkusit načíst proměnnou označení
    If ($vsTag="4DV6COMMAND@") ` Dostali jsme správný tag?
            ` Získat jazyk z tagu
        $CurLang:=Substring($vsTag;Length($vsTag)-1)
        If (($CurLang="US") | ($CurLang="FR")) ` Dostali jsme správný záznam
                ` Kolik příkazů je v dokumentu?
            RECEIVE VARIABLE($vlNbCmd)
            If ($vlNbCmd>0) ` Pokud je alespoň jeden
                For ($vlCmd;1;$vlNbCmd) ` Pro každý uložený záznam [Příkazy]
                        ` Načíst záznam
                    RECEIVE RECORD([Příkazy])
                        ` Zavolá podprogram který uloží nový záznam nebo zkopíruje
                        ` jeho hodnotu do existujícího záznamu
                    CM_IMP_CMD ($CurLang)
                        ` Načíst počet parametrů (pokud jsou nějaké)
                    RECEIVE VARIABLE($vlNbParm)
                    If ($vlNbParm>=0)
                            ` Zavolá podprogram který provede RECEIVE RECORD a pak
                            ` uloží nový záznam nebo jej zkopíruje do existujíxího záznamu
                        CM_IMP_PARM ($vlNbParm;$CurLang)
                    End if
                        ` Načte počet "See Also" (pokud nějaké)
                    RECEIVE VARIABLE($vlNbSee)
                    If ($vlNbSee>0)
```

```
                        ` Zavolá podprogram který provede RECEIVE RECORD a pak
                        ` uloží nový záznam nebo jej zkopíruje do existujícího záznamu
                    CM_IMP_SEEA ($vlNbSee;$CurLang)
                End if
            End for
        Else
            ALERT("Počet příkazů v exportu je neplatný.")
        End if
    Else
        ALERT("Jazyk tohoto exportu je neznámý.")
    End if
Else
    ALERT("Tento dokument není export příkazů.")
End if
SET CHANNEL(11) ` Zavřít dokument
End if
```

Všiměte si, že netestujeme proměnnou OK při načítání dat ani nezachycujeme chyby. Nicméně, protože v dokumentu ukládáme proměnné, které popisují samotný dokument a jestliže tyto proměnné, byly jednou načteny, budou správně a pravděpodobnost že vznikne nějaká další chyba je velmi nízká. Pokud například uživatel otevře špatný dokument, první test jej zastaví.

Dále si přečtěte

RECEIVE VARIABLE, SEND RECORD, SEND VARIABLE.

Systémové proměnné a Sady

Proměnná OK je nastavena na 1 pokud byl záznam správně načten. Jinak je systémová proměnná OK nastavena na 0.

Příkazy a odkazy pro Komunikace

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Příkazy kompilátoru

Příkazy a odkazy pro Kompilátor

Verze 6.0

4D Compiler překládá vaše databázové aplikace do strojového kódu. Možnosti 4D Compileru jsou:

Rychlost: Vaše databáze bude 3 až 1000 krát rychlejší.
Testování kódu: Vaše databáze je prohlédnuta pro správnost kódu. Jsou zachyceny jak logické tak syntactické chyby.
Ochrana: Zkompilovaná databáze je naprosto identická s nezkompilovanou pouze s tím rozdílem že struktura a metody nemohou být prohlíženy nebo měněny. Kompilace databáze zaručuje bezpečnost.
Samostatné poklepatelné aplikace: 4D Compiler vytváří samostatné aplikace (.EXE soubory) s jejich vlastní ikonou.

Příkazy v této kapitole jsou vztaženy k použití kompilátorem. Umožní vám normalizovat datové typy v databázi. Příkaz IDLE je specificky použitý ve zkompilované databázi.

C_BLOB	C_INTEGER	C_REAL	IDLE
C_BOOLEAN	C_LONGINT	C_STRING	
C_DATE	C_PICTURE	C_TEXT	
C_GRAPH	C_POINTER	C_TIME	

Tyto příkazy, mimo IDLE, deklarují proměnné a přidělují jim funkci specifických typů dat. Definování proměnných řeší dvojznačnost týkající se typů dat proměnných. Pokud není proměnná definována jedním z těchto příkazů, kompilátor se pokusí určit typ dat proměnné. Pro kompilátor je složité určit typy dat pro proměnné použité ve formuláři. Proto je důležité použít tyto příkazy pro deklarování proměných použitých ve formuláři.

Číselné operace na long integer a integer jsou obvykle daleko rychlejší než operace s výchozím číselným typem (real).

Základní pravidla pro psaní kódu který bude kompilován

• Nepřímá deklarace proměnných jak je použita ve 4th Dimension verze 1 není povolena. Nemůžete použít alfa nepřímé odkazy se symbolem sekce (§) k nepřímému odkazu na proměnnou. Stejně tak nemůžete použít číselnou nepřímé odkazy se složenými závorkami ({...}) z tohoto důvodu. Složené závorky mohou být použity pouze ke dostupu na prvek v array který byl již vytvořen. Nicméně můžete použít přesměrování parametrů jak je popsáno v příručce ke 4D Compiler.

• Nemůžete měnit typ dat žádné proměnné nebo array.

• Nemůžete měnit dvourozměrný array na normální a naopak.

• Nemůžete měnit délku řetězce nebo prvků v řetězcovém array.

• Ačkoli 4D copmiler pro vás zapíše typy proměnných, můžete specifikovat typ proměnných s použitím možností kompilátoru pro místa, kde datové typy jsou dvojznačné.

• Další důvod pro definování proměnných je optimalizace vašeho kódu. Toto pravidlo platí specielně pro proměnné použité jako čítače. Pro maximální rychlost pouižívejte proměnné typu long integer.

• K vymazání proměnné (nastavení na nulu) použijte příkaz CLEAR VARIABLE s názvem proměnné. V příkazu CLEAR VARIABLE nepoužívejte řetězec k znázornění názvu proměnné.

• Funkce Undefined musí vždy vracet False. Proměnné jsou vždy definované.

Příklady

1. Následující jsou základní definice proměnných pro 4D Compiler:

```
   ` Proměnná procesu vxMyBlob je definovaná jako proměnný typu BLOB
C_BLOB(vxMyBlob)
   ` Meziprocesováproměnná <>OnWindows je definovaná jako proměnná typu Logická
C_BOOLEAN(<>OnWindows)
   ` Místní proměnná $vdCurDate je definovaná jako proměnná typu Date
C_DATE($vdCurDate)
   ` 3 proměnné procesu vg1, vg2 a vg3 jsou definované jako proměnné typu Diagram
C_GRAPH(vg1;vg2;vg3)
```

2. V následujícím příkladu si metoda projektu vytvoří 3 parametry:

```
    ` Metoda projektu OneMethodAmongOthers
   ` OneMethodAmongOthers ( Real ; Integer    {; Long}    )
   ` OneMethodAmongOthers ( Amount ; Percentage    {; Ratio}    )
C_REAL($1) ` Parametr typu Real
C_INTEGER($2) ` Parametr typu Integer
C_LONGINT($3) ` Parametr typu Long Integer
` ...:
```

3. V následujícím příkladu přijme metoda parametr řetězec a vrátí řetězec:

```
   ` Metoda projektu Capitalize
   ` Capitalize ( Řetězec ) → Řetězec
   ` Capitalize ( Výchozí řetězec ) → Zvětšený řetězec
C_STRING(255;$0;$1)
$0:=Uppercase(Substring($1;1;1))+Lowercase(Substring($1;2))
```

4. V následujícím příkladu přijímá metoda projektu časový parametr následovaný proměnným počtem textových parametrů:

```
   ` Metoda projektu SEND PACKETS
   ` SEND PACKETS ( Čas ; Text { ; Text2... ; TextN } )
   ` SEND PACKETS ( odkDok ; Data    {; Data2... ; DataN}    )
C_TIME ($1)
C_TEXT ($${2})
C_LONGINT ($vlPacket)
For ($vlPacket;2;Count parameters)
   SEND PACKET ($1;$${$vlPacket})
End for
```

5. V následujícím příkladu metoda projektu COMPILER_Param_Predeclare28 předefinuje syntaxi jiných metod projektu pro 4D Compiler:

```
   ` Metoda projektu COMPILER_Param_Predeclare28
   ` OneMethodAmongOthers ( Real ; Integer    {; Long } )
C_REAL(OneMethodAmongOthers;$1)
C_INTEGER(OneMethodAmongOthers;$2) ` ...
C_LONGINT(OneMethodAmongOthers;$3) ` ...
   ` Capitalize ( Řetězec ) → Řetězec
C_STRING(Capitalize;255;$0;$1)


   ` SEND PACKETS ( Čas ; Text {; Text2... ; TextN } )
```

```
C_TIME(SEND PACKETS;$1)
C_TEXT(SEND PACKETS;${2}) ` ...
```

Dále si přečtěte

C_BLOB, C_BOOLEAN, C_DATE, C_GRAPH, C_INTEGER, C_LONGINT, C_PICTURE, C_POINTER, C_REAL, C_STRING, C_TEXT, C_TIME, IDLE.

Příkazy a odkazy pro Kompilátor

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

C_BLOB

Příkazy a odkazy pro Kompilátor

Verze 6.0

C_BLOB ({metoda; }proměnná{; proměnná2;...;proměnnáN})

Parametr	Typ		Popis
metoda	Metoda	→	Volitelný název metody
proměnná	Proměnná nebo ${...}	→	Název proměnných k vytvoření

Popis

C_BLOB určí proměnnou *proměnná* jako proměnnou typu BLOB.

První způsob použití tohoto příkazu, kde volitelná metoda NENÍ předána, je pro deklaraci jakékoli procesové, meziprocesové nebo místní proměnné *proměnná*.

Poznámka: Tento způsob může být použit v nezkompilované databázi.

Druhý způsob, kde je použit volitelný parametr *metoda*, je pro předdefinování výsledku pro 4D Compiler a nebo pro předdefinování parametrů ($0, $1, $2 atd.) pro zadanou metodu. Použijte tento způsob k přeskočení počátečního natypování proměnných při kompilaci, ušetříte čas kompilace.

UPOZORNĚNÍ: Druhý způsob nemůže být použit v nezkompilavané databázi. Z tohoto důvodu umístěte tento zápis do metody, která se nebude provádět v nezkompilované databázi. Název metody musí začínat slovem "COMPILER."

Tip pro rozšíření: Zápis C_BLOB (${...}) vám umožní vytvořit proměnný počet parametrů stejného typu pod podmínkou že toto jsou poslední parametry pro metodu. Například zápis C_BLOB (${5}) řekne 4D a 4D Compileru že od pátého parametru budou všechny parametry stejného typu a je jedno kolik jich bude. Pro další informace si přečtěte příkaz Count parameters.

Příklady

Podívejte se na příklad u části " Příkazy kompilátoru.

Dále si přečtěte

Příkazy kompilátoru.

Příkazy a odkazy pro Kompilátor

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

C_BOOLEAN

Příkazy a odkazy pro Kompilátor

Verze 3

C_BOOLEAN ({metoda; }proměnná{; proměnná2;...;proměnnáN})

Parametr	Typ		Popis
metoda	Metoda	→	Volitelný název metody
proměnná	Proměnná nebo ${...}	→	Název proměnných k vytvoření

Popis

C_BOOLEAN určí proměnnou *proměnná* jako proměnnou typu logická.

První způsob použití tohoto příkazu, kde volitelná *metoda* NENÍ předána, je pro deklaraci jakékoli procesové, meziprocesové nebo místní proměnné.

Poznámka: Tento způsob může být použit v nezkompilované databázi.

Druhý způsob, kde je použit volitelný parametr *metoda*, je pro předdefinování výsledku pro 4D Compiler a nebo pro předdefinování parametrů ($0, $1, $2 atd.) pro zadanou metodu. Použijte tento způsob k přeskočení počátečního natypování proměnných při kompilaci, ušetříte čas kompilace.

UPOZORNĚNÍ: Druhý způsob nemůže být použit v nezkompilavané databázi. Z tohoto důvodu umístěte tento zápis do metody, která se nebude provádět v nezkompilované databázi. Název metody musí začínat slovem "COMPILER."

Tip pro rozšíření: Zápis C_BOOLEAN (${...}) vám umožní vytvořit proměnný počet parametrů stejného typu pod podmínkou že toto jsou poslední parametry pro metodu. Například zápis C_BOOLEAN (${5}) řekne 4D a 4D Compileru že od pátého parametru budou všechny parametry stejného typu a je jedno kolik jich bude. Pro další informace si přečtěte příkaz Count parameters.

Příklady

Podívejte se na příklad u části " Příkazy kompilátoru".

Dále si přečtěte

Příkazy kompilátoru, Count parameters.

Příkazy a odkazy pro Kompilátor

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

C_DATE

Příkazy a odkazy pro Kompilátor

Verze 3

C_DATE ({metoda; }proměnná{; proměnná2;...;proměnnáN})

Parametr	Typ		Popis
metoda	Metoda	→	Volitelný název metody
proměnná	Proměnná nebo ${...}	→	Název proměnných k vytvoření

Popis

C_DATE určí proměnnou *proměnná* jako proměnnou typu datum.

První způsob použití tohoto příkazu, kde volitelná *metoda* NENÍ předána, je pro deklaraci jakékoli procesové, meziprocesové nebo místní proměnné.

Poznámka: Tento způsob může být použit v nezkompilované databázi.

Druhý způsob, kde je použit volitelný parametr *metoda*, je pro předdefinování výsledku pro 4D Compiler a nebo pro předdefinování parametrů ($0, $1, $2 atd.) pro zadanou metodu. Použijte tento způsob k přeskočení počátečního natypování proměnných při kompilaci, ušetříte čas kompilace.

UPOZORNĚNÍ: Druhý způsob nemůže být použit v nezkompilavané databázi. Z tohoto důvodu umístěte tento zápis do metody, která se nebude provádět v nezkompilované databázi. Název metody musí začínat slovem "COMPILER."

Tip pro rozšíření: Zápis C_DATE (${...}) vám umožní vytvořit proměnný počet parametrů stejného typu pod podmínkou že toto jsou poslední parametry pro metodu. Například zápis C_DATE (${5}) řekne 4D a 4D Compileru že od pátého parametru budou všechny parametry stejného typu a je jedno kolik jich bude. Pro další informace si přečtěte příkaz Count parameters.

Příklady

Podívejte se na příklad u části " Příkazy kompilátoru".

Dále si přečtěte

Příkazy kompilátoru, Count parameters.

Příkazy a odkazy pro Kompilátor

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

C_GRAPH

Příkazy a odkazy pro Kompilátor

Verze 3

C_GRAPH ({metoda; }proměnná{; proměnná2;...;proměnnáN})

Parametr	Typ		Popis
metoda	Metoda	→	Volitelný název metody
proměnná	Proměnná nebo ${...}	→	Název proměnných k vytvoření

Popis

C_GRAPH určí proměnnou *proměnná* jako proměnnou typu diagram.

První způsob použití tohoto příkazu, kde volitelná *metoda* NENÍ předána, je pro deklaraci jakékoli procesové, meziprocesové nebo místní proměnné.

Poznámka: Tento způsob může být použit v nezkompilované databázi.

Druhý způsob, kde je použit volitelný parametr *metoda*, je pro předdefinování výsledku pro 4D Compiler a nebo pro předdefinování parametrů ($0, $1, $2 atd.) pro zadanou metodu. Použijte tento způsob k přeskočení počátečního natypování proměnných při kompilaci, ušetříte čas kompilace.

UPOZORNĚNÍ: Druhý způsob nemůže být použit v nezkompilované databázi. Z tohoto důvodu umístěte tento zápis do metody, která se nebude provádět v nezkompilované databázi. Název metody musí začínat slovem "COMPILER."

Tip pro rozšíření: Zápis C_GRAPH (${...}) vám umožní vytvořit proměnný počet parametrů stejného typu pod podmínkou že toto jsou poslední parametry pro metodu. Například zápis C_GRAPH (${5}) řekne 4D a 4D Compileru že od pátého parametru budou všechny parametry stejného typu a je jedno kolik jich bude. Pro další informace si přečtěte příkaz Count parameters.

Příklady

Podívejte se na příklad u části " Příkazy kompilátoru".

Dále si přečtěte

Příkazy kompilátoru.

Příkazy a odkazy pro Kompilátor

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

C_INTEGER

Příkazy a odkazy pro Kompilátor

Verze 3

C_INTEGER ({metoda; }proměnná{; proměnná2;...;proměnnáN})

Parametr	Typ		Popis
metoda	Metoda	→	Volitelný název metody
proměnná	Proměnná nebo ${...}	→	Název proměnných k vytvoření

Popis

C_INTEGER určí proměnnou *proměnná* jako proměnnou typu integer.

První způsob použití tohoto příkazu, kde volitelná *metoda* NENÍ předána, je pro deklaraci jakékoli procesové, meziprocesové nebo místní proměnné.

Poznámka: Tento způsob může být použit v nezkompilované databázi.

Druhý způsob, kde je použit volitelný parametr *metoda*, je pro předdefinování výsledku pro 4D Compiler a nebo pro předdefinování parametrů ($0, $1, $2 atd.) pro zadanou metodu. Použijte tento způsob k přeskočení počátečního natypování proměnných při kompilaci, ušetříte čas kompilace.

UPOZORNĚNÍ: Druhý způsob nemůže být použit v nezkompilavané databázi. Z tohoto důvodu umístěte tento zápis do metody, která se nebude provádět v nezkompilované databázi. Název metody musí začínat slovem "COMPILER."

Tip pro rozšíření: Zápis C_INTEGER (${...}) vám umožní vytvořit proměnný počet parametrů stejného typu pod podmínkou že toto jsou poslední parametry pro metodu. Například zápis C_INTEGER (${5}) řekne 4D a 4D Compileru že od pátého parametru budou všechny parametry stejného typu a je jedno kolik jich bude. Pro další informace si přečtěte příkaz Count parameters.

Příklady

Podívejte se na příklad u části " Příkazy kompilátoru".

Dále si přečtěte

Příkazy kompilátoru, Count parameters, C_LONGINT, C_REAL.

Příkazy a odkazy pro Kompilátor

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

C_LONGINT

Příkazy a odkazy pro Kompilátor

Verze 3

C_LONGINT ({metoda; }proměnná{; proměnná2;...;proměnnáN})

Parametr	Typ		Popis
metoda	Metoda	→	Volitelný název metody
proměnná	Proměnná nebo ${...}	→	Název proměnných k vytvoření

Popis

C_LONGINT určí proměnnou *proměnná* jako proměnnou typu Long Integer.

První způsob použití tohoto příkazu, kde volitelná *metoda* NENÍ předána, je pro deklaraci jakékoli procesové, meziprocesové nebo místní proměnné.

Poznámka: Tento způsob může být použit v nezkompilované databázi.

Druhý způsob, kde je použit volitelný parametr *metoda*, je pro předdefinování výsledku pro 4D Compiler a nebo pro předdefinování parametrů ($0, $1, $2 atd.) pro zadanou metodu. Použijte tento způsob k přeskočení počátečního natypování proměnných při kompilaci, ušetříte čas kompilace.

UPOZORNĚNÍ: Druhý způsob nemůže být použit v nezkompilované databázi. Z tohoto důvodu umístěte tento zápis do metody, která se nebude provádět v nezkompilované databázi. Název metody musí začínat slovem "COMPILER."

Tip pro rozšíření: Zápis C_LONGINT (${...}) vám umožní vytvořit proměnný počet parametrů stejného typu pod podmínkou že toto jsou poslední parametry pro metodu. Například zápis C_LONGINT (${5}) řekne 4D a 4D Compileru že od pátého parametru budou všechny parametry stejného typu a je jedno kolik jich bude. Pro další informace si přečtěte příkaz Count parameters.

Příklady

Podívejte se na příklad u části " Příkazy kompilátoru".

Dále si přečtěte

Příkazy kompilátoru, Count parameters, C_INTEGER, C_REAL.

Příkazy a odkazy pro Kompilátor

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

C_PICTURE

Příkazy a odkazy pro Kompilátor

Verze 3

C_PICTURE ({metoda; }proměnná{; proměnná2;...;proměnnáN})

Parametr	Typ		Popis
metoda	Metoda	→	Volitelný název metody
proměnná	Proměnná nebo ${...}	→	Název proměnných k vytvoření

Popis

C_PICTURE určí proměnnou *proměnná* jako proměnnou typu obrázek.

První způsob použití tohoto příkazu, kde volitelná *metoda* NENÍ předána, je pro deklaraci jakékoli procesové, meziprocesové nebo místní proměnné.

Poznámka: Tento způsob může být použit v nezkompilované databázi.

Druhý způsob, kde je použit volitelný parametr *metoda*, je pro předdefinování výsledku pro 4D Compiler a nebo pro předdefinování parametrů ($0, $1, $2 atd.) pro zadanou metodu. Použijte tento způsob k přeskočení počátečního natypování proměnných při kompilaci, ušetříte čas kompilace.

UPOZORNĚNÍ: Druhý způsob nemůže být použit v nezkompilované databázi. Z tohoto důvodu umístěte tento zápis do metody, která se nebude provádět v nezkompilované databázi. Název metody musí začínat slovem "COMPILER."

Tip pro rozšíření: Zápis C_PICTURE (${...}) vám umožní vytvořit proměnný počet parametrů stejného typu pod podmínkou že toto jsou poslední parametry pro metodu. Například zápis C_PICTURE (${5}) řekne 4D a 4D Compileru že od pátého parametru budou všechny parametry stejného typu a je jedno kolik jich bude. Pro další informace si přečtěte příkaz Count parameters.

Příklady

Podívejte se na příklad u části " Příkazy kompilátoru".

Dále si přečtěte

Příkazy kompilátoru, Count parameters.

Příkazy a odkazy pro Kompilátor

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

C_POINTER

Příkazy a odkazy pro Kompilátor

Verze 3

C_POINTER ({metoda; }proměnná{; proměnná2;...;proměnnáN})

Parametr	Typ		Popis
metoda	Metoda	→	Volitelný název metody
proměnná	Proměnná nebo ${...}	→	Název proměnných k vytvoření

Popis

C_POINTER určí proměnnou *proměnná* jako proměnnou typu ukazatel.

První způsob použití tohoto příkazu, kde volitelná *metoda* NENÍ předána, je pro deklaraci jakékoli procesové, meziprocesové nebo místní proměnné.

Poznámka: Tento způsob může být použit nezkompilované databázi.

Druhý způsob, kde je použit volitelný parametr *metoda*, je pro předdefinování výsledku pro 4D Compiler a nebo pro předdefinování parametrů ($0, $1, $2 atd.) pro zadanou metodu. Použijte tento způsob k přeskočení počátečního natypování proměnných při kompilaci, ušetříte čas kompilace.

UPOZORNĚNÍ: Druhý způsob nemůže být použit v nezkompilované databázi. Z tohoto důvodu umístěte tento zápis do metody, která se nebude provádět v nezkompilované databázi. Název metody musí začínat slovem "COMPILER."

Tip pro rozšíření: Zápis C_POINTER (${...}) vám umožní vytvořit proměnný počet parametrů stejného typu pod podmínkou že toto jsou poslední parametry pro metodu. Například zápis C_POINTER (${5}) řekne 4D a 4D Compileru že od pátého parametru budou všechny parametry stejného typu a je jedno kolik jich bude. Pro další informace si přečtěte příkaz Count parameters.

Příklady

Podívejte se na příklad u části " Příkazy kompilátoru".

Dále si přečtěte

Příkazy kompilátoru, Count parameters.

Příkazy a odkazy pro Kompilátor

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

C_REAL

Příkazy a odkazy pro Kompilátor

Verze 3

C_REAL ({metoda; }proměnná{; proměnná2;...;proměnnáN})

Parametr	Typ		Popis
metoda	Metoda	→	Volitelný název metody
proměnná	Proměnná nebo ${...}	→	Název proměnných k vytvoření

Popis

C_REAL určí proměnnou *proměnná* jako proměnnou typu real.

První způsob použití tohoto příkazu, kde volitelná *metoda* NENÍ předána, je pro deklaraci jakékoli procesové, meziprocesové nebo místní proměnné.

Poznámka: Tento způsob může být použit nezkompilované databázi.

Druhý způsob, kde je použit volitelný parametr *metoda*, je pro předdefinování výsledku pro 4D Compiler a nebo pro předdefinování parametrů ($0, $1, $2 atd.) pro zadanou metodu. Použijte tento způsob k přeskočení počátečního natypování proměnných při kompilaci, ušetříte čas kompilace.

UPOZORNĚNÍ: Druhý způsob nemůže být použit v nezkompilavané databázi. Z tohoto důvodu umístěte tento zápis do metody, která se nebude provádět v nezkompilované databázi. Název metody musí začínat slovem "COMPILER."

Tip pro rozšíření: Zápis C_REAL (${...}) vám umožní vytvořit proměnný počet parametrů stejného typu pod podmínkou že toto jsou poslední parametry pro metodu. Například zápis C_REAL (${5}) řekne 4D a 4D Compileru že od pátého parametru budou všechny parametry stejného typu a je jedno kolik jich bude. Pro další informace si přečtěte příkaz Count parameters.

Příklady

Podívejte se na příklad u části " Příkazy kompilátoru".

Dále si přečtěte

Příkazy kompilátoru, Count parameters, C_INTEGER, C_LONGINT.

Příkazy a odkazy pro Kompilátor

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

C_STRING

Příkazy a odkazy pro Kompilátor

Verze 3

C_STRING ({metoda; }velikost; proměnná{; proměnná2;...;proměnnáN})

Parametr	Typ		Popis
metoda	Metoda	→	Volitelný název metody
velikost	Číslo	→	Velikost řetězce
proměnná	Proměnná nebo ${...}	→	Název proměnných k vytvoření

Popis

C_STRING určí proměnnou *proměnná* jako proměnnou typu řetězec.

Parametr velikost definuje maximální délku řetězce který může proměnná obsahovat. Řetězec je omezen na 255 znaků. Pokud je pro vás důležitá rychlost, používejte raději proměnné typu řetězec než typu text.

První způsob použití tohoto příkazu, kde volitelná *metoda* NENÍ předána, je pro deklaraci jakékoli procesové, meziprocesové nebo místní proměnné.

Poznámka: Tento způsob může být použit nezkompilované databázi.

Druhý způsob, kde je použit volitelný parametr *metoda*, je pro předdefinování výsledku pro 4D Compiler a nebo pro předdefinování parametrů ($0, $1, $2 atd.) pro zadanou metodu. Použijte tento způsob k přeskočení počátečního natypování proměnných při kompilaci, ušetříte čas kompilace..

UPOZORNĚNÍ: Druhý způsob nemůže být použit v nezkompilavané databázi. Z tohoto důvodu umístěte tento zápis do metody, která se nebude provádět v nezkompilované databázi. Název metody musí začínat slovem "COMPILER."

Tip pro rozšíření: Zápis C_STRING (${...}) vám umožní vytvořit proměnný počet parametrů stejného typu pod podmínkou že toto jsou poslední parametry pro metodu. Například zápis C_STRING (${5}) řekne 4D a 4D Compileru že od pátého parametru budou všechny parametry stejného typu a je jedno kolik jich bude. Pro další informace si přečtěte příkaz Count parameters.

Příklady

Podívejte se na příklad u části " Příkazy kompilátoru".

Dále si přečtěte

Příkazy kompilátoru, Count parameters, C_TEXT.

Příkazy a odkazy pro Kompilátor

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

C_TEXT

Příkazy a odkazy pro Kompilátor

Verze 3

C_TEXT ({metoda; }proměnná{; proměnná2;...;proměnnáN})

Parametr	Typ		Popis
metoda	Metoda	→	Volitelný název metody
proměnná	Proměnná nebo ${...}	→	Název proměnných k vytvoření

Popis

C_TEXT určí proměnnou *proměnná* jako proměnnou typu text.

První způsob použití tohoto příkazu, kde volitelná *metoda* NENÍ předána, je pro deklaraci jakékoli procesové, meziprocesové nebo místní proměnné.

Poznámka: Tento způsob může být použit v nezkompilované databázi.

Druhý způsob, kde je použit volitelný parametr *metoda*, je pro předdefinování výsledku pro 4D Compiler a nebo pro předdefinování parametrů ($0, $1, $2 atd.) pro zadanou metodu. Použijte tento způsob k přeskočení počátečního natypování proměnných při kompilaci, ušetříte čas kompilace.

UPOZORNĚNÍ: Druhý způsob nemůže být použit v nezkompilované databázi. Z tohoto důvodu umístěte tento zápis do metody, která se nebude provádět v nezkompilované databázi. Název metody musí začínat slovem "COMPILER."

Tip pro rozšíření: Zápis C_TEXT (${...}) vám umožní vytvořit proměnný počet parametrů stejného typu pod podmínkou že toto jsou poslední parametry pro metodu. Například zápis C_TEXT (${5}) řekne 4D a 4D Compileru že od pátého parametru budou všechny parametry stejného typu a je jedno kolik jich bude. Pro další informace si přečtěte příkaz Count parameters.

Příklady

Podívejte se na příklad u části " Příkazy kompilátoru".

Dále si přečtěte

Příkazy kompilátoru, Count parameters, C_STRING.

Příkazy a odkazy pro Kompilátor

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

C_TIME

Příkazy a odkazy pro Kompilátor

Verze 3

C_TIME ({metoda; }proměnná{; proměnná2;...;proměnnáN})

Parametr	Typ		Popis
metoda	Metoda	→	Volitelný název metody
proměnná	Proměnná nebo ${...}	→	Název proměnných k vytvoření

Popis

C_TIME určí proměnnou *proměnná* jako proměnnou typu čas.

První způsob použití tohoto příkazu, kde volitelná *metoda* NENÍ předána, je pro deklaraci jakékoli procesové, meziprocesové nebo místní proměnné.

Poznámka: Tento způsob může být použit v nezkompilované databázi.

Druhý způsob, kde je použit volitelný parametr *metoda*, je pro předdefinování výsledku pro 4D Compiler a nebo pro předdefinování parametrů ($0, $1, $2 atd.) pro zadanou metodu. Použijte tento způsob k přeskočení počátečního natypování proměnných při kompilaci, ušetříte čas kompilace.

UPOZORNĚNÍ: Druhý způsob nemůže být použit v nezkompilované databázi. Z tohoto důvodu umístěte tento zápis do metody, která se nebude provádět v nezkompilované databázi. Název metody musí začínat slovem "COMPILER."

Tip pro rozšíření: Zápis C_TIME (${...}) vám umožní vytvořit proměnný počet parametrů stejného typu pod podmínkou že toto jsou poslední parametry pro metodu. Například zápis C_TIME (${5}) řekne 4D a 4D Compileru že od pátého parametru budou všechny parametry stejného typu a je jedno kolik jich bude. Pro další informace si přečtěte příkaz Count parameters.

Příklady

Podívejte se na příklad u části " Příkazy kompilátoru".

Dále si přečtěte

Příkazy kompilátoru, Count parameters.

Příkazy a odkazy pro Kompilátor

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

IDLE (PRODLEVA)

Verze 3

IDLE

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz IDLE je navržen pouze pro 4D Compiler. Tento příkaz je používán pouze ve zkompilovaných databázích, ve kterých jsou uživatelské metody napsány tak, že nevolají zpět stroj 4D. Například pokud metoda obsahuje smyčku For...End for, ve které nejsou prováděny žádné příkazy 4th Dimension, nemůže být smyčka přerušena procesem instalovaným pomocí ON SERIAL PORT CALL nebo ON EVENT CALL, ani se uživatel nemůže přepnout do jiné aplikace. V tomto případě můžete použít příkaz IDLE a umožnit 4th Dimension zachytit události. Pokud nechcete žádná přerušení, vynechte IDLE.

Příklady

V následujícím příkladu nemůže být smyčka nikdy ukončena bez volání příkazu IDLE:

```
   ` Metoda projektu Udělat něco
ON EVENT CALL ("METODA UDÁLOSTI")
<>vbNášStop:=False
MESSAGE ("Probíhá..."+Char(13)+"Stiskněte libovolnou klávesu k přerušení...")
Repeat
      ` Zde nějaký kód, který nezahrnuje příkazy 4D
   IDLE
Until (<>vbNášStop)
ON EVENT CALL ("")
```

A dále:

```
   ` Metoda projektu METODA UDÁLOSTI
If (Undefined(KeyCode))
   KeyCode:=0
End if
If (KeyCode#0)
   CONFIRM ("Opravdu chcete zastavit tuto akci?")
   If (OK=1)
      <>vbNášStop:=True
   End if
End if
```

Dále si přečtěte

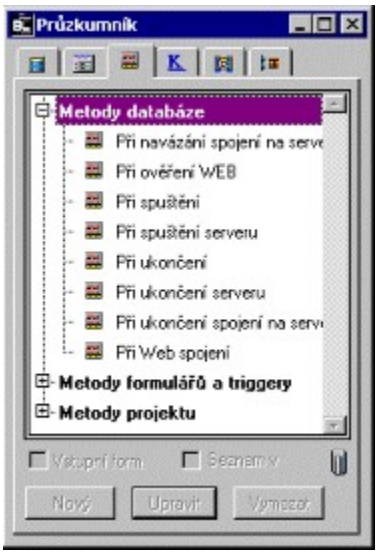
Příkazy kompilátoru, ON EVENT CALL, ON SERIAL PORT CALL.

Metody databáze

Příkazy a odkazy pro Metody databáze

Verze 6.0

Metody databáze jsou metody které jsou automaticky prováděny pokud se udějí základní události 4th Dimension.



K vytvoření nebo otevření a upravě metod databáze:
1. Otevřete okno **Průzkumníka.**
2. Vyberte stránku **Metody**.
3. Rozšiřte část **Metody databáze**.
4. Poklepejte na metodu.

Nebo:
1. Vyberte metodu.
2. Klepněte na tlačítko **Upravit**.

Metody databáze jsou upravovány stejně jako jiné metody.

Metody databáze nemůžete volat z jiné metody. Metody databáze jsou automaticky prováděny, když se 4th Dimension nachází v určitých typických okamžicích práce. Následující tabulka ukazuje kdy se provádí:

Metoda databáze	***4th Dimension***	***4D Server***	***4D Client***
Při spuštění	*Ano, jednou*	*Ne*	*Ano, Jednou*
Při ukončení	*Ano, jednou*	*Ne*	*Ano, Jednou*
Při Web spojení	*Ano, víckrát*	*Ano, víckrát*	*Ne*
Při ověření Web	*Ano, víckrát*	*Ano, víckrát*	*Ne*
Při spuštění serveru	*Ne*	*Ano, jednou*	*Ne*
Při ukončení serveru	*Ne*	*Ano, jednou*	*Ne*
Při navázání spojení na serveru	*Ne*	*Ano, víckrát*	*Ne*
Při ukončení spojení na serveru	*Ne*	*Ano, víckrát*	*Ne*

Jestli chcete vědět více informací o jednotlivých metodách databáze, přečtěte si následující části:

• Metoda databáze Při spuštění

- Metoda databáze Při ukončení
- Metoda databáze Při Web spojení
- Metoda databáze Při ověření WEB
- Metoda databáze Při spuštění serveru (Manuál 4D Server)
- Metoda databáze Při ukončení serveru (Manuál 4D Server)
- Metoda databáze Při navázání spojení na serveru (Manuál 4D Server)
- Metoda databáze Při ukončení spojení na serveru (Manuál 4D Server)

Dále si přečtěte

Metody.

Příkazy a odkazy pro Metody databáze

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Metoda databáze Při spuštění

Příkazy a odkazy pro Metody databáze

Verze 6.0

Metoda databáze Při spuštění je volána jednou při spuštění databáze.

Je prováděna v následujících prostředích 4th Dimension:

- 4th Dimension
- 4D Client (na straně klienta po navázání spojení na serveru)
- 4D Runtime
- 4D aplikace kompilované s 4D Compiler s připojeným 4D Engine.

Poznámka: Metoda databáze Při spuštění NENÍ prováděna na 4D Server.

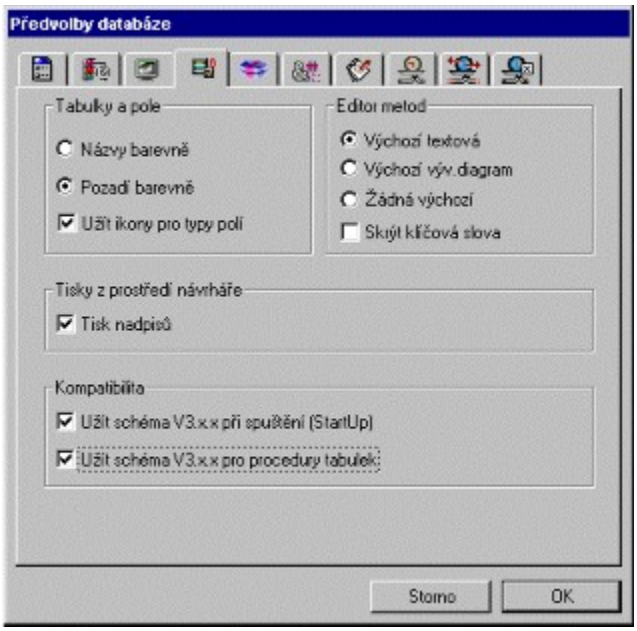
Metoda databáze Při spuštění je automaticky prováděna 4th Dimension: na rozdíl od metod projektu jí nemůžete nijak volat. K provedení akcí v metodě databáze Při spuštění používejte podprogramy.

Metoda databáze Při spuštění je vhodná pro následující akce:

- Nastavení meziprocesových proměnných používaných při práci databáze.
- Automatrické spuštění procesů, potřebných od otevření databáze.
- Načtení předvoleb a nastavení uložených pro tento účel během předchozí práce.
- Zabránění spuštění databáze, pokud nejsou splněny určité podmínky (např. chybí systémové zdroje) a v tomto případě zavolání QUIT 4D
- Provedení všech akcí, které potřebujete splnit pokaždé při otevření databáze.

Kompatibilita s předchozí verzí 4D

Metody databáze jsou nový typ metod obsažený ve verzi 6. V předchozích verzích 4th Dimension byla pouze jedna metoda (procedura) která byla automaticky prováděna při otevření databáze. Tato procedura se nazývala STARTUP (US, INTL verze) nebo DEBUT (Francouzská verze). Pokud převádíte svou databázi z verze 3 na verzi 6 a chcete využít možnosti metod databáze, musíte odznačit předvolbu **Užít staré schéma při spuštění (StartUp)** v okně **Předvolby databáze**. Tato předvolba ovlivňuje pouze alternativně metody STARTUP/Při spuštění. Ať je předchozí předvolba zaškrtnuta jakkoliv a vy napíšete kód do Metody databáze Při ukončení, bude tato metoda při ukončení databáze vždy spuštěna.



Příklad

Přečtěte si příklad u části Metoda databáze Při ukončení.

Dále si přečtěte

Metody databáze, Metody, Metoda databáze Při ukončení, QUIT 4D.

Příkazy a odkazy pro Metody databáze

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Metoda databáze Při ukončení

Příkazy a odkazy pro Metody databáze

Verze 6.0

Metoda databáze Při ukončení je volána jednou při ukončení databáze.

Tato metoda je použita v následujících prostředích 4D:

• 4th Dimension
• 4D Client (na klientovi)
• 4D Runtime
• 4D aplikace kompilované s 4D Compiler s připojeným 4D Engine.

Poznámka: Metoda databáze Při ukončení NENÍ prováděna na 4D Server.

Metoda databáze Při ukončení je automaticky prováděna 4th Dimension: na rozdíl od metod projektu jí nemůžete nijak volat. K provedení akcí v Metodě databáze Při ukončení používejte podprogramy.

Databáze bude ukončena, pokud se stane následující:

• Uživatel vybere položku Konec z nabídky Soubor v prostředí uživatele nebo návrháře.
• Je proveden příkaz QUIT 4D.
• 4D Plug-in zavolá příkaz QUIT 4D.

Nezáleží na tom jak je databáze vypnuta, 4D vždy provede následující:
• Pokud neexistuje Metoda databáze Při ukončení, 4D bez rozdílu přeruší všechny procesy jeden po druhém. Pokud uživatel zrovna provádí zadávání dat, je záznam zrušen a neuložen.
• Pokud existuje Metoda databáze Při ukončení, 4D ji spustí v novém místním procesu. Můžete tuto metodu využít k informování jiných procesů, přes meziprocesovoukomunikaci, že databáze bude skončena a tyto musí ukončit zadávání dat a ukončit své provádění. Jakmile se databáze jednou vypínána, můžete v Metodě databáze Při ukončení provést všechny operace uzavření a ukončení které potřebujete a odložit tak ukončení, ale nemůžete pomocí této metody zrušit vypnutí databáze. Jednou bude tato metoda ukončena a po ní okamžitě i databáze.

Metoda databáze Při ukončení je vhodná pro provedení následujících akcí:

• Ukončení procesů, které byli automaticky spuštěny při zapnutí databáze.
• Uložení (místně na disk) Předvolbeb a Nastavení tak, aby bylo možno je znovu použít při dalším spuštění databáze v Metodě databáze Při spuštění,.
• Provedení všech dalších akcí, které chcete splnit pokaždé, když se databáze vypíná.

Příklad

Následující příklad pokrývá metody, použité v databázi k sledování významných události, spouštejí se během práce a zapisují popis do dokumentu "Žurnál."

• Metoda databáze Při spuštění.nastavuje meziprocesovou proměnnou <>vbQuit4D, která říká všem procesům jestli se databáze vypíná. Také vytváří soubor Žurnál, pokud neexistuje:

```
   ` Metoda databáze Při spuštění
C_TEXT(<>vtIPMessage)
C_BOOLEAN(<>vbQuit4D)
<>vbQuit4D:=False
If (Test path name ("Žurnál") # Is a document)
   $vhDokumRef:=Create document("Žurnál")
```

```
    If (OK=1)
        CLOSE DOCUMENT($vhDokumRef)
    End if
End if
PSÁT ŽURNÁL ("Otevření databáze")
```

• Metoda projektu *PSÁT ŽURNÁL,* použitá jako podprogram jinými metodami, zapisuje informace které jsou jí předány do souboru Žurnál:

```
    ` Metoda projektu PSÁT ŽURNÁL
    ` PSÁT ŽURNÁL ( Text )
    ` PSÁT ŽURNÁL ( Popis události )
C_TEXT($1)
C_TIME($vhDokumRef)
While (Semaphore("$Žurnál"))
    DELAY PROCESS( Current process;1)
End while
$vhDokumRef:=Append document("Žurnál")
If (OK=1)
    PROCESS PROPERTIES( Current process;$vsProcessName;$vlState; $vlElapsedTime;$vbVisible)
    SEND PACKET($vhDokumRef;String(Current date)+Char(9)+String(Current time)
            +Char(9)+String(Current process)+Char(9) +$vsProcessName+Char(9)+$1+Char(13))
    CLOSE DOCUMENT($vhDokumRef)
End if
CLEAR SEMAPHORE("$Žurnál")
```

Všiměte si, že dokument je pokaždé znovu otevřen a uzavřen. Také si všiměte použití semaforů jako "ochrany přístupu" k dokumentu - nechceme aby se dva procesy pokoušely zapisovat do dokumentu ve stejnou chvíli.

• Metoda M_PRIDAT_ZAZNAMY je spuštěna při vybrání položky nabídky **Přidat záznam,** z nabídek v prostředí Vlastních nabídek:

```
` Metoda projektu M_ PRIDAT_ZAZNAMY
MENU BAR(1)
Repeat
    ADD RECORD([Tabulka1];*)
    If (OK=1)
        PSÁT ŽURNÁL ("Přidání záznamu #"+String(Record number([Tabulka1]))+" do Tabulky1")
    End if
Until ((OK=0) | <>vbQuit4D)
```

Tato metoda se opakuje dokud uživatel nezruší zadávání nebo dokud nevypne databázi.

• Vstupní formulář pro tabulku [Tabulka1] obsahuje kód pro událost formuláře *Při vnějším volání* (On outside call). Pokud se v procesu a formuláři zadávají data, může uživatel vybrat jestli chce záznam uložit nebo zrušit:

```
    ` Metoda formuláře [Tabulka1];"Vstup"
Case of
    ÷ (Form event=On Outside Call)
        If (<>vtIPMessage="QUIT")
```

```
            CONFIRM("Končí se !! Chcete uložit změny v tomto záznamu?")
            If (OK=1)
               ACCEPT
            Else
               CANCEL
            End if
         End if
   End case
```

• Metoda projektu M_QUIT je spuštěna při vybrání položky Konec z nabídky Soubor v prostředí Vlastní nabídky:

```
   ` Metoda projektu M_QUIT

$vlProcessID:=New process ("DO_QUIT";32*1024;"$DO_QUIT")
```

Tato metoda používá malý podvod. Pokud je příkaz QUIT 4D spuštěn, má okamžitý účinek na přerušení procesu, kde je spuštěn. Proces, ze kterého je spuštěn tento příkaz, je pozastaven dokud není databáze vypnuta. Protože tento proces může být jedním z procesů, do kterých se zadávají data, je tento příkaz prováděn z místního procesu, který je vytořen pouze z tohoto důvodu.

Zde je metoda DO_QUIT:

```
   ` Metoda projektu DO_QUIT
CONFIRM("Opravdu chcete vypnout databázi?")
If (OK=1)
   PSÁT ŽURNÁL ("Vypíná se databáze")
   QUIT 4D
      `Quit 4D má okamžitý efekt v tomto procesu a proto žádný následující řádek nebude spuštěn
      ` ...
End if
```

• Konečně je zde Metoda databáze Při ukončení, která řekne všem ostatním procesům "Je čas skončit!" Nastaví proměnnou <>vbQuit4D na True a pošle meziprocesovou zprávu k procesům uživatelů ve kterých se zadávají data:

```
   ` Metoda databáze Při ukončení
<>vbQuit4D:=True
Repeat
   $vbDone:=True
   For ($vlProcess;1;Count tasks)
      PROCESS PROPERTIES($vlProcess;$vsProcessName;$vlState; $vlElapsedTime;$vbVisible)
      If ((($vsProcessName="ML_@") |($vsProcessName="M_@"))) & ($vlState>=0))
         $vbDone:=False
         <>vtIPMessage:="QUIT"
         BRING TO FRONT($vlProcess)
         CALL PROCESS($vlProcess)
         $vhStart:=Current time
         Repeat
            DELAY PROCESS( Current process;60)
         Until ((Process state ($vlProcess)<0) | ((Current time-$vhStart)>=?00:01:00?))
      End if
   End for
Until ($vbDone)
WRITE ŽURNÁL ("Vypnutí databáze")
```

Poznámka: Procesy které začínají na "ML_..." nebo "M_..." jsou spuštěné položkou nabídky (menu), pro kterou byla zaškrtnuta předvolba **Začít nový proces**. V tomto příkladu jsou tyto procesy prováděné pomocí položky **Přidat záznam**.

Test (**Current time**-$vhStart)>=?00:01:00? v předchozí Metodě databáze Při ukončení, umožní této metodě databáze čekat na jiné procesy ve symčce Repeat, pokud jiné procesy nezafungují na volání okamžitě.

Následující ukázka je typický příklad souboru Žurnál vytvořeného databází:

2/6/97	*15:47:25*	*1*	*Proces uživatele/aplikace*	*Otevření databáze*
2/6/97	*15:55:43*	*5*	*ML_1*	*Přidání záznamu #23 do Tabulka1*
2/6/97	*15:55:46*	*5*	*ML_1*	*Přidání záznamu #24 do Tabulka1*
2/6/97	*15:55:54*	*6*	*$DO_QUIT*	*Vypínání databáze*
2/6/97	*15:55:58*	*7*	*$xx*	*Vypnutí databáze*

Poznámka: Název $xx je název místního procesu zapnutém 4D pro provedení metody databáze Při ukončení.

Dále si přečtěte

Metoda databáze Při spuštění, QUIT 4D.

Příkazy a odkazy pro Metody databáze

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ADD RECORD (PŘIDAT ZÁZNAM)

Příkazy a odkazy pro Vstup dat

Verze 3

ADD RECORD ({tabulka}{;}{*})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro vstup dat, pokud vynecháno tak Platná tabulka
*		→	Skryje posuvník

Popis

Příkaz ADD RECORD umožní uživateli zadat nový záznam do databáze a to pro tabulku *tabulka* nebo pro platnou tabulku, pokud je parametr vynechán.

ADD RECORD vytvoří nový záznam, udělá z něj platný záznam pro platný proces a zobrazí vstupní formulář. Pokud uživatel v prostředí vlastních nabídek potvrdí nový záznam, je výběr záznamů omezen pouze na tento jediný záznam.

Následující obrázek ukazuje typický vstupní formulář.

Formulář je zobrazen v tom okně procesu, které je na popředí. Okno má posuvník a políčko pro změnu velikosti. Definováním volitelného parametru * umožníte měnit velikost okna bez posuvníků, posuvníky budou skryté.

ADD RECORD zobrazuje formulář dokud uživatel nepřijme nový záznam. Pokud uživatel zadává více záznamů, musí být tento příkaz proveden pro každý záznam znovu.

Záznam je uložen když uživatel klepne na tlačítko Přijmout nebo stiskne klávesu Enter (na číselné klávesnici) nebo je proveden příkaz ACCEPT.

Záznam není uložen (je zrušen) pokud uživatel klepne na tlačítko Storno / Zrušit nebo stiskne kombinaci kláves pro zrušení (Ctrl-Tečka na Windows nebo Command-Tečka na Macintoshi) nebo je proveden příkaz CANCEL.

Po provedení příkazu ADD RECORD je proměnná OK nastavena na 1 pokud je záznam přijmut a na 0 pokud je zrušen.

Poznámka: Pokud je záznam zrušen, zůstane stále v paměti a může být uložen pomocí SAVE RECORD před tím,

než se změní ukazatel na záznam.

Příklady

1. Následující příkaz je smyčka k zadávání nových záznamů do databáze:

```
INPUT FORM ([Zákazníci];"Std Vstup") ` Nastaví vstupní formulář pro tabulku [Zákazníci]
Repeat ` Opakovat dokud uživatel nezruší
   ADD RECORD ([Zákazníci];*) ` Přidat záznam do tabulky [Zákazníci]
Until (OK=0) ` Dokud uživatel nezruší záznam
```

2. Následující příklad vyhledá v databázi konkrétního zákazníka. Podle výsledku hledání se provede jedna ze dvou akcí. Pokud nebyl nalezen žádný záznam, může uživatel přidat nový záznam pomocí ADD RECORD. Pokud byl nalezen alespoň jeden záznam, může uživatel upravit první záznam ve výběru pomocí MODIFY RECORD:

```
READ WRITE([Zákazníci])
INPUT FORM([Zákazníci];"Vstup") ` Nastaví vstupní formulář
vlCustNum:=Num(Request ("Zadejte číslo zákazníka:")) ` Dostane číslo zákazníka
If (OK=1)
   QUERY ([Zákazníci];[Zákazníci]CustNo=vlCustNum) ` Hledá zákazníka
   If (Records in selection([Zákazníci])=0) ` Pokud nebyl nalezen žádný…
      ADD RECORD([Zákazníci]) ` Přidat nového zákazníka
   Else`byl nalezen
      If(Not(Locked([Zákazníci]))) ` není uzamčen
         MODIFY RECORD([Zákazníci]) ` Upravit záznam
         UNLOAD RECORD([Zákazníci])
      Else
         ALERT("Záznam se právě používá.")
      End if
   End if
End if
```

Dále si přečtěte

ACCEPT, CANCEL, CREATE RECORD, MODIFY RECORD, SAVE RECORD.

Systémové proměnné a Sady

Uložením záznamu nastavíte systémovou proměnnou OK na 1; zrušením záznamu nastavíte proměnnou OK na0. Systémová proměnná OK je nastavena pouze pokud je záznam přijat nebo zrušen.

Příkazy a odkazy pro Vstup dat

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

MODIFY RECORD (UPRAVIT ZÁZNAM)

Příkazy a odkazy pro Vstup dat

Verze 3

MODIFY RECORD ({tabulka}{; }{*})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ve které změnit záznam, pokud vynecháno pak použít platnou tabulku
*		→	Skryje posuvníky

Popis

Příkaz MODIFY RECORD umožní uživateli upravit platný záznam pro tabulku *tabulka* a pokud je tento parametr vynechán, tak se použije platná tabulka. MODIFY RECORD načte záznam, pokud není načten nebo otevřen platným procesem a zobrazí vstupní formulář. Pokud nexistuje žádný platný záznam, příkaz neudělá nic. MODIFY RECORD nemění platný výběr.

Formulář je zobrazen v tom okně procesu, které je na popředí. Okno má posuvník a políčko pro změnu velikosti. Definováním volitelného parametru * umožníte změnit velikost okna bez posuvníků, posuvníky jsou skryty.

Pro použití příkazu MODIFY RECORD musí být tabulka ve stavu číst/psát a záznam nesmí být uzamčen.

Pokud formulář obsahuje tlačítka pro přesun v platném výběru, MODIFY RECORD umožní uživateli přesunout se na jiný záznam.

Záznam je uložen pokud uživatel klepne na tlačítko Přijmout nebo stiskne klávesu Enter (na číselné klávesnici) nebo je proveden příkaz ACCEPT.

Záznam není uložen (zrušen) pokud uživatel klepne na tlačítko Zrušit nebo stiskne kombinaci kláves pro zrušení (Ctrl-Tečka na Windows nebo Command-Tečka na Macintoshi) nebo je proveden příkaz CANCEL.

Po provedení příkazu MODIFY RECORD je proměnná OK nastavena na 1 pokud je záznam přijmut a na 0 pokud je zrušen.

Poznámka: Pokud je záznam zrušen, zůstane v paměti a může být uložen pomocí SAVE RECORD před tím, než se změní ukazatel na záznamu.

Pokud používáte příkaz MODIFY RECORD a uživatel nezmění data v záznamu, t.j není nastaven příznak, že byl záznam upraven, není záznam znovu ukládán (akce Přijmout neprovade nic). Akce jako upravení proměnných či označení zaškrtávacího políčka nejsou považovány za změny. Pouze upravení dat v poli ať už metodou nebo manuálně způsobí uložení záznamu při klepnutí na tlačítko Přijmout.

Příklad

Podívejte se na příklad u příkazu ADD RECORD.

Dále si přečtěte

ADD RECORD, Locked, Modified record, READ WRITE, UNLOAD RECORD.

Systémové proměnné a Sady

Uložením záznamu nastavíte systémovou proměnnou OK na 1; zrušením záznamu nastavíte proměnnou OK na0. Systémová proměnná OK je nastavena pouze pokud je záznam přijatý nebo zrušený.

Příkazy a odkazy pro Vstup dat

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ADD SUBRECORD (PŘIDAT PODZÁZNAM)

Verze 3

ADD SUBRECORD (podtabulka; formulář {; *})

Parametr	Typ		Popis
podtabulka	Podtabulka	→	Podtabulka použitá pro vstup dat
formulář	Řetězec	→	Formulář pro vstup dat
*		→	Skrýt posuvníky

Popis

Příkaz ADD SUBRECORD umožní uživateli zadat podzáznam do podtabulky *podtabulka* s použitím formuláře *formulář*. ADD SUBRECORD vytvoří nový podzáznam v paměti, udělá z něj platný podzáznam a zobrazí formulář. Pro tuto akci musí existovat platný podzáznam pro tabulku "rodičů." Pokud neexistuje žádný platný rodičovský záznam, příkaz neprovede nic. Formulář musí patřit k *podtabulce*.

Podzáznam je uložen pokud uživatel klepne na tlačítko Přijmout nebo stiskne klávesu Enter (na číselné klávesnici) nebo je proveden příkaz ACCEPT.

Podzáznam není uložen (zrušen) pokud uživatel klepne na tlačítko Zrušit nebo stiskne kombinaci kláves pro zrušení (Ctrl-Tečka na Windows nebo Command-Tečka na Macintoshi) nebo je proveden příkaz CANCEL.

Po provedení příkazu ADD SUBRECORD je proměnná OK nastavena na 1 pokud je podzáznam přijmut a na 0 pokud je zrušen.

Formulář je zobrazen v tom okně procesu, které je na popředí. Okno má posuvník a políčko pro změnu velikosti. Definováním volitelného parametru * umožníte změnit velikost okna bez posuvníků, posuvníky jsou skryty.

Příklad

Následující příklad je část metody, která přidá nový podzáznam pro dítě k novému zaměstnanci. Data pro děti jsou uložena v podtabulce [Zaměstnanci]Děti. Všiměte si, že záznam v tabulce [Zaměstnanci] musí být po uložení nového podzáznamu uložen:

```
ADD SUBRECORD([Zaměstnanci]Děti;"FormulářPřidat")
If (OK=1) ` Pokud uživatel uloží nový podzáznam
   SAVE RECORD ([Zaměstnanci]) ` uložit záznam zaměstnance
End if
```

Dále si přečtěte

ACCEPT, CANCEL, MODIFY SUBRECORD, SAVE RECORD.

Systémové proměnné a Sady

Uložením podzáznamu nastavíte systémovou proměnnou OK na 1; zrušením záznamu nastavíte proměnnou OK na 0.

MODIFY SUBRECORD (UPRAVIT PODZÁZNAM)

Verze 3

MODIFY SUBRECORD (podtabulka; formulář; {; *})

Parametr	Typ		Popis
podtabulka	Podtabulka	→	Podtabulka použitá pro vstup dat
formulář	Řetězec	→	Formulář pro vstup dat
*		→	Skrýt posuvníky

Popis

Příkaz MODIFY SUBRECORD zobrazí platný podzáznam podtabulky *podtabulka* pro upravení ve formuláři *formulář* pro podtabulku. Formulář musí patřit podtabulce.

Musí existovat platný podzáznam pro tabulku "rodiče." Pokud neexistuje žádný platný rodičovský záznam, příkaz nic neprovede.

Podzáznam je uložen pokud uživatel klepne na tlačítko Přijmout nebo stiskne klávesu Enter (na číselné klávesnici) nebo je proveden příkaz ACCEPT.

Podzáznam není uložen (zrušen) pokud uživatel klepne na tlačítko Zrušit nebo stiskne kombinaci kláves pro zrušení (Ctrl-Tečka na Windows nebo Command-Tečka na Macintoshi) nebo je proveden příkaz CANCEL.

Po provedení příkazu MODIFY SUBRECORD je proměnná OK nastavena na 1 pokud je záznam přijmut a nastavena na 0 pokud je zrušen.

Formulář je zobrazen v tom okně procesu, které je na popředí. Okno má posuvník a políčko pro změnu velikosti. Definováním volitelného parametru * umožníte změnit velikost okna bez posuvníků, posuvníky jsou skryty.

Dále si přečtěte

ACCEPT, ADD SUBRECORD, CANCEL, SAVE RECORD.

Systémové proměnné a Sady

Uložením podzáznamu nastavíte systémovou proměnnou OK na 1; zrušením podzáznamu nastavíte proměnnou OK na 0.

Příkazy a odkazy pro Vstup dat

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DIALOG (dialogové okno)

Příkazy a odkazy pro Vstup dat

Verze 3

DIALOG ({tabulka;} formulář)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ve které je formulář, pokud vynecháno použita platná tabulka
formulář	Formulář	→	Formulář k zobrazení jako dialog

Popis

Příkaz DIALOG zobrazí formulář *formulář* uživateli. Tento příkaz se nejčastěji používá k získání informací od uživatele z proměnných, které může zadat, nebo k informování uživatele, třeba o možnostech provádění určité akce.

Je obvyklé, že se formulář zobrazí uvnitř modálního okna, které bylo před tím vytvořeno pomocí příkazu Open window.

Zde je příklad dialogu:

V dialogu můžete zadávat data pouze pomocí proměnných. Pole mohou být zobrazena s platnými hodnotami, ale nebudou dostupná k změnám.

Tip: Pokud potřebujete provést zadávání do polí, může být dialog simulován příkazem ADD RECORD. V tomto případě pokud je formulář potvrzen, je do databáze přidán záznam.

Tip: Naopak lze i dialog použít k zadávání nových dat. V tomto případě musíte záznam vytvořit a uložit programem. Samotý příkaz DIALOG nepracuje se záznamy.

Pokud informace, které potřebujete představit uživateli, nebo odezva, již potřebujete, je složitější, než mohou zvládnout příkazy ALERT, CONFIRM nebo Request použijte zde příkaz DIALOG.

Na rozdíl od příkazů ADD RECORD nebo MODIFY RECORD, DIALOG nepoužívá platný vstupní formulář. Formulář, který chcete použít musíte definovat pomocí parametrů. Pokud nejsou ve formuláři použita žádná tlačítka není zde použit celý výchozí panel tlačítek. Jsou vytvořena pouze tlačítka OK a Zrušit. Předání vlastních tlačítek do formuláře vynechá výchozí tlačítka OK a Zrušit.

Dialog je přijmut pokud uživatel klepne na tlačítko Přijmout nebo stiskne klávesu Enter (na číselné klávesnici) nebo je proveden příkaz ACCEPT.

Dialog je zrušen pokud uživatel klepne na tlačítko Zrušit nebo stiskne kombinaci kláves pro zrušení (Ctrl-Tečka na

Windows nebo Command-Tečka na Macintoshi) nebo je proveden příkaz CANCEL.

Po provedení příkazu DIALOG je proměnná OK nastavena na 1 pokud je záznam přijmut a na 0 pokud je zrušen.

Příklad

Následující příklad ukazuje použití DIALOG k definování kritérií pro hledání. Vlastní formulář obsahující proměnnou vName a vState je zobrazen a uživatel může zadat hledání

```
Open window (10;40;370;220) ` Otevře modální okno
DIALOG([Firmy];"Search Dialog") ` Zobrazí vlastní dialog hlednání
CLOSE WINDOW ` Okno již není potřeba
If (OK=1) ` Pokud byl dialog potvrzen
   QUERY ([Firmy];[Firmy]Název=vName;*)
   QUERY ([Firmy];&;[Firmy]State=vState)
End if
```

Dále si přečtěte

ACCEPT, ADD RECORD, CANCEL, Open window.

Systémové proměnné a Sady

Potvrzením dialogu nastavíte systémovou proměnnou OK na 1; zrušením dialogu nastavíte proměnnou OK na 0.

Příkazy a odkazy pro Vstup dat

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Modified (Upraveno)

Příkazy a odkazy pro Vstup dat

Verze 3

Modified (pole) → Logické

Parametr	Typ		Popis
pole	Pole	→	Pole k testování
Výsledek funkce	Logické	←	True pokud byla k poli přiřazena nová hodnota, jinak False

Popis

Modified vrací True pokud byla do pole předána nová hodnota, buď programově, nebo během vstupu dat z klávesnice.

Během vstupu dat je pole upraveno pokud uživatel změní hodnotu pole a pak přejde na jiné pole nebo jinou ovládací kartu. Všiměte si, že pouhý přechod tabelátorem z pole na pole nenastaví Modified na True. Pole musí být skutečně upraveno aby Modified vrátilo True.

Při provádění metody je pole upraveno tehdy, pokud mu byla přiřazena hodnota (ať již odlišná nebo ne od současné).

Poznámka: Modified vždy vrátí True po provedení příkazů PUSH RECORD a POP RECORD.

V libovolném případě lze použít příkaz Old k detekování jestli byla hodnota skutečně změněna.

Poznámka: Ačkoliv Modified může být použit na všechny typy polí, pokud jej použijete v kombinaci s příkazem Old mějte na paměti omezení příkazu Old. Jestli chcete vědět více informací, přečtěte si příkaz Old.

Během vstupu dat je obvykle jednodušší provést požadované operace v metodě objektu než použít Modified v metodě formuláře a zde pak tuto operaci. Protože metoda objektu posílá událost *Při aktualizaci* při každé upravě pole, je ekvivalentní použít metodu objektu a Modified v metodě formuláře.

Příklady

1. Následující příklad testuje jestli byla změněna pole [Objednávky]Množství nebo [Objednávky]Cena. Pokud bylo jedno z těchto polí změněno, je přepočítáno pole [Objednávky]Celkem.

```
If ((Modified ([Objednávky]Množství) | (Modified ([Objednávky]Cena))
    [Objednávky]Celkem :=[Objednávky]Množství*[Objednávky]Cena
End if
```

Všiměte si, že stejná věc může být provedena s použitím druhého řádku jako podprogramu volaného metodou objektu pro pole [Objednávky]Množství nebo [Objednávky]Cena.

2. Označíte záznam v tabulce [TatoTabulka], pak zavoláte více podprogramů které mohou upravit pole [TatoTabulka]Důležité, ale neuloží záznam. Na konci hlavní metody použijete příkaz Modified k zjištění, jestli bylo pole změněno a záznam je nutno uložit:

```
    ` Zde byl záznam označen jako platný záznam
    ` Pak provedete akce s použitím podprogramů
UDĚLAT NĚCO
```

UDĚLAT NĚCO JINÉHO
NEZAPOMEŇTE UDĚLAT TOTO
` ...
` Pak můžete testovat pole k detekování jestli má být záznam uložen
If (**Modified**([TatoTabulka]Důležité))
SAVE RECORD([TatoTabulka])
End if

Dále si přečtěte

Old.

Příkazy a odkazy pro Vstup dat

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Old (Staré)

Příkazy a odkazy pro Vstup dat

Verze 3

Old (pole) → Výraz

Parametr	Typ		Popis
pole	Pole	→	Pole ke kterému vrátit starou hodnotu
Výsledek funkce	Výraz	←	Originální hodnota pole

Popis

Příkaz Old vrací hodnotu uloženou původně v *poli* před tím, než bylo programově nebo manuálně změněno.

Pokaždé když změníte platný záznam pro tabulku, 4D vytvoří a uloží v paměti kopii "náhledu" nového platného záznamu jakmile je načten do paměti. Při práci se záznamem pracujete s originálem a ne s kopií. Tato kopie je zrušena jakmile změníte platný záznam.

Old vrací hodnotu z kopie záznamu. Jinými slovy pro existující záznam vrací hodnotu pole jak je uložena na disku. Pokud je záznam nový, Old vrací výchozí prázdnou hodnotu patřičného typu. Například pokud je pole typu Alfa, vrátí Old prázdný řetězec, pokud je pole číselné, vrátí příkaz 0, atd.

Old pracuje s poli ať již byly změněny programově nebo během vstupu dat.

Old nemůže být použito na Text, Obrázek a BLOB pole. Může být použito na všechny ostatní typy polí, včetně podpolí, ale nelze jej používat na samotnou podtabulku.

Pokud chcete k poli přiřadit původní hodnotu, přiřaďtě k němu hodnotu vrácenou příkazem Old.

Dále si přečtěte

Modified.

Příkazy a odkazy pro Vstup dat

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Current date (Platné datum)

Příkazy a odkazy pro Datum a Čas

Verze 3

Current date ({*}) → Datum

Parametr	Typ		Popis
*		→	Vrací platné datum ze serveru
Výsledek funkce	Datum	←	Platné datum

Popis

Current date vrací platné datum ze systémových hodin.

4D Server: pokud při spuštění tohoto příkazu na 4D Client použijete parametr hvězdička (*), dostanete platný datum ze serveru.

Příklady

1. Následující příklad zobrazí výstražné okno obsahující platné datum:

```
ALERT ("Dnešní datum je " + String(Current date)+".")
```

2. Pokud píšete aplikaci pro mezinárodní obchod, můžete potřebovat vědět jestli verze 4D se kterou pracujete používá formátování datumů MM/DD/RRRR (US verze) nebo DD/MM/RRRR (Česká verze). Je to důležité vědět pro úpravy polí během vstupu dat:

Následující metoda projektu vám to umožní:

```
   ` Globální funkce Formát datumů
   ` Formát datumů → Řetězec
   ` Formát datumů → Výchozí formát datumu
C_STRING(31;$0;$vsDatum;$vsMDR;$vsMesic;$vsDen;$vsRok)
C_LONGINT($1;$vlPos)
C_DATE($vdDatum)

   ` Vzít hodnotu datum kde měsíc, den a rok jsou odlišné hodnoty
$vdDatum:=Current date
Repeat
   $vsMesic:=String(Month of($vdDatum))
   $vsDen:=String(Day of($vdDatum))
   $vsRok:=String(Year of($vdDatum)%100)
   If (($vsMesic=$vsDen) | ($vsMesic=$vsRok) | ($vsDen=$vsRok))
      vOK:=0
      $vdDatum:=$vdDatum+1
   Else
      vOK:=1
   End if
Until (vOK=1)
$0:=""   ` Nastavit výsledek funkce
$vsDatum:=String($vdDatum)
$vlPos:=Position("/";$vsDatum)   ` Najít první / oddělovač   v řetězci ../../..
$vsMDR:=Substring($vsDatum;1;$vlPos-1)   ` Vytáhnout první znaky z datumu
```

```
    ` Oddělit první znaky stejně jako první / oddělovač
  $vsDatum:=Substring($vsDatum;$vlPos+1)
  Case of
    ÷ ($vsMDR=$vsMesic)   ` Znaky znamenají měsíc
        $0:="MM"
    ÷ ($vsMDR=$vsDen) ` Znaky znamenají den
        $0:="DD"
    ÷ ($vsMDR=$vsRok) ` Znaky znamenají rok
        $0:="RRRR"
  End case
  $0:=$0+"/"   ` Začít vytvářet výsledek funkce
  $vlPos:=Position("/";$vsDatum)   ` Najít druhý oddělovač   v řetězci ../..
  $vsMDR:=Substring($vsDatum;1;$vlPos-1)   ` Vytáhnout další znaky z datumu
    ` Omezit řetězec na poslední znaky datumu
  $vsDatum:=Substring($vsDatum;$vlPos+1)
  Case of
    ÷ ($vsMDR=$vsMesic)   ` Znaky znamenají měsíc
        $0:=$0+"MM"
    ÷ ($vsMDR=$vsDen)   ` Znaky znamenají den
        $0:=$0+"DD"
    ÷ ($vsMDR=$vsRok)   ` Znaky znamenají rok
        $0:=$0+"RRRR"
  End case
  $0:=$0+"/"   ` Pokračovat ve vytváření funkce
  Case of
    ÷ ($vsDatum=$vsMesic)   ` Znaky znamenají měsíc
        $0:=$0+"MM"
    ÷ ($vsDatum=$vsDen)   ` Znaky znamenají den
        $0:=$0+"DD"
    ÷ ($vsDatum=$vsRok)   ` Znaky znamenají rok
        $0:=$0+"RRRR"
  End case
` V této chvíli je $0 buď MM/DD/RRRR nebo DD/MM/RRRR nebo...
```

Dále si přečtěte

Datumové operátory, Day of, Month of, Year of.

Příkazy a odkazy pro Datum a Čas

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Day of (Den z)

Příkazy a odkazy pro Datum a Čas

Verze 3

Day of (datum) → Číslo

Parametr	Typ		Popis
datum	Datum	→	Datum pro který vrátit den
Výsledek funkce	Číslo	←	Den měsíce v datumu

Popis

Příkaz Day of vrací den v měsíci ze zadaného datumu.

Poznámka: Day of vrací hodnotu mezi 1 a 31. Aby jste získali číslo pořadí dne v týdnu, použijte příkaz Day number.

Příklady

1. Následující příklad ukazuje použití Day of. Výsledky jsou přiřazeny do proměnné vResult. Komentáře popisují, co bude v proměnné:

```
vResult := Day of (!12/25/92!) ` vResult obsahuje 25
vResult := Day of (Current date) ` vResult obsahuje den platného datumu
```

2. Podívejte se na příklad u příkazu Current date.

Dále si přečtěte

Day number, Month of, Year of.

Příkazy a odkazy pro Datum a Čas

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Month of (Měsíc z)

Příkazy a odkazy pro Datum a Čas

Verze 3

Month of (datum) → Číslo

Parametr	Typ		Popis
datum	Datum	→	Datum ze kterého vzít měsíc
Výsledek funkce	Číslo	←	Měsíc získaný z datumu

Popis

Příkaz Month of vrací měsíc z datumu.

Poznámka: Month of vrací měsíc jako číslo, nikoliv jeho název (Podívejte se na Příklad 1).

4th Dimension obsahuje následující předdefinované konstanty:

Konstanta	***Typ***	***Hodnota***
January	*Long Integer*	*1*
February	*Long Integer*	*2*
March	*Long Integer*	*3*
April	*Long Integer*	*4*
May	*Long Integer*	*5*
June	*Long Integer*	*6*
July	*Long Integer*	*7*
August	*Long Integer*	*8*
September	*Long Integer*	*9*
October	*Long Integer*	*10*
November	*Long Integer*	*11*
December	*Long Integer*	*12*

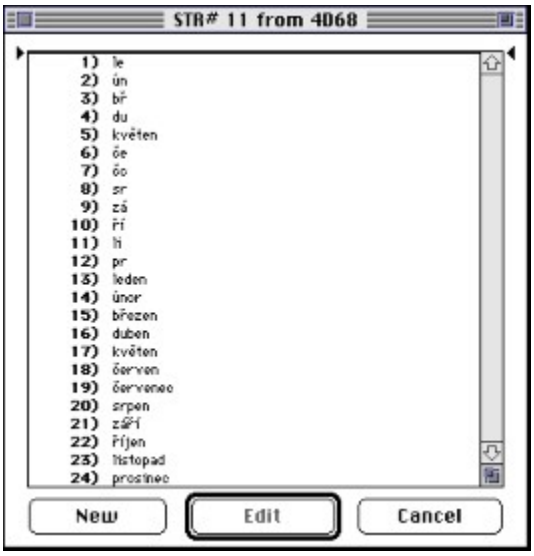
Příklady

1. Následující příklad ukazuje použití Month of. Výsledky jsou přiřazeny do proměnné vResult. Komentáře popisují, co bude v proměnné:

```
vResult := Month of (!12/25/92!) ` vResult obsahuje 12
vResult := Month of (Current date) ` vResult obsahuje měsíc z platného datumu
```

2. Podívejte se na příklad u příkazu Current date.

3. 4th Dimension zdroj "STR#" ID=11 obsahuje názvy měsíců lokalizovaných pro jednotlivé státy:



Následující metoda projektu vrací název měsíce pro datum:

```
` Metoda projektu Název měsíce
` Název měsíce (Datum) → Řetězec
` Název měsíce (Datum) → Název měsíce
$0 := Get indexed string (11;12+Month of ($1))
```

Následující metoda projektu vrací zkratku měsíce pro datum:

```
` Metoda projektu Zkratka měsíce
` Zkratka měsíce (Datum) → Řetězec
` Zkratka měsíce (Datum) → název měsíce
$0 := Get indexed string (11;Month of ($1))
```

Dále si přečtěte

Day of, Year of.

Příkazy a odkazy pro Datum a Čas

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Year of (Rok z)

Příkazy a odkazy pro Datum a Čas

Verze 3

Year of (datum) → Číslo

Parametr	Typ		Popis
datum	Datum	→	Datum pro který vrátit rok
Výsledek funkce	Číslo	←	Číslo obsahující rok z datumu

Popis

Příkaz Year of vrací rok z datumu.

Příklady

1. Následující příklad ukazuje použití **Year of** . Výsledky jsou přiřazeny do proměnné vResult:

```
vResult := Year of (!12/25/92!) ` vResult obsahuje 1992
vResult := Year of (!12/25/1992!) ` vResult obsahuje 1992
vResult := Year of (!12/25/1892!) ` vResult obsahuje 1892
vResult := Year of (!12/25/2092!) ` vResult obsahuje 2092
vResult := Year of (Current date) ` vResult obsahuje rok platného datumu
```

2. Přečtěte si příklad u příkazu Current date.

Dále si přečtěte

Day of, Month of.

Příkazy a odkazy pro Datum a Čas

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Day number (Číslo dne)

Příkazy a odkazy pro Datum a Čas

Verze 3

Day number (datum) → Číslo

Parametr	Typ		Popis
datum	Datum	→	Datum pro který vrátit číslo
Výsledek funkce	Číslo	←	Číslo označující den v týdnu

Popis

Příkaz Day number vrací číslo označující den v týdnu pro zadané datum.

Poznámka: Day number vrací 2 pro nulové datum.

4th Dimension obsahuje následující předdefinované konstanty:

Konstanta	***Typ***	***Hodnota***
Monday	*Long Integer*	*2*
Tuesday	*Long Integer*	*3*
Wednesday	*Long Integer*	*4*
Thursday	*Long Integer*	*5*
Friday	*Long Integer*	*6*
Saturday	*Long Integer*	*7*
Sunday	*Long Integer*	*1*

Poznámka: Day number vrací číslo mezi 1 a 7. k získání pořadí dne v měsíci, použijte Day of.

Příklad

Následující příklad je funkce, která vrací platný den jako řetězec:

```
$viDay := Day number (Current date) ` $viDay obsahuje číslo platného dne
Case of
   ÷ ($viDay = 1)
      $0 := "Neděle"
   ÷ ($viDay = 2)
      $0 := "Pondělí"
   ÷ ($viDay = 3)
      $0 := "Úterý"
   ÷ ($viDay = 4)
      $0 := "Středa"
   ÷ ($viDay = 5)
      $0 := "Čtvrtek"
   ÷ ($viDay = 6)
      $0 := "Pátek"
   ÷ ($viDay = 7)
      $0 := "Sobota"
End case
```

Dále si přečtěte

Day of.

Příkazy a odkazy pro Datum a Čas

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Add to date (Přidat k datumu)

Příkazy a odkazy pro Datum a Čas

Verze 6.0

Add to date (datum; roků; měsíců; dní) → Datum

Parametr	Typ		Popis
datum	Datum	→	Datum ke kterému přidat roky, měsíce a dny
roků	Číslo	→	Počet roků k přidání
měsíců	Číslo	→	Počet měsíců k přidání
dní	Číslo	→	Počet dní k přidání
Výsledek funkce	Datum	←	Výsledné datum

Popis

Příkaz Add to date přidá k datumu roky, měsíce a dny které zadáte do parametrů a vrátí datum.

Můžete použít Datumové operátory k přidání dní do datumu. Add to date vám umožní přidat měsíce a roky bez toho aby jste věděli počet dní v měsíci a přestupné roky (což musíte vědět pokud použijete datumový operátor +).

Příklady

```
    ` Tento řádek přičte k datumu jeden rok
$vdInOneYear:=Add to date(Current date;1;0;0)
    ` Tento řádek přičte k datumu jeden měsíc
$vdNextMonth:=Add to date(Current date;0;1;0)
    ` Tento řádek provede to samé jako $ vdZítrejDen:=Current date+1
$ vdZítrejDen:=Add to date(Current date;0;0;1)
```

Dále si přečtěte

Datumové operátory.

Příkazy a odkazy pro Datum a Čas

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Date (Datum)

Příkazy a odkazy pro Datum a Čas

Verze 3

Date (ŘetězecDatum) → Datum

Parametr	Typ		Popis
ŘetězecDatum	Řetězec	→	Řetězec udávající datum k navrácení
Výsledek funkce	Datum	←	Datum

Popis

Příkaz Date vezme řetězec a vrací datum.

Parametr *ŘetězecDatum* musí splňovat základní požadavky 4D na datumový formát.

V CZ verzi 4D musí být datum v pořadí DD/MM/RR (měsíc, den, rok). Den nebo měsíc mohou být jeden nebo dva znaky. Rok může být dva nebo čtyři znaky. Pokud je rok dvoumístný, pak Date k němu přidá 19 jako století. Pokud chcete používat jiné století, nastavte ho příkazem SET DEFAULT CENTURY. Následující znaky jsou platné oddělovače datumu: lomítko (/), mezera, tečka (.) a pomlčka (-).

Date nezkouší jestli je *ŘetězecDatum* správné datum. Pokud je datum neplatné (třeba "13/35/95") pak Date vrátí neplatné datum. Pokud nemůže být *Datum* přeložen jako datum (například "aa/12/95"), je vráceno nulové datum (! 00/00/00!).

Je pouze na vás, aby jste testovali správnost datumu.

Příklady

1. Následující příklad využívá žádost k uživateli k získání datumu. Řetězec zadaný uživatelem je převedený na datum a uložen do proměnné vdPožadDatum:

```
vdPožadDatum:=Date(Request ("Zadejte datum:";String(Current date)))
If (OK=1)
   ` Udělejte něco s datumem uloženým v proměnné vdPožadDatum
End if
```

2. Následující příklad vrací řetězec "12/12/94" jako datum:

```
vdDatum := Date ("12/12/94")
```

Příkazy a odkazy pro Datum a Čas

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Current time (Platný čas)

Příkazy a odkazy pro Datum a Čas

Verze 3

Current time ({*}) → Čas

Parametr	Typ		Popis
*		→	Vrací platný čas ze serveru
Výsledek funkce	Čas	←	Platný čas

Popis

Příkaz Current time vrací platný čas ze systémových hodin.

Current time je vždy mezi 00:00:00 a 23:59:59. Použijte funkce String a Time string k získání řetězce z času vráceného funkcí Current time.

4D Server: Pokud použijete parametr hvězdička (*), pak při použití této funkce na stroji 4D Client, dostanete čas na serveru.

Příklady

1. Následující příklad ukazuje jak měřit trvání operace. Je zde měřena metoda:

```
$vhStartTime:=Current time ` Uloží platný čas
LongOperation ` Provede nějakou akci
ALERT ("Tato operace trvala "+String(Current time-$vhStartTime)) ` Zobrazí jak dlouho trvala operace
```

2. Následující příklad oddělí hodiny, minuty a sekundy z platného času:

```
$vhNow:=Current time
ALERT("Hodin je: "+String($vhNow600))
ALERT("Minut je: "+String(($vhNow0)%60))
ALERT("Sekund je: "+String($vhNow%60))
```

Dále si přečtěte

Milliseconds, String, Tickcount, Časové operátory.

Příkazy a odkazy pro Datum a Čas

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Time string (Časový řetězec)

Příkazy a odkazy pro Datum a Čas

Verze 3

Time string (sekund) → Čas

Parametr	Typ		Popis
sekund	Číslo	→	Sekund od půlnoci
Výsledek funkce	Řetězec	←	Čas jako řetězec ve 24 hodinovém formátu

Popis

Příkaz Time string vrací časový řetězec z časového výrazu který přiřadíte jako parametr.

Řetězec je ve formátu HH:MM:SS.

Pokud předáte číslo větší než je počet sekund ve dni (86400) Time string bude pokračovat ve přes 24 hodin. Pokud například předáte Time string (86401) dostanete 24:00:01.

Poznámka: Pokud potřebujete pro časový výraz formátovaný řetězec v různých formátech, použijte String.

Příklad

Následující příklad zobrazí výstražné okno se zprávou "46800 sekund je 13:00:00."

```
ALERT ("46800 sekund je "+Time string (46800))
```

Dále si přečtěte

String, Time.

Příkazy a odkazy pro Datum a Čas

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Time (Čas)

Příkazy a odkazy pro Datum a Čas

Verze 3

Time (časovýŘetězec) → Čas

Parametr	Typ		Popis
časovýŘetězec	Čas	→	Čas pro který vrátit počet sekund
Výsledek funkce	Čas	←	Čas definovaný pomocí časovýŘetězec

Popis

Příkaz Time vrací časový výraz shodný s časem zadaným jako řetězec do časovýŘetězec.

Parametr časovýŘetězec musí být ve formátu HH:MM:SS a být ve 24 hodinovém formátu.

Příklad

Následující příklad zobrazí výstražné okno se zprávou "1:01 P.M. = 13 hodin 1 minuta":

```
ALERT ("1:01 P.M. = "+String(Time("13:01:00");Hour Min))
```

Dále si přečtěte

String, Time string.

Příkazy a odkazy pro Datum a Čas

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Tickcount (Počet tiků)

Příkazy a odkazy pro Datum a Čas

Verze 6.0

Tickcount → Číslo

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Číslo	←	Počet tiků (60 na sekundu), které proběhly od zapnutí počítače.

Popis

Tickcount vrací počet tiků (60 na sekundu), které proběhly od zapnutí počítače.

Poznámka: Tickount vrací hodnotu typu Long Integer.

Příklad

Podívejte se na příklad u příkazu Milliseconds.

Dále si přečtěte

Current time, Milliseconds.

Příkazy a odkazy pro Datum a Čas

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Milliseconds (Milisekund)

Příkazy a odkazy pro Datum a Čas

Verze 6.0

Milliseconds → Číslo

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Číslo	←	Počet milisekund od chvíle kdy byl počítač zapnut.

Popis

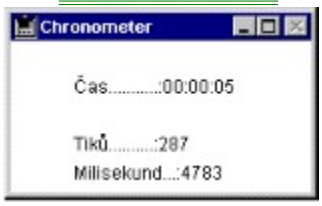
Příkaz Milliseconds vrací počet milisekund (1000 na sekundu) od chvíle kdy byl počítač zapnut.

Poznámka: Milliseconds vrací hodnotu typu Real.

Příklad

Následující příklad zobrazí okno "Chronometr" po jednu minutu:

```
Open window (100;100;300;200;0;"Chronometer")
$vhČasStart:=Current time
$vlTicksStart:=Tickcount
$vrMilliseconds Start:=Milliseconds
Repeat
   GOTO XY (2;1)
   MESSAGE ("Čas.....:"+String (Current time -$vhČasStart))
   GOTO XY (2;3)
   MESSAGE ("Tiků.....:"+String (Tickcount -$vlTicksStart))
   GOTO XY (2;5)
   MESSAGE ("Milisekund...:"+String (Milliseconds -$vrMilliseconds Start))
Until ((Current time -$vhČasStart)>=?00:01:00?)
CLOSE WINDOW
```



Dále si přečtěte

Current time, Tickcount.

Příkazy a odkazy pro Datum a Čas

4th Dimension 6.5, Seznam témat příkazů

SET DEFAULT CENTURY (NASTAVIT VÝCHOZÍ STOLETÍ)

Verze 6.0

SET DEFAULT CENTURY (století {; dělícíRok})

Parametr	Typ		Popis
století	Číslo	→	Výchozí století (mínus jedna) pro vstup datumu s dvojmístným rokem
dělícíRok	Číslo	→	Dělící rok pro vstup datumu s dvojmístným rokem

Popis

Příkaz SET DEFAULT CENTURY vám umožní pro 4D definovat výchozí století pokud do datuového pole nebo proměnné zadáte dvojmístný rok.

Jako výchozí 4D nastaví 20-té století. Například:

- 25.01.97 znamená 25. ledna 1997
- 25.01.07 znamená 25. ledna 1907

K změně tohoto století vložte jako parametr *století* výchozí století mínus jedna. Například po provedení:

```
SET DEFAULT CENTURY (20) ` Přepne na 21 století jako výchozí
```

- 25.01.97 znamená 25. ledna 2097
- 25.01.07 znamená 25. ledna 2007

Dále pokud definujete parametr *dělícíRok*, 4D přeloží dvojmístný rok následovně:

- Jestliže je rok větší než zadaný dělící rok, 4D použije platné výchozí století.
- Jestliže je rok menší než dělící rok, 4D použije další století (ve vztahu k výchozímu století).

Například po provedení tohoto řádku kde dělící rok je 1995:

```
SET DEFAULT CENTURY (19;95) ` Přepne na 21 století jako výchozí jestli je rok menší než zadaný
```

- 25.01.97 znamená 25. ledna 1997
- 25.01.07 znamená 25. ledna 2007

Poznámka: Tento příkaz ovlivňuje pouze datumy se zadaným dvojmístným rokem.

Ve všech případech:

- 25.01.1997 znamená 25. ledna 1997
- 25.01.2097 znamená 25. ledna 2097
- 25.01.1907 znamená 25. ledna 1907
- 25.01.2007 znamená 25. ledna 2007

Tento příkaz pouze ovlivňuje vstup dat. Nemá žádný vliv na výpočty datumu, ukládání atd.

SET DEFAULT CENTURY má okamžitý efekt.

Příkazy a odkazy pro Datum a Čas

4th Dimension 6.5, Seznam témat příkazů

Proč ladění?

Příkazy a odkazy pro Ladění

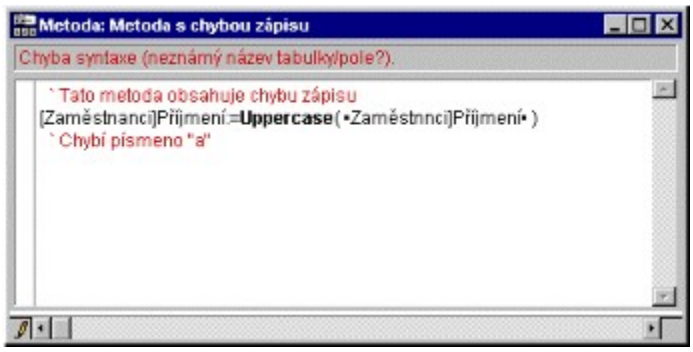
Verze 6.0

Při vytváření a testování vašich metod je důležité aby jste mohli kontrolovat a odchytávat případné chyby.

Existuje mnoho chyb, které můžete při používání jazyka udělat; chyby v zápisu, chyby syntaxe, chyby prostředí, návrhářské nebo logické chyby a chyby aplikace.

Chyby v zápisu

Chyby v zápisu jsou zachytávány **Editorem metod** a jsou označeny tečkami (?). Následující okno ukazuje chybu v zápisu:

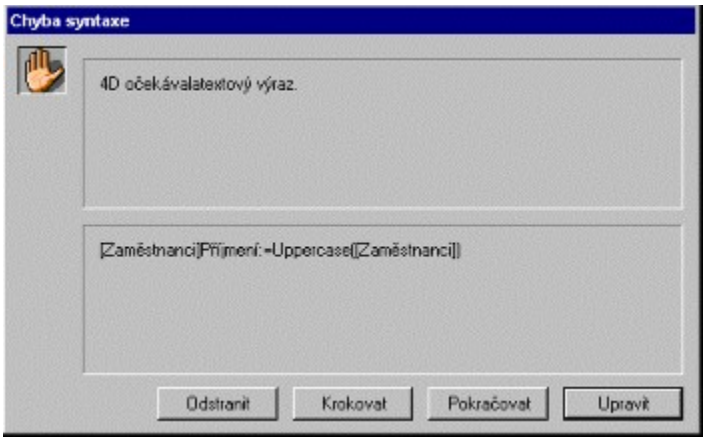


Poznámka: Komentáře byly do metody předány pro účely tohoto manuálu. 4D sama vloží pouze tečky k označení chyby.

Pokud se to stane, opravte chybu a stiskněte Enter (na číselné klávesnici) k potvrzení opravy. Jestli chcete vědět více informací o Editoru metod, přečtěte si *Příručku návrháře 4th Dimension.*

Chyby syntaxe

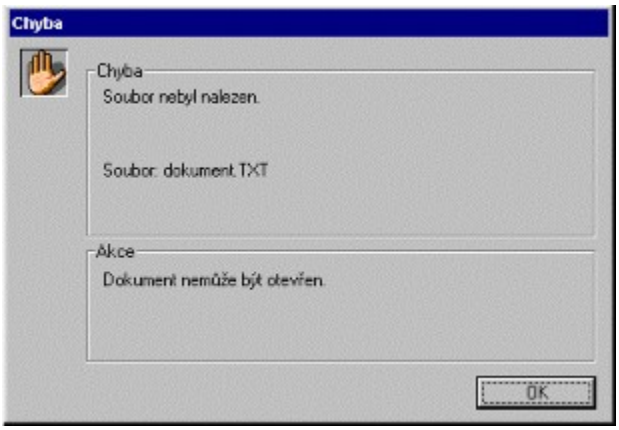
Chyby syntaxe jsou objeveny až při spuštění metody. Okno chyby syntaxe je zobrazeno pokud nastane chyba syntaxe. Například:



V tomto okně je chyba, že byl název tabulky předán do příkazu Uppercace, který vyžaduje pouze textový výraz. Aby jste se naučili o těchto chybách a o okně chyb, přečtěte si část Okno chyby syntaxe.

Chyby prostředí

Obvykle v tomto případě, není dostatek paměti k vytvoření array nebo BLOBu. Pokud otevíráte dokument z disku, tak tento nemusí existovat, nebo může být otevřen jinou aplikací. Ve většině případů se zobrazí okno s popisem chyby a s akcí která proto nemůže být provedena. Například:



Tyto chyby nejsou způsobeny psaním kódu nebo způsobem jakým kód píšete: objevují se protože "se něco stalo špatně". Většinou je tyto chyby jednoduché zachytit pomocí metody instalované příkazem ON ERR CALL. Pro více informací si přečtěte popis ON ERR CALL.

Chyby návrhu nebo Logické chyby

Tyto chyby je nejobtížnější zachytit - k jejich zachycení používejte nástroj Ladění. Všimněte si, že kromě chyb při zápisu, jsou všechny předchozí chyby z části chyby návrhu nebo logiky. Například:

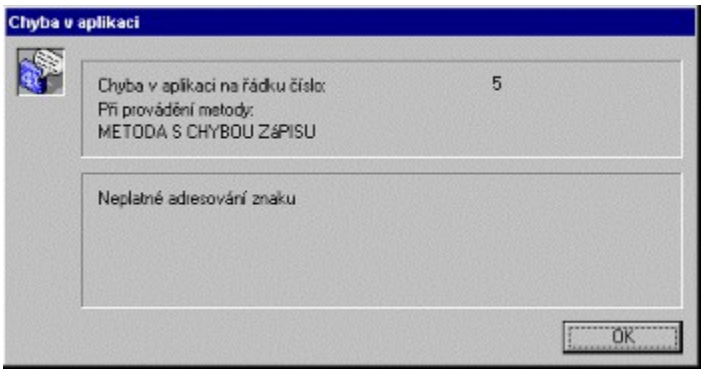
• Chyba syntaxe může být způsobena pokud se pokusíte použít proměnnou, kterou jste neinicializovali.
• Chyba prostředí může být způsobena protože se pokusíte otevřít dokument, který vytvořil podprogram a špatně vám vrátil parametr. Nezapomeňte, že tak jak v tomto příkladu ty části kódu které se přerušují nemusí být základ problému.

Chyby návrhu a logické chyby mohou také obsahovat situace jako:

• Záznam není správně obnoven, protože před voláním příkazu SAVE RECORD jste zapoměli zjistit, jestli byl záznam zamčený.
• Metoda neprovede to co má, protože není testováno jestli nejsou předány volitelné parametry.

Chyby v aplikaci

Ve zkompilovaném módu můžete objevit chyby, ketré jste v nezkompilované verzi nikdy neviděli. Zde je příklad:



Toto říká "Pokusili jste se zadat znak za pozici definovanou délkou řetězce." Aby jste rychle objevili původ problému, tak si zapametujte název metody a řádek, na kterém chyba vznikla. Znovu otevřete nezkompilovanou databázi, patřičnou metodu a chybu opravte.

Co dělat když se objeví chyba?

Chyby jsou časté. Je téměř nemožné napsat mnohořádkovou metodu bez nějakých chyb. Úplně stejně je normální hledání a opravování chyb.

Ve víceprocesovém prostředí, vám 4D umožnuje rychlé upravení a spuštění metod pouze přepnutím do jiného okna. Objevíte jak je jednoduché a rychlé opravit chyby a omyly pokud se nevracíte ke každé zvlášť. Stejně objevíte, jak je jednoduché objevit chyby pomocí okna Ladění.

Běžný začátečník při objevení okna chyby stiskne Odstranit a vrátí se do metody a pokusí se chybu nějak najít čtením kódu. To nedělejte. Je vždy jednodušší použít Ladění a chyby hledat v něm.

- Pokud se objeví chyba syntaxe, použijte Ladění.
- Pokud se obejví chyba prostředí, použijte Ladění.
- Pokud seobjeví jakákoli jiná chyba, použijte Ladění.

V 99% případů vám Ladění zobrazí dostatek informací k tomu, aby jste mohli zjistit proč metoda nefunguje. Jakmile máte tuto informaci, budete vědět jak chybu odstranit.

Tip: Několik hodin strávených při učení a experimentování s oknem Ladění vám může ušetřit dny a týdny v budoucnosti, když budete hledat chyby v kódu.

Další důvod pro použití Ladění je z důvodu vývoje kódu. Občas můžete vytvořit algoritmus, který více složitý než obvykle. Ve většině případů nezjistíte, jestli je váš kód správný, dokud jej nevyzkoušíte. Místo toho aby proběhl "na slepo" je jednodušší použít na začátku kódu příkaz TRACE. Pak metodu spustit a po jedotlivých krocích kontrolovat co provádí a zda je to správně. Puntičkáři mohou tuto metodu odsuzovat, ale zaplatí za to rychlostí a nakonec stejně použijí Ladění.

Základní doporučení

Požívejte Ladění.

Dále si přečtěte

Přerušení, Ladění, Zkratky ladění, ON ERR CALL, Okno chyby syntaxe, Krokování procesu neviditelného nebo neprovádějícího kód.

Příkazy a odkazy pro Ladění

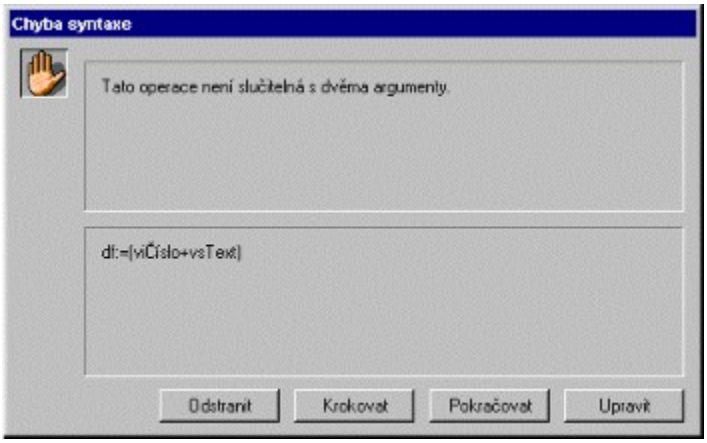
Okno chyby syntaxe

Verze 6.0

Okno chyby syntaxe se zobrazí pokud je přerušeno provádění metody. Provádění metoda může být přerušeno ze dvou důvodů:

• 4th Dimension zastaví provádění metody protože se objeví chyba syntaxe, která vylučuje další běh metody.
• Metodu jste při jejím běhu přerušili stisknutím Alt+Klepnutí (Windows) nebo Option+Klepnutí (Macintosh.

Okno chyby syntaxe je zobrazeno zde:



Vrchní textová oblast okna zobrazuje zprávu zobrazující popis chyby. Spodní část ukazuje řádek, na kterém se chyba stala; oblast kde se objevila chyba je zvýrazněná.
V tomto okně jsou čtyři tlačítka: **Odstranit**, **Krokovat**, **Pokračovat** a **Upravit**.
• **Odstranit**: Metoda je zastavena a vrátíte se do pozice před spuštěním metody. Pokud je to metoda formuláře nebo objektu reakgující na nějakou událost, vrátíte se do formuláře. Pokud je metoda prováděna z prostředí Vlastních nabídek, jste navráceni do prostředí Vlastních nabídek.

• **Krokovat**: Dostanete se do módu Krokovat/Ladění a otevře se okno Ladění. Pokud chcete spustit řádek, můžete klepnout na tlačítko Krok. Jakmile je metoda dokončena, zavře se okno Ladění.

• **Pokračovat**: Běh metody bude pokračovat. Řádek s chybou bude proveden částečně podle toho kde je chyba umístěna. Pokračujte opatrně - chyba může způsobit, že vaše metoda neprovede to co od ní očekáváte. Obvykle ale nebudete chtít pokračovat. Můžete pokračovat pokud je chyba na nedůležitém řádku jako SET WINDOW TITLE, který nemůže ohrozit funkčnost kódu. Můžete se chtít soustředit na důležité části kódu a méně důležité opravit později.

• **Upravit**: Běh metody je zastaven, 4th Dimension přepne do prostředí Návrháře. Metoda, ve které se objevila chyba, je otevřena v Editoru metod, kde vám umožní opravit chybu. Používejte tuto volbu pouze pokud jste si jisti kde chyba vznikla a jak ji opravit.

Dále si přečtěte

Ladění, ON ERR CALL, Proč Ladění?.

Příkazy a odkazy pro Ladění

4th Dimension 6.5, Seznam témat příkazů

Ladění

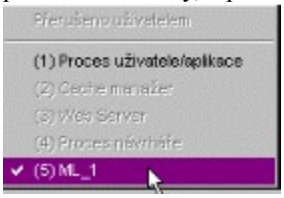
Příkazy a odkazy pro Ladění

Verze 6.0

Anglický termín pro nástroj Ladění Debugger pochází ze slova štěnice (anglicky bug). Štěnice v metodě je omyl který chcete vyhubit. Jakmile se chyba objeví, nebo potřebujete hlídat provádění kódu, otevřete okno Ladění. Ladění vám pomůže vychytat tyto "štěnice" tím, že bude pomalu procházet celou metodou a ukazovat vám, co dělají jednotlivé řádky. Tento proces prohlížení se nazývá **Krokování**.

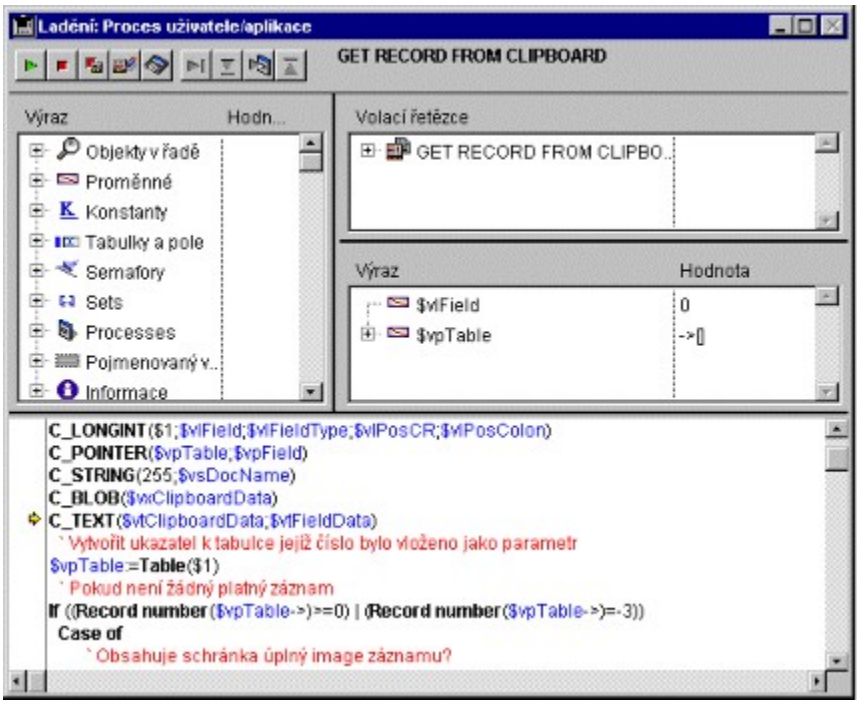
Okno Ladění můžete otevřít několika způsoby:

• Klepnutím na tlačítko **Krokovat** v okně chyby syntaxe
• Použít příkaz TRACE
• Klepnutím na tlačítko **Ladit** v okně Provést metodu.
• Stisknutím Alt+Shift+Pravé tlačítko myši (Windows) nebo Control+Option+Command+Klenutí (Macintosh) při provádění metody, a pak vybrat proces ke krokování z rozevírací nabídky.



• Klepnutím na tlačítko **Krokovat** na straně **Proces** v **Průzkumníku provádění**. (viz Krokování procesu neviditelného nebo neprovádějícího kód)
• Vytvořením a úpravou Příkazu zachycení nebo Přerušení v Editoru metod nebo na stránkách **Přerušení** a **Zachytit** v **Průzkumníku provádění**.
Poznámka: Pokud pro databázi existuje systém hesel, může pouze návrhář a uživatelé s přístupem ke struktuře krokovat metody.

Okno Ladění je zobrazeno zde:

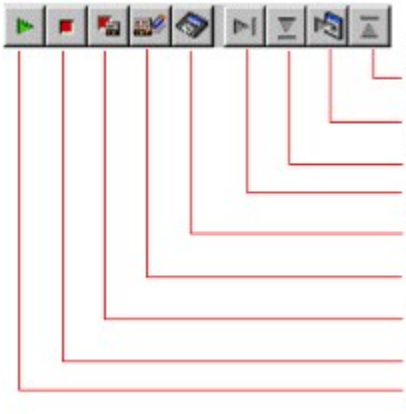


Okno Ladění můžete přesunout nebo změnit velikost, jak celého okna, tak i jeho vnitřních částí.
4D je víceprocesové prostředí. Pokud probíhá více procesů uživatele, můžete je krokovat nezávisle na sobě. Můžete

mít otevřené jedno okno Ladění pro každý proces.

Tlačítka ovládání provádění

Na vrchu okna Ladění je osm tlačítek pro kontrolu průběhu



Tlačítko		Windows
Vyskočit	F7	Ctrl-U
Krok do procesu		
Krok do	F8	Ctrl-I
Přeskočit	F10	Ctrl-S
Uložit nastavení	F3	
Upravit	F2	Ctrl-E
Odstranit a upravit		
Odstranit	F6	Ctrl-K
Nekrokovat	F5	Ctrl-R

Tlačítko Nekrokovat
Krokování je zastaveno a pokračuje normální běh metody.

Poznámka: Alt+F5 (Windows) a Option-Command-R (Macintosh) navrátí nrmální běh. Také deaktivují všechny ostatní příkazy TRACE pro platný proces.

Tlačítko Odstranit

Metoda je zastavena a jste navráceni, tam odkud jste metodu spustili. Pokud jste krokovali metodu formuláře nebo objektu, je tato zastavena a vy se navrátíte do formuláře. Pokud jste krokovali metodu spuštěnou z prostředí Vlastních nabídek, jste navráceni do prostředí Vlastních nabídek.

Tlačítko Odstranit a Upravit

Metoda je přerušena stejně jako u tlačítka Odstranit a, je-li to nutné, 4D otevře okno Návrháře a otevře se okno Editoru metod a metoda v něm se zobrazí.

Tip: Použijte toto tlačítko pokud víte jaké změny chcete udělat do kódu, aby se chyby již neopakovaly. Po té co skončíte s úpravami, znovu spusťte metodu.

Tlačítko Upravit

Klepnutí na toto tlačítko provede stejnou věc jako tlačítko **Odstranit a Upravit**, ale nepřeruší běh metody. Metoda je pouze pozastavena. Pokud to bude potřeba, 4th Dimension otevře prostředí návrháře a dále otevře okno Editoru metod a zobrazí v něm metodu, kterou chcete upravit.

DŮLEŽITÉ: Zobrazenou metodu můžete upravit, ale vaše změny neoprojeví v metodě, kterou právě provádíte a která je zobrazena v okně Ladění. Změny se projeví při dalším spuštění metody, jakmile bude současný běh této metody přerušen nebo dokončen. Jinými slovy metoda musí být aplikací znovu načtena aby se změny projevily.

Tip: Použijte toto tlačítko pokud jste si jisti jaké změny k správné funkčnosti vyžaduje metoda a pokud těmito změnami nezpůsobíte jiné problémy.

Tip: Metody objektu jsou znovu aplikací načteny pro každou událost. Pokud krokujete metodu objektu (např. metodu u tlačítka) nepotřebujete opouštět formulář. Můžete upravit metodu, překlepnout zpět do okna formuláře a metodu použít. Pokud upravujete/krokujete metodu formuláře, musíte formulář uzavřít a opět jej otevřít aby se metoda znovu načetla. Pokud často krokujete metodu formuláře, je jednodušší si část tohoto kódu vložit do metody projektu a používat jí jako podprogram pro metodu formuláře. Pokud to uděláte, můžete ve formuláři zůstat a krokovat, upravovat a znovu testovat váš formulář, protože podprogramy se budou načítat při každém novém použití

a volání metodou formuláře.

Přeskočit

Platný řádek metody (označený žlutou šipknou) je proveden a Ladění udělá krok na další řádek.Tlačítko **Přeskočit** nebude krokovat podprogramy a funkce, zůstane s laděním na úrovni metody kterou právě testujete. Pokud chcete krokovat i podprogram použijte tlačítko Krok do.

Tlačítko Krok do

Při spuštění řádku který volá jinou metodu (podprogram nebo funkci), umožní toto tlačítko otevřít okno Ladění pro tuto metodu a krokovat jí. Nová metoda se stane platnou (vrchní) metodou v Části volací řetězec,. Pokud toto tlačítko není použito na řádku s podprogramem nebo funkcí, má stejný účinek jako Přeskočit.

Krok do procesu

Na řádku, na kterém je vytvářen nový proces (příkaz New process), otevře toto tlačítko nové okno Ladění, kde můžete krokovat metodu, která vytvoří nový proces. Při použití na řádku který nespouští nový proces, má toto tlačítko stejnou funkci jako Přeskočit.

Tlačítko Přeskočit

Pokud krokujete podprogram nebo funkci, umožní vám toto tlačítko proběhnout tuto metodu bez krokování a vrátit se do volací metody. Okno Ladění je vráceno k metodě, ze které byl podprogram volán. Pokud je metoda, u které použijete toto tlačítko poslední v řetězci metod, je okno Ladění uzavřeno.

Informace o Panelu ovládání provádění

Na pravo od tlačítek kontroly provádění obsahuje okno Ladění následující informace:

- Název metody kterou zrovna krokujete (zobrazeno černě)
- Problém který je příčinou zobrazení okna Ladění (zobrazen červeně)

S použitím příkladového okna ukázaného výše, jsou zobrazeny následující informace:

- Metoda CALL RECORD FROM CLIPBOARD je metoda která se krokuje.
- Okno Ladění se objevilo protože zachytilo příkaz C_DATE, který je na seznamu zachycení.

Zde jsou možné důvody pro objevení okna Ladění a pro zprávu (zobrazenou červeně):

- **Příkaz TRACE**: Byl proveden příkaz TRACE.
- **Nalezení tečky přerušení**: Bylo objeveno dočasné nebo stálé přerušení metody
- **Přerušení uživatelem**: Pokud klepnete na Alt+Klenutí (Windows) nebo Option+Klepnutí (Macintosh) nebo vyberete položku **Krokovat** z nabídky **Proces**
- **Zachycení volání: Název příkazu**: Bylo zachyceno volání k příkazu, který má být zachycen.
- **Krokování do nového procesu**: Použijete tlačítko **Krok do procesu** a tato zpráva se objeví v okně Ladění pro nový proces.

Části okna Ladění

Okno Ladění se skládá z výše popsaného panelu pro ovládání a čtyř částí v oblastech s možnou změnou velikosti:

- Část prohlížení
- Část volací řetězec

• Část Vlastní prohlížení

• Část Zdrojový kód

První tři části jsou použity pro navigaci hierarchických seznamů zobrazující informace ladění. Čtvrtá část, Část zdrojového kódu, zobrazuje zdrojový kód metody která se krokuje. Každá z těchto částí má vlastní funkci, která vám pomůže při ladění kódu. S použitím myši můžete jednotlivé části zvětšit a posunout podle vašich potřeb. Dále první tři panely obsahují tečkovanou čáru, kterou můžete vodorovně měnit velikosti uvnitř jednotlivých částí.

Dále si přečtěte

Přerušení, Část volací řetězec, Část Vlastní prohlížení, Zkratky ladění, ON ERR CALL, Část Zdrojový kód, Okno chyby syntaxe, TRACE, Část prohlížení, Proč Ladění?.

Příkazy a odkazy pro Ladění

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

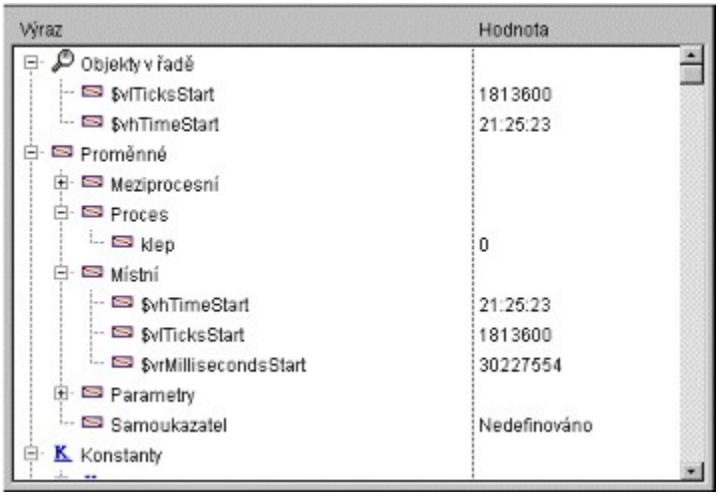
4th Dimension 6.5, Seznam témat konstant

Část Prohlížení

Příkazy a odkazy pro Ladění

Verze 6.0

Část prohlížení je zobrazena v levém horním rohu okna Ladění pod ovládacím panelem s tlačítky. Zde je příklad:



Tato část zobrazuje základní informace o systému, prostředí 4D a prostředí provádění.

Sloupec **Výraz** zobrazuje názvy objektů a výrazů. **Sloupec** Hodnota zobrazuje platnou hodnotu pro vybraný objekt nebo výraz.

Klepnutím na hodnotu ve sloupci na pravé straně části můžete upravit hodnotu objektu, pokud vám to tento objekt umožní.

Víceúrovňový hierarchický seznam je seřazen do témat. Zde jsou tato témata:

- Objekty v řadě
- Proměnné
- Tabulky a pole
- Semafory
- Sets
- Procesy
- Pojmenovaný výběr
- Informace

Podle tématu, může mít každá položka jednu či více podúrovní. Klepnutím na ukazatel u šipky můžete rozbalit tato podtémata. Jakmile je téma rozbaleno, jsou jeho položky viditelné. Pokud má toto téma více úrovní, můžete rozbalit ještě další a další dokud se nedostanete k vámi požadované informaci.

V každém případě můžete přetahovat témata, podtémata (pokud jsou nějaká) a položky do části Vlastní prohlížení,.

Informace: Zobrazuje celkové informace jako Platná tabulka (pokud je nějaká). Výrazy z této části nemohou být upraveny nebo změněny.

Pojmenovaný výběr: Zobrazuje seznam pojmenovaných výběrů pro platný proces (ten který zrovna krokujete) a také meziprocesní pojmenované výběry. Pro každý pojmenovaný výběr je zobrazen počet záznamů, které obsahuje, a tabulku ze které jsou tyto záznamy. Tento seznam může být prázdný pokud nepoužíváte pojmenované výběry. Výrazy z této části nemohou být měněny.

Procesy: Zobrazuje seznam procesů které jsou aktuálně v běhu. Sloupec Hodnota zobrazuje stav jednotlivých procesů. Výrazy z této části nemohou být měněny.

Sets (Sady): Zobrazí seznam sad které jsou používány v platném procesu (proces ve kterém krokujete) a meziprocesové sady. Pro každou sadu zobrazuje sloupec Hodnota počet záznamů a název tabulky. Tento seznam může být prázdný pokud nepoužíváte sady. Výrazy z této části nemohou být měněny.

Semafory: Zobrazí seznam místních a globálních semaforů, které se právě používají. Pro každý semafor je zobrazen název procesu, který jej používá. Tento seznam může být prázdný pokud nepoužíváte semafory. Výrazy z této části nemohou být měněny.

Pole a Tabulky: Tato část zobrazuje tabulky a pole z celé databáze; nezobrazuje podpole. Pro tabulky zobrazuje sloupec hodnota počet záznamů v platném výběru platného procesu. Pro každé pole je zobrazena hodnota pole (mimo obrázků, podtabulek a BLOBů) pro platný záznam, pokud je nějaký. V této části mohou být informace měněny (nelze zpět), nelze však měnit informace o tabulce.

Konstanty: Zobrazuje předdefinované konstanty 4D jako stránka Konstanty v Průzkumníku. Výrazy z této části nemohou být měněny.

Proměnné: Tato část obsahuje následující podtémata:

• **Meziprocesní**: Zobrazí seznam existujících meziprocesních proměnných. Tento seznam může být prázdný pokud nepoužíváte meziprocesové proměnné. Výrazy z této části mohou být měněny.

• **Proces:** Zobrazí seznam existujících procesových proměnných pro platný proces. Tento seznam je málokdy prázdný. Výrazy z této části mohou být měněny.

• **Místní**: Zobrazí seznam místních proměnných pro právě krokovanou metodu. Tento seznam může být prázdný pokud nepoužíváte místní proměnné nebo tyto ještě nebyly vytvořeny Výrazy z této části mohou být měněny.

• **Parametr**: Zobrazí seznam parametrů obdržených metodou. Tento seznam může být prázdný, pokud metoda nedostává žádné parametry. Hodnota parametru může být změněna.

• **Samoukazatel**: Zobrazí ukazatel na platnému objektu pokud krokujete metodu objektu. Tato hodnota nemůže být změněna.

Poznámka: Můžete měnit hodnotu proměnných typu Text, Řetězec, Číslo, Datum a Čas; jinými slovy můžete měnit hodnoty které je možné změnit z klávesnice.

Array, jako jiné proměnné, se objevují v částech meziprocesní, proces a místní podle jejich typu. Okno Ladění zobrazí každý array s hierarchickým seznamem; to vám umožňuje prohlédnout, nebo upravit prvky array, pokud nějaké jsou. Ladění zobrazí prvních 100 prvků včetně prvku 0. Sloupec Hodnota zobrazuje velikost array vedle jeho názvu. Pokud rozevřete seznam prvků array, zobrazí se nejdříve položka s číslem aktuálního prvku, pak prvek nula a pak ostatní prvky (do 100). Můžete upravit array typu Řetězec, Text, Číslo a Datum. Můžete upravit číslo označeného záznamu, prvek nula a jakýkoli jiný prvek (do 100). Nemůžete změnit velikost array.

Připomínka: Kdykoli můžete přetáhnout položky z části Prohlížení do části Vlastní prohlížení, a to včetně jednotlivých prvků array.

Objekty v řadě

Toto téma zobrazuje hodnotu objektů nebo výrazů které jsou:
• použité v řádku kódu který se má provést (řádek označený žlutou šipkou v části Zdrojový kód
• použité v předchozím řádku kódu
Protože předchozí řádek kódu je ten který byl právě proveden, téma Objekty v řadě ukáže objekty nebo výrazy platného řádku před a po provedení. Řekněme že provádíte tuto metodu:

```
TRACE
a:=1
b:=a+1
c:=a+b
```

1. Vstoupíte do okna Ladění na řádku a:=1. V tomto okamžiku zobrazí Objekty v řadě následující:

a: Nedefinováno

Proměnná je ukázána protože je použita v řádku, který má být proveden (ale ještě nebyla nastavena).

2. Postoupíte o jeden řádek. Ukazatel programu v části Zdrojový kód je nyní na řádku b:=a+1. V tomto okamžiku zobrazí Objekty v řadě následující:

a: 1
b: Nedefinováno

Proměnná *a* je ukázána, protože byla v řádku který byl právě proveden a byla jí přiřazena hodnota 1. Je také ukázána protože je použita v řádku, který má být proveden. Proměnná *b* je použita protože je použita v řádku který má být proveden (ale ještě nebyla nastavena).

3. Opět postoupíme o jeden řádek. Ukazatel programu je nyní na řádku c:=a+b. V tomto okamžiku budou Objekty v řadě ukazovat toto:

c: Nedefinováno
a: 1
b: 2

Proměnná *c* je ukázána protože je v řádku, který má být spuštěn (ale ještě není nastavena). Proměnná *a* a *b* jsou ukázány protože jsou použity jak v předchozím řádku tak v řádku který má být spuštěn. Atd.

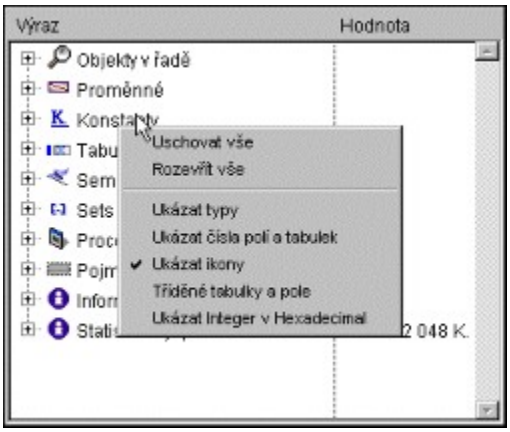
Téma Objekty v řadě je velmi používané - nemusíte totiž zadávat hodnoty, které chcete znát z provedeného řádku, do části Vlastní prohlížení, ale stačí se podívat do Objektů v řadě.

Plovoucí nabídka

Dodatečné možnosti jsou zobrazeny v rychlé nabídce části prohlížení. K zobrazení této nabídky:

- Na Windows klepněte kdekoli do části prohlížení **pravým** tlačítkem myši
- Na Macintoshi Control+klepnutí kdekoli do části prohlížení

Plovoucí nabídka je ukázána zde:



Uschovat vše: Zavře všechny úrovně hierarchického seznamu v části Prohlížení

Rozevřít vše: Rozevře první úroveň hierarchického seznamu v části Prohlížení

Ukázat typy: Zobrazí typ objektu pro všechny objekty.

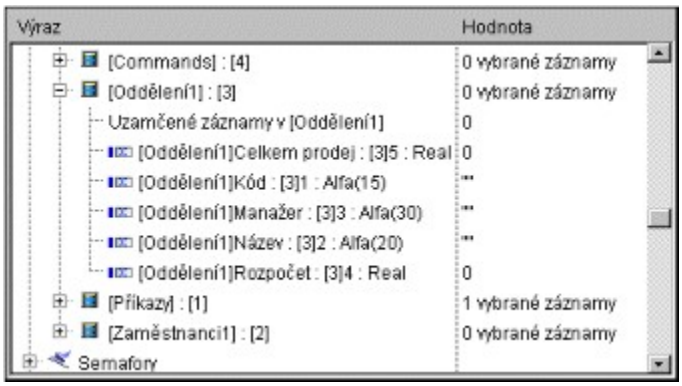
Ukázat čísla polí a tabulek: Zobrazí čísla tabulek a polí pod tématem **Tabulky a pole**. Pokud pracujete s čísly tabulek a polí nebo s ukazateli pomocí příkazů jako Table nebo Field, je tato možnost velmi užitečná.

Ukázat ikony: Zobrazí ikony definující typ objektu pro každý objekt. Můžete tuto vlastnost vypnout, protože chcete zrychlit zobrazování, nebo protože dáváte přednost pouze **Ukázání typů**.

Tříděné tabulky a pole: Setřídí seznam polí a tabulek podle abecedy.

Ukázat Integer v Hexadecimal: Čísla jsou většinou zobrazena v desítkové soustavě. Tato možnost je zobrazí v hexadecimalní soustavě. **Poznámka**: K zadání číselné hodnoty v hexadecimal, zadejte 0x (nula+x), následované hexadecimálními znaky.

Následující obrázek ukazuje zobrazení části prohlížení s označenými všemi možnostmi:



Dále si přečtěte

Část volací řetězec, Část Vlastní prohlížení, Ladění, Zkratky Ladění, Část Zdrojový kód.

Příkazy a odkazy pro Ladění

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

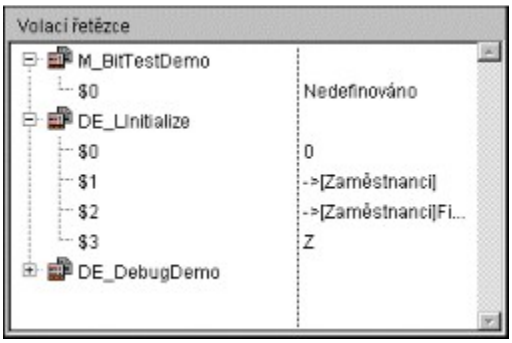
4th Dimension 6.5, Seznam témat konstant

Část Volací řetězec

Příkazy a odkazy pro Ladění

Verze 6.0

Jedna metoda může volat jiné metody, které mohou volat opět jiné metody. Z tohoto důvodu je účinnou pomocí zobrazení tohoto návazného řetězu metod či **Volacího řetězce**, během ladění procesu. Část volací řetězec, která obsahuje tuto funkci, je nahoře vpravo v okně Ladění. Tato část je zobrazena pomocí hierarchického seznamu. Zde je příklad této části:



Každá hlavní položka je název metody. Nejvyšší položka je metoda kterou aktuálně krokujete, metoda pod ní je volací metoda, další metoda je volací metoda volací metody atd. V příkladu ukázaném zde je metoda M_BitTestDemo právě krokována; byla volána metodou DE_LInitialize, která byla volána metodou DE_DebugDemo.

Poklepáním na metodu si tuto zobrazíte do části Zdrojového kódu. Pokud to uděláte, můžete se rychle podívat jak volací metoda provedla své volání k volané metodě. Tímto způsobem můžete prohlédnout kteroukoli část volacího řetězce.

Pokud metodu rozbalíte, objeví se vám parametry ($1, $2,...) předané od volací metody a volitelný parametr pro výsledek funkce metody ($0). Hodnoty se objeví napravo od názvu parametru. Klepnutím na hodnotu parametru můžete upravit hodnotu jakéhokoli parametru nebo výsledku funkce. V obrázku nahoře:

1. M_BitTestDemo nedostal žádné parametry.

2. Výsledek $0 pro M_BitTestDemo ještě nebyl definován a metoda tedy ještě nepřiřadila žádnou hodnotu jako výsledek (protože ještě nedokončila funkci a nebo protože to není funkce ale podprogram).

3. DE_LInitialize dostal tři parametry z DE_DebugDemo. $1 je ukazatel na tabulce [Zaměstnanci], $2 je ukazatel na poli [Zaměstnanci]Firma a $3 je alfanumerický parametr "Z".

Po té co si parametry prohlédnete můžete je kdykoli přetáhnout do části Vlastní prohlížení.

Dále si přečtěte

Část Vlastní prohlížení, Ladění, Zkratky ladění, Část Zdrojový kód, Část prohlížení.

Příkazy a odkazy pro Ladění

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Část Vlastní prohlížení

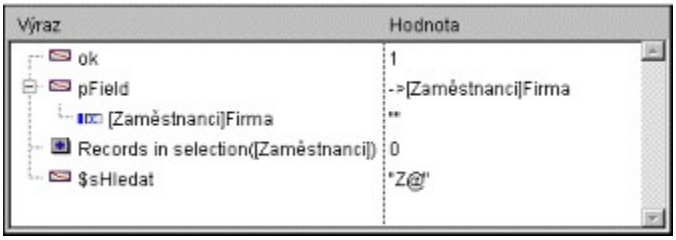
Příkazy a odkazy pro Ladění

Verze 6.0

Hned pod částí volací řetězec, je část Vlastní prohlížení. Tato část se používá k určení hodnoty výrazu. Mnoho z výrazů může být takto zjištěno včetně polí, proměnných, ukazatelů, výpočtů, vestavěných funkcí, vašich vlastních funkcí a čehokoliv jiného co vrací hodnotu.

Můžete si nechat zobrazit výrazy které mohou být ukázány v textovém formátu. Proto nelze zobrazit obrázky a BLOBy. K zobrazení hodnoty array a ukazatů Ladění používá hierarchické seznamy. Pro zobrazení hodnoty BLOBu použijte jeden z příkazů pro BLOB jako třeba BLOB to text.

V následujícím příkladu můžete vidět mnoho z těchto položek: dvě proměnné, ukazatel na poli, výsledek vestavěné funkce 4D a výpočet:



Předání nového výrazu

Do této části můžete vložit nový výraz následujícími způsoby:

- Přetáhnout objekty nebo výrazy z části prohlížení
- Přetáhnout objekty nebo výrazy z části volací řetězec,
- V části Zdrojový kód klepnout na výraz, který chcete zobrazit

K vytvoření prázdného výrazu poklepejte kdekoli ve volném místě této části. Vytvoříte tím nový výraz komentovaný `nový výraz kam můžete zapsat svůj výraz. Můžete vložit jakýkoli výraz 4D, který vrací výsledek.

Po té, co jste vložili výraz, klepněte na něj a označete jej tak, a pak ještě jednou (nebo stiskněte klávesu Enter), aby jste ho mohli upravit.

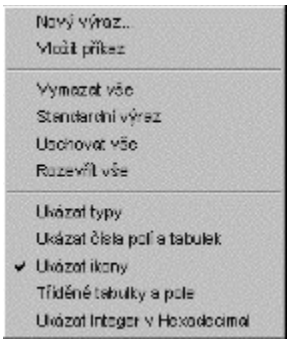
Pokud již tento výraz nepotřebujete, stačí na něj klepnout a stisknout **Backspace** nebo **Delete**.

Plovoucí nabídka pro část Vlastní prohlížení

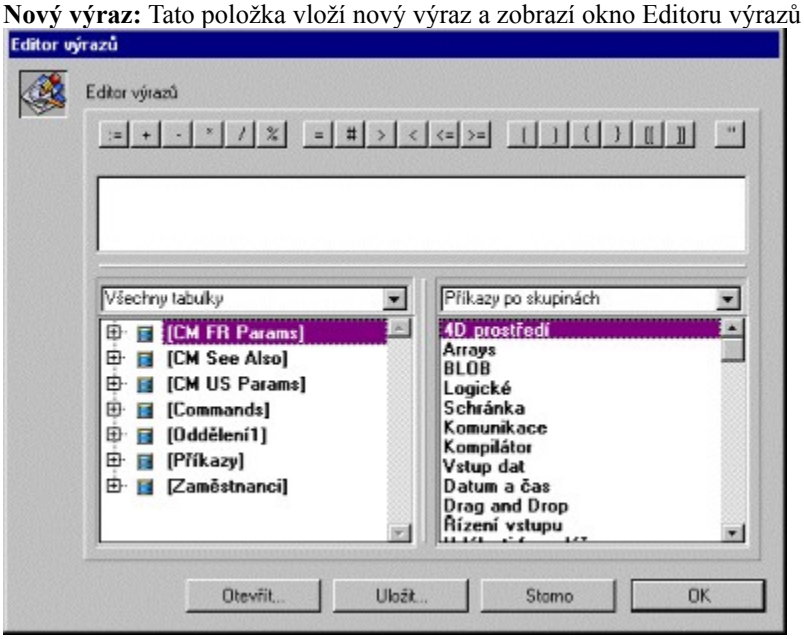
Aby jsme vám ulehčili zadávání nových výrazů, obsahuje tato část plovoucí nabídku, která Vám zpřístupní Editor výrazů. Dále ještě nabízí tato nabídka několik dalších možností.

K zobrazení této nabídky:

- Na Windows klepněte kdekoli do části Vlastní prohlížení **pravým** tlačítkem myši
- Na Macintoshi Control+klepnutí kdekoli do části Vlastní prohlížení



Nový výraz: Tato položka vloží nový výraz a zobrazí okno Editoru výrazů (ukázáno níže) k upravě výrazu.



Jestli chcete vědět více informací o Editoru výrazů, přečtěte si *Příručku uživatele 4th Dimension.*
Vložit příkaz: Tato hierarchická nabídka umožňuje vložit příkaz bez použití editoru výrazů
Vymazat vše: Vymaže všechny výrazy v části.
Uschovat vše/Rozevřít vše: Rozevře nebo uzavře všechny výrazy které jsou hierarchické (array, ukazatele, atd.)

Ukázat typy: Zobrazí typy objektů pro všechny objekty (pokud je to možné)

Ukázat čísla polí a tabulek: Zobrazí čísla tabulek a polí pod tématem **Tabulky a pole**. Pokud pracujete s čísly tabulek a polí nebo s ukazateli pomocí příkazů jako Table nebo Field, je tato možnost velmi užitečná.

Ukázat ikony: Zobrazí ikony definující typ objektu pro každý objekt. Můžete tuto vlastnost vypnout, protože chcete zrychlit zobrazování, nebo protože dáváte přednost pouze **Ukázání typů**.

Tříděné tabulky a pole: Setřídí seznam polí a tabulek podle abecedy.

Ukázat Integer v Hexadecimal: Čísla jsou většinou zobrazena v desítkové soustavě. Tato možnost je zobrazí v hexadecimalní soustavě. **Poznámka**: K zadání číselné hodnoty v hexadecimal, zadejte 0x (nula+x), následované hexadecimálními znaky.

Dále si přečtěte

Část volací řetězec, Ladění, Zkratky ladění, Část Zdrojový kód, Část prohlížení.

Příkazy a odkazy pro Ladění

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Část Zdrojový kód

Příkazy a odkazy pro Ladění

Verze 6.0

Tato část ukazuje zdrojový kód metody kterou krokujete.

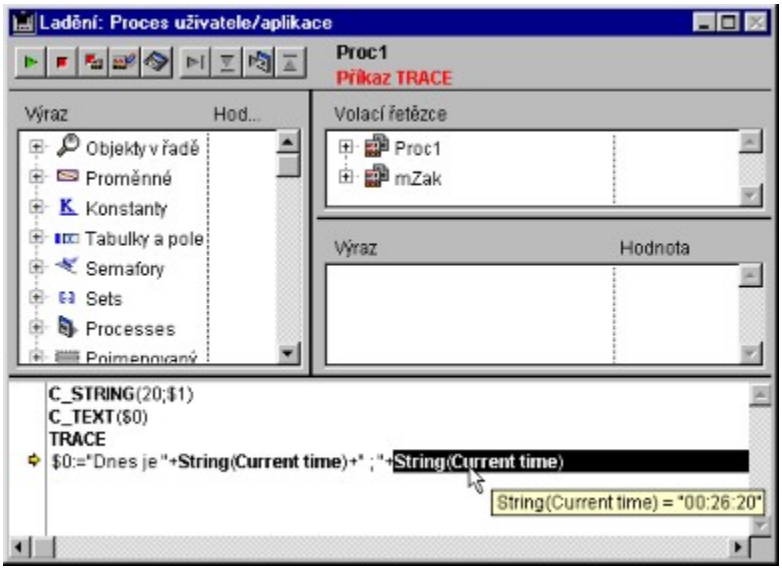
• Pokud je metoda příliš dlouhá než aby se vešla do přidělené oblasti, můžete použít posuvník kódem k zobrazení dalších částí.

• Posunutí myši nad výraz, který může být interpretován (pole, proměnná, ukazatel, array,...) zobrazí **Nástroj tip,** který vám ukáže aktuální hodnotu nebo stav vybraného výrazu a jeho typ.

Zde je příklad části Zdrojový kód:

Nástroj tip je zobrazen, protože myš byla nad ukazatelem na tabulku [Zaměstnanci].

• Můžete také označit část textu zobrazenou v oblasti prováděného kódu. Pokud v tomto případě přesunete kurzor nad označený text, zobrazí se vám tip, zobrazující hodnotu označeného objektu.



Když klepnete na proměnnou nebo pole, je tato automaticky označena.
Tip: Pokud v části Zdrojový kód klepnete na výraz, který může být vyjádřen nebo zastoupen hodnotou, přesunete se do části Vlastní prohlížení,.

Počitadlo programu

Žlutá šipka na levé straně části Zdrojový kód (podívejte se na příklad nahoře) označuje, který řádek bude spuštěn nyní. Tato šipka se nazývá Počitadlo programu. Vždy označuje řádek, který bude nyní spuštěn.

Z důvodů ladění můžete změnit umístění počitadla programu pro metodu na vrcholu řetězu metod (metodu kterou právě krokujete). Aby jste to udělali, uchopte šipku a přetáhněte ji na místo kam chcete.

UPOZORNĚNÍ: Tuto možnost používáte s nebezpečím!

Posunutí počitadla dolu NEZNAMENÁ, že se se přeskočené řádky se rychle provedou. A ani NEZNAMENÁ že posunutí počitadla nahoru zruší provedení řádků.

Posunutí počitadla pouze řekne Ladění "pokračuj v krokování nebo ladění zde". Žádné z nastavení, polí či proměnných nejsou nijak ovlivněny tímto posunem a jejich hodnoty zůstávají stejné.

Zde je příklad pro posunutí programového počitadla. Řekněme že máme následující kód:

```
` ...
If (tato podmínka)
   UDĚLAT NĚCO
Else
   UDĚLAT NĚCO JINÉHO
End if
   ` ...
```

Počitadlo programu je nastaveno na řádek *If (tato podmínka)*. Uděláte jeden krok dopředu a zjistíte, že se počitadlo přesunulo na *UDĚLAT NĚCO JINÉHO*. Toto nechcete, protože potřebujete aby se spustila první možnost. V tomto případě, a za podmínky že *Tato podmínka* neprovádí nějakou akci, která by mohla ovlivnit další kód, pouze přesunete počitadlo na řádek *UDĚLAT NĚCO*. Nyní můžete pokračovat v provádění krokování té části, která vás zajímá.

Nastavení Přerušení

Během procesu ladění můžete potřebovat přeskočit ladění určité části kódu. Ladění vám umožňuje několik způsobů jak **spustit kód od určitého bodu**:

• Během ladění můžete použít tlačítko **Přeskočit** místo **Krok do** při přecházení přes podprogramy nebo funkce volané z vaší databáze.

• Pokud již do této metody vstoupíte, můžete ji rychle provést a vrátit se do volací metody pomocí tlačítka **Vyskočit.**

• Pokud máte umístěn příkaz TRACE, můžete klepnout na **Nekrokovat** a Ladění vás znovu zastaví na dalším příkazu TRACE.

Nyní řekněme, že máme tento kód a Počítadlo programu je nastaveno na řádek ALL RECORDS ([Příkazy]):

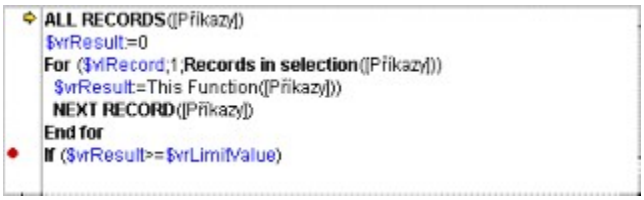
```
   ` ...
ALL RECORDS([ Příkazy])
$vrResult:=0
For($vlZaznam;1;Records in selection([ Příkazy]))
   $vrResult:=This function([Příkazy]))
   NEXT RECORD([ Příkazy])
End for
If ($vrResult>=$vrLimitValue)
```

Váš požadavek je vyhodnotit hodnotu proměnné $vrResult po provedení celé smyčky, ale to obnáší mnoho smyček a příliš času pro krokování. Nechcete tuto akci zrušit, ale nechcete postupovat po krocích, zároveň již nemůžete vložit příkaz TRACE za konec smyčky....

Jeden ze způsobů je projít přes smyčku po krocích, ačkoli tabulka [Příkazy] obsahuje stovky záznamů, Tato operace pak spotřebuje ale celý den. V tomto typu situací vám nabízí Ladění předání tečky přerušení. Můžete tuto tečku vložit na místo, kde chcete kód přerušit a odtud pokračovat v krokování.

Příklad:

Klepnete na levý okraj části Zdrojový kód u řádku If ($vrResult...:



Vložíte tak přerušení do řádku. Přerušení je ukázáno jako červená tečka. Potom můžete klepnout na tlačítko **Nekrokovat**.
Běh pokračuje normálně (okno Ladění se uzavře) až do místa kde je předáno přerušení. Tento řádek nebude spuštěn, ale provádění se se u něj zastaví a opět se zobrazí okno Ladění. V tomto příkladu proběhla smyčka normální rychlostí a provádění se zastavilo na řádku s tečkou. K odstranění tečky stačí najet myší nad červenou tečku a stisknout tlačítko myši.

Nastavení bodu přerušení za počítadlo programu vám umožní přeskočit ladění **části** kódu, pokud tyto nechcete krokovat.

Červený bod přerušení je **stálé** přerušení. Jakmile jej jednou vytvoříte, tak zůstane tam kde je. I když databázi vypnete a posléze zapnete, přerušení tam bude stále.

Jsou dva způsoby jak odstranit trvalé přerušení:

• Pokud přes něj procházíte, stačí klepnout na červenou tečku a přerušení zmizí.
• Pokud přes něj nepřecházíte, můžete jej chtít zachovat. Pokud chcete dočasně znemožnit přerušení jeho úpravou, přečtěte si část Přerušení.

Dále si přečtěte

Přerušení, Část volací řetězec, Část Vlastní prohlížení, Ladění, Část Prohlížení.

Příkazy a odkazy pro Ladění

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Přerušení

Příkazy a odkazy pro Ladění

Verze 6.0

Jak bylo popsáno v předchozí části, zadáte Přerušení klepnutím na levý okraj části Zdrojový kód na stejné úrovni jako je řádek, u kterého se chcete zastavit.

Poznámka: Protože můžete vkládat, upravovat nebo odstraňovat přerušení buď v části Zdrojový kód v Ladění nebo přímo v Editoru metod, existuje dynamická interakce ohledně přerušení mezi Editorem metod a Laděním (rovněž Průzkumníkem provádění). Dočasná přerušení však mohou být definována pouze v okně Ladění (viz níže)-

V následujícím obrázku je přerušení nastaveno na řádek **If** ($vrResult>=$vrLimitValue):

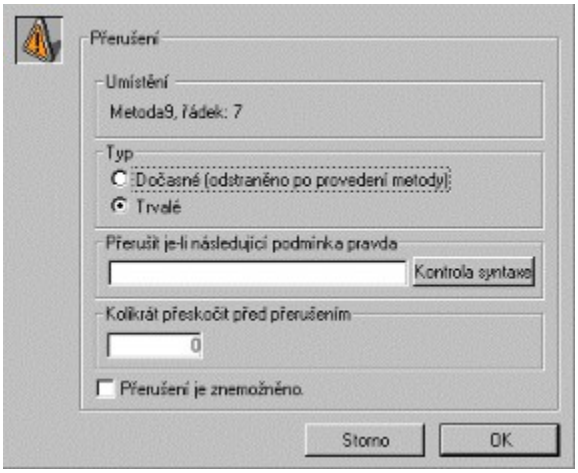
Pokud na červenou tečku klepnete ještě jednou, tak zmizí.

Úprava přerušení

Stisknutím Alt+Klepnutí (Windows) nebo Option+Klepnutí (Macintosh) na levou stránku okraje části Zdrojový kód, na požadovaný řádek, se vám zobrazí okno **Vlastnosti přerušení**.

• Pokud klepnete na existující přerušení, je zobrazeno okno předvoleb pro toto přerušení.
• Pokud klepnete na řádek, kde žádné přerušení není, Ladění jej vytvoří a zobrazí okno předvoleb pro nové přerušení.

Okno **Vlastnosti přerušení** vypadá následovně:



Jsou zde následující vlastnosti:
Umístění: Řekne vám název metody a číslo řádku na kterém přerušení je. Tuto informaci nemůžete změnit.
Typ: Jako výchozí je nastaveno **trvalé** přerušení označené červenou tečkou v okně Ladění. K vytvoření dočasného přerušení, zvolte předvolbu **Dočasné**. Tato možnost je nastavena pokud chcete toto přerušení použít pouze jednou pro toto krokování metody. Dočasné přerušení je nastaveno zelenou tečkou v okně Ladění v části Zdrojový kód.

Poznámka: Dočasné přerušení můžete nastavit také z okna ladění, pokud klpenete na levý okraj části Zdrojový kód

se stisknutými klávesami Alt-Shift (Windows) nebo Option-Shift (Macintosh).

Přerušit je-li následující podmínka pravda: Můžete vytvořit vlastní přerušení zadáním výrazu 4D, který vrací True nebo False. Například pokud chcete přerušit na tomto řádku pouze pokud bude Records in selection([aTabulka])=0, zadejte tento výraz a přerušení se provede pouze pokud bude tato podmínka splněna. Pokud si nejste jisti správností zápisu ve vašem výrazu, stiskněte tlačítko **Kontrola syntaxe.**

Kolikrát přeskočit před přerušením: Přerušení můžete nastavit do části kódu, který je smyčkou (While, Repeat nebo For), nebo jej umístit do podprogramu nebo funkce volané ze smyčky. Například víte že problém, který se vyskytne se neobjeví prvních 200 proběhnutí smyčky. Proto předáte 200 a přerušení bude 200 krát ignorováno a po 201 se teprve aktivuje.

Přerušení je znemožněno: Pokud zrovna nyní nepotřebujete přerušení, ale možná jej budete potřebovat později, můžete dočasně deaktivovat přerušení jeho upravením. Deaktivované přerušení se objeví jako pomlčka (-) a ne jako tečka (?).

Přerušení vytváříte a upravujete pouze z okna Ladění. Existující přerušení můžete také upravit ze seznamu přerušení v prostředí návrháře.

Dále si přečtěte

Přerušení, Ladění, Část Zdrojový kód.

Příkazy a odkazy pro Ladění

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Seznam přerušení

Příkazy a odkazy pro Ladění

Verze 6.0

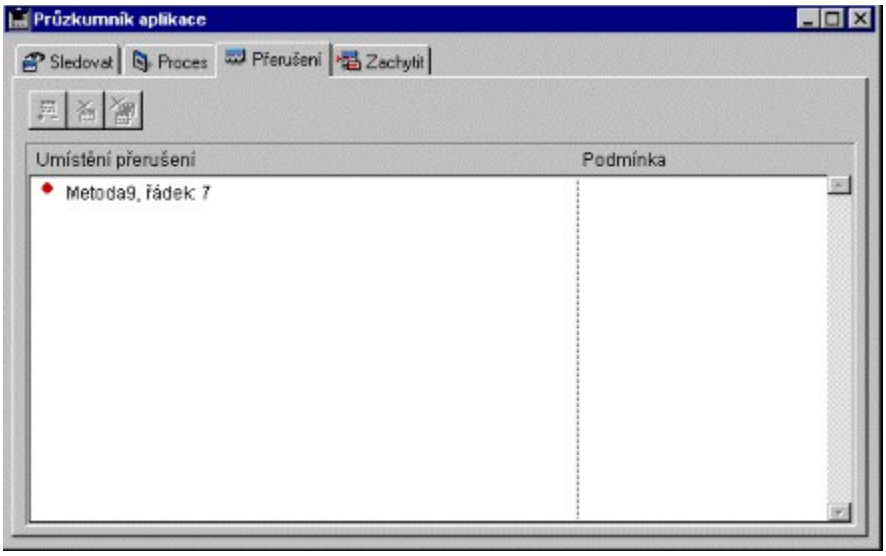
Seznam přerušení je stránka v Průzkumníku provádění, která vám umožní kontrolovat přerušení zadaná v Ladění nebo v Editoru metod.

K otevření Seznamu přerušení:

1. Pokud tam nejste, přepněte se do prostředí návrháře.
2. Z nabídky **Nástroje** vyberte položku **Průzkumník provádění**.

Průzkumník provádění může být otevřen v plovoucím okně. V tomto případě bude toto okno stále zobrazeno na popředí. K tomu podržte klávesu **Shift** při otevírání **Průzkumníka provádění** z nabídky **Nástroj** nebo použijte zkratku **Ctrl+Shift+F9** (Windows) nebo **Command+Shift+F9** (Macintosh).

Objeví se okno Průzkumníka provádění.



Seznam přerušení je složen ze dvou sloupců:

• Levý sloupec zobrazuje stav přerušení Dostupné/Nedostupné, následované názvem metody a číslem řádku na kterém je zadáno přerušení (s použitím Ladění nebo Editoru metod).
• Pravýsloupec zobrazuje podmínku, která je kpřerušení předána, pokud nějaká existuje.

S pomocí tohoto okna můžete:
• Nastavit podmínku pro přerušení,
• Zapnout, vypnout nebo vymazat každé přerušení.
• Poklepáním na přerušení otevřít Editor metod s metodou, která obsahuje přerušení.

Z tohoto okna však nemůžete přidávat nová trvalá přerušení. Trvalá přerušení mohou být přidána pouze z Ladění nebo z Editoru metod.

Nastavení podmínky pro přerušení

K nastavení podmínky pro přerušení, postupujte následovně:

1. Klepněte v pravém sloupci pro zpřístupnění vstupu.
2. Vložte výraz 4D (Výraz, příkaz, nebo metodu projektu), který vrací logickou hodnotu.

Poznámka: K odstranění podmínky vymažte její výraz.

Dostupné/Nedostupné přerušení

K zapnutí nebo vypnutí přerušení:

1. Klepnutím označte přerušení nebo použijte šipky pro posun v seznamu, dokud není označeno požadované přerušení.
2. Klepněte na tlačítko **Dostupné/Nedostupné** nebo vyberte položku **Nedostupné** z plovoucí nabídky.

Zkratka: Každé přerušení v seznamu může být přepnuto dostupné/nedostupné klepnutím na tečku. Tečka se změní na pomlčku, pokud je přerušení nedostupné.

Vymazání přerušení

K vymazání přerušení:

1. Klepnutím označte přerušení nebo použijte šipky pro posun v seznamu, dokud není označeno požadované přerušení
2. Klepněte na tlačítko **Vymazat** nebo stiskněte klávesu **Vymazat**.

Poznámka: K vymazání všech přerušení klepněte na tlačítko **Vymazat vše** nebo vyberte položku **Vymazat vše** z plovoucí nabídky.

Tipy

• Přidání podmínky k přerušení zpomalí provádění kódu, protože podmínka musí být testována pokaždé když je potkáno přerušení. Na druhou stránku přidání podmínky zrychlí ladění, protože budou automaticky přeskočena přerušení, která nesplňují podmínku.

• Nedostupné přerušení má stejný efekt jako vymazání přerušení. Během provádění bude ladění ignorovat toto přerušení. Výhoda znedostupnění přerušení je to, že jej nemusíte znovu vytvářet pokud jej budete opět potřebovat.

Dále si přečtěte

Přerušení, Zachytávání příkazů, Ladění, Část Zdrojový kód, Proč ladění?.

Příkazy a odkazy pro Ladění

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Zachytávání příkazů

Příkazy a odkazy pro Ladění

Verze 6.5

Seznam zachtávání příkazů je stránka v Průzkumníku provádění, která vám umožní přidat přerušení předáním instrukce pro zachycení příkazu 4D.

Zachytávání příkazů vám umožní začít krokování jakéhokoli procesu ihned jak proces zavolá zachytávaný příkaz. Na rozdíl od přerušení, které je umístěno pouze v jedné metodě, je rozsah zachytávání příkazu rozšířen na všechny procesy, které 4D kód obsahuje a které volají zachytávaný příkaz.

Zachytávání příkazu je způsob, jak krokovat části kódu bez nastavování přerušení na mnoha různých místech. Jestliže například objevíte, že byl vymazán záznam záznam, který by neměl být vymazán poté co jste spustili jeden nebo více procesů, můžete zkusit omezit pole své krokování pouze na zachytávání příkazů jako DELETE RECORD a DELETE SELECTION. Pokaždé, když bude tento příkaz volán jej zachytíte, otestujete jestli byl záznam v této části kódu vymazán a tak můžete objevit chybnou část kódu.

S určitými zkušenostmi můžete kombinovat použití přerušení se zachytáváním příkazů.

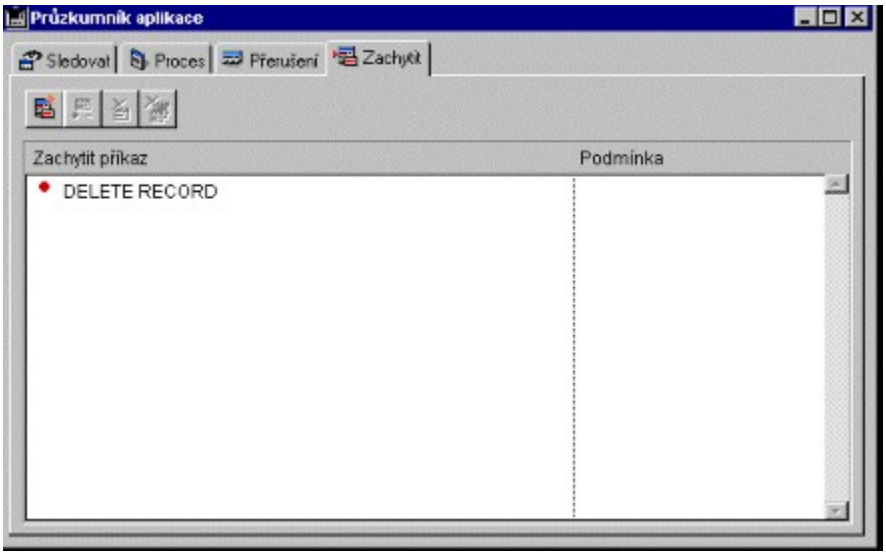
K otevření stránky Zachytávání příkazů:

1. Pokud tam nejste, přepněte se do prostředí návrháře.
2. Z nabídky **Nástroje** vyberte položku **Průzkumník provádění**.

Průzkumník provádění může být i otevřen v plovoucím okně. V tomto případě bude toto okno stále zobrazeno na popředí. K tomu podržte klávesu **Shift** při otevírání **Průzkumníka provádění** z nabídky **Nástroj** nebo použijte zkratku **Ctrl+Shift+F9** (Windows) nebo **Command+Shift+F9** (Macintosh).

Objeví se okno Průzkumníka provádění.

3. Přejděte na stránku Zachytit k zobrazení seznamu zachytávaných příkazů:



Tento seznam je složen ze dvou sloupců:

• Levý sloupec zobrazuje stav zachytávání Dostupné/Nedostupné, následované názvem příkazu k zachycení.
• Pravý sloupec zobrazuje podmínku, která je k zachycení přiřazena, pokud nějaká existuje.

Přidání nového příkazu k zachytávání

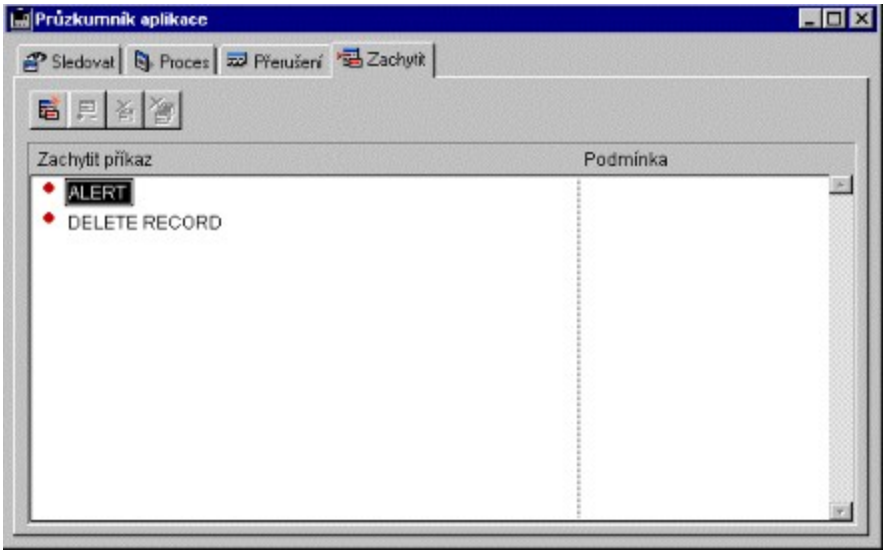
K přidání nového příkazu:

1. Klepněte na tlačítko **Přidat zachytávání** (první nad seznamem).

NEBO

Poklepejte levým tlačítkem myši v Seznamu zachytávání.

V obou případech se objeví zadávací oblast se zápisem příkazu ALERT jako výchozím. Oblast je připravena pro vstup:



2. Zadejte název příkazu který chcete zachytávat.

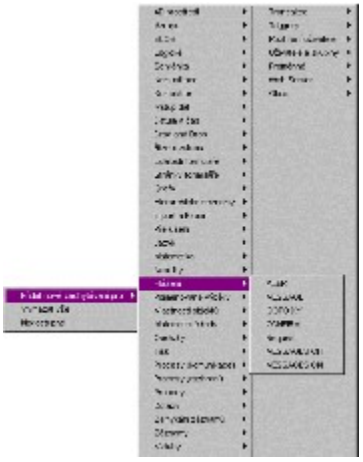
3. Stiskněte Enter nebo Return k potvrzení výběru.

NEBO

1. Stiskněte pravé tlačíko na myši (Control+Klepnutí na Macintoshi) k zobrazení plovoucí nabídky

2. Vyberte **Přidat nové zachytávání pro** a vyberte příkaz který chcete zachytit.

Do zadávací oblasti je předán nový vstup s novým příkazem.



Upravení názvu zachytávaného příkazu

K upravení názvu zachytávaného příkazu:

1. Označte položku kterou chcete změnit nebo se na ni přesuňte pomocí šipek (pokud již požadovaná položka není označena).

2. K zpřístupnění položky pro úpravy stiskněte Return nebo Enter.

3. Vložte nový nebo upravte stávající název příkazu.

4. K potvrzení změny stiskněte Return nebo Enter. Pokud zadaný text není příkaz 4D, vrátí se text položky na původní hodnotu. Pokud je tato položka nová, bude její obsah ALERT.

Umožnit/Znemožnit zachytávání příkazu

K umožnění nebo znemožnění zachytávání příkazu:

1. Označte položku kterou chcete změnit nebo se na ni přesuňte pomocí šipek (pokud již požadovaná položka není označena).

2. Pokud je položka v dostupném módu klepněte na Return nebo Enter k přepnutí na mód pro výběr.

3. Klepněte na tlačítko **Umožnit/Znemožnit** nebo vyberte z plovoucí nabídky Nedostupné.

Zkratka: Každá položka může být umožněna/znemožněna klepnutím na tečku před názvem příkazu. Tečka se změní na pomlčku (-) pokud je zachycení znemožněno.

Vymazání zachytávání příkazu

K vymazání zachytávání příkazu:

1. Označte položku kterou chcete změnit nebo se na ni přesuňte pomocí šipek (pokud již požadovaná položka není označena).

2. Pokud je položka v dostupném módu klepněte na Return nebo Enter k přepnutí na mód výběru.

3. Stiskněte klávesu **Delete** nebo klepněte na tlačítko **Vymazat**.

Poznámka: K vymazání všech příkazů klepněte na tlačítko **Vymazat vše**, nebo vyberte **Vymazat vše** z plovoucí nabídky.

Nastavení podmínky pro zachytávání příkazu

K nastavení podmínky pro zachytávání příkazu:

1. Klepněte do oblasti v pravém sloupci.

2. Vložte výraz 4D (výraz, příkaz 4D nebo metodu projektu), který vrací logickou hodnotu.

Poznámka: K odstranění podmínky, vymažte předaný výraz.

Tipy

• Vkládání podmínek pro zachytávání příkazů zpomaluje průběh kódu, protože podmínka musí být testována pokaždé když 4D narazí na příkaz. Na druhou stránku vkládání podmínek urychluje ladění, protože se nebude zastavovat na každém příkazu, ale pouze na těch, které splňují vaší podmínku.
• Znemožnění zachycení příkazu má téměř tentýž efekt jako jeho vymazání. Během provádění kódu se ladění nebude těmito přerušeními zdržovat. Výhoda této možnosti je v tom, že když budete potřebovat znovu zachytit stejný příkaz, nemusíte jej již vytvářet.

Dále si přečtěte

Přerušení, Ladění, Seznam přerušení.

Příkazy a odkazy pro Ladění

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Zkratky ladění

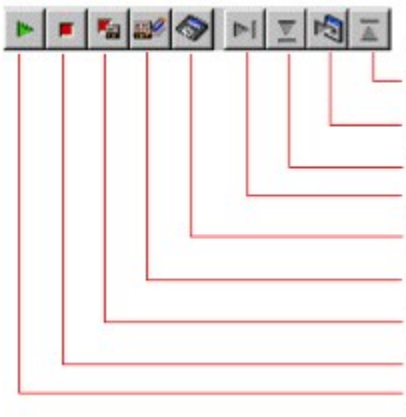
Příkazy a odkazy pro Ladění

Verze 6.0

Tato část ukazuje všechny zkratky pro okno Ladění.

Panel ovládání provádění

Následující obrázek ukazuje zkratky pro osm tlačítek umístěných v levém horním rohu okna Ladění:



Tlačítko		Windows
Vyskočit	F7	Ctrl-U
Krok do procesu		
Krok do	F8	Ctrl-I
Přeskočit	F10	Ctrl-S
Uložit nastavení	F3	
Upravit	F2	Ctrl-E
Odstranit a upravit		
Odstranit	F6	Ctrl-K
Nekrokovat	F5	Ctrl-R

Na Macintosh jsou použity tytéž zkratky, jako na Windows pouze s modifikační klávesou jablíčko.

Alt-F5 (Windows) nebo Command-Option-R (Macintosh) obnoví provádění. Také deaktivuje všechny příkazy TRACE v platném procesu.

Část Prohlížení

• Pravé tlačítko myši (Windows) nebo Control+Klepnutí (Macintosh) vám zobrazí plovoucí nabídku.
• Poklepání na položku v části Prohlížení vám tuto položku přesune části Vlastní prohlížení.

Část Volací řetězec

• Poklepání na název metody v části Volací řetězec zobrazí metodu v části Zdrojový kód na řádku odpovídajícím volání ve volacím řetězci.

Část Vlastního prohlížení

• Pravé tlačítko myši (Windows) nebo Control+Klepnutí (Macintosh) vám zobrazí plovoucí nabídku.
• Poklepání kdekoli v části Vlastní prohlížení vytvoří nové prohlížení.

Část Zdrojového kódu

• Klepnutí na levý okraj části Zdrojový kód vytvoří nebo vymaže (stálé) přerušení.
• Alt-Shit-Klepnutí (Windows) nebo Option-Shift-Klepnutí (Macintosh) nastaví dočasné přerušení.
• Alt-Klepnutí (Windows) nebo Option-Klepnutí (Macintosh) zobrazí okno **Vlastnosti přerušení.**
• Klepnutí na vybraný výraz se stisknutou klávesou **Ctrl** (Windows) nebo **Command** (MacOS) jej přenese do části Vlastní prohlížení.
• Kombinace kláves **Ctrl+D** (Windows) nebo **Command+D** (MacOS) na vybraný výraz jej přenese do části Vlastní prohlížení.

Všechny části

• Pokud není označena žádná položka v žádné části, stisknutí Enter provede další krok.
• Pokud je označena položka, použijte šipky pro posouvání v seznamu.
• Při úpravě položky, použijte šipky pro přesun kurzoru: použijte Ctrl-A/X/C/V (Windows) nebo Command-A/X/C/V (Macintosh) jako zkratky pro akce Označit vše/Vyjmout/Kopírovat/Vložit z nabídky **Upravit**.

Dále si přečtěte

Část volací řetězec, Část Vlastní prohlížení, Ladění, Část Zdrojový kód, Část Prohlížení.

Příkazy a odkazy pro Ladění

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Krokování procesu neviditelného nebo neprovádějícího kód

Příkazy a odkazy pro Ladění

Verze 6.0

Tlačítko Krok do procesu v okně Ladění vám umožní krokovat proces v té chvíli,kdy jej vytvoříte příkazem New process.

Můžete ale také chtít krokovat proces dlouho po té, co jste jej nastartovali.

Pokud má proces alespoň jedno viditelné okno a jestliže je na popředí, stisknutím Alt-Klepnutí (Windows) nebo Option-Klepnutí (Macintosh) do tohoto okna začnete mód krokování pro tento proces.

S použitím Alt-Klepnutí nebo Option-Klepnutí může být nedostatečné pokud proces:

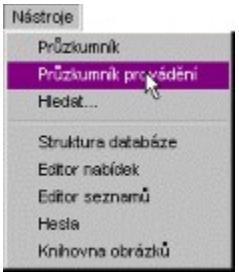
- provádí kód, ale jeho okna jsou za ostatními okny a vy je nechcete přesunout na popředí k zpřístupnění procesu.
- provádí kód, ale neobsahuje rozhraní uživatele (žádné okno)
- je ve vstupu dat a čeká na událost (*)
- je přerušen (*)
- je odložen (*)

(*) znamená že běží, ale neprovádí kód.

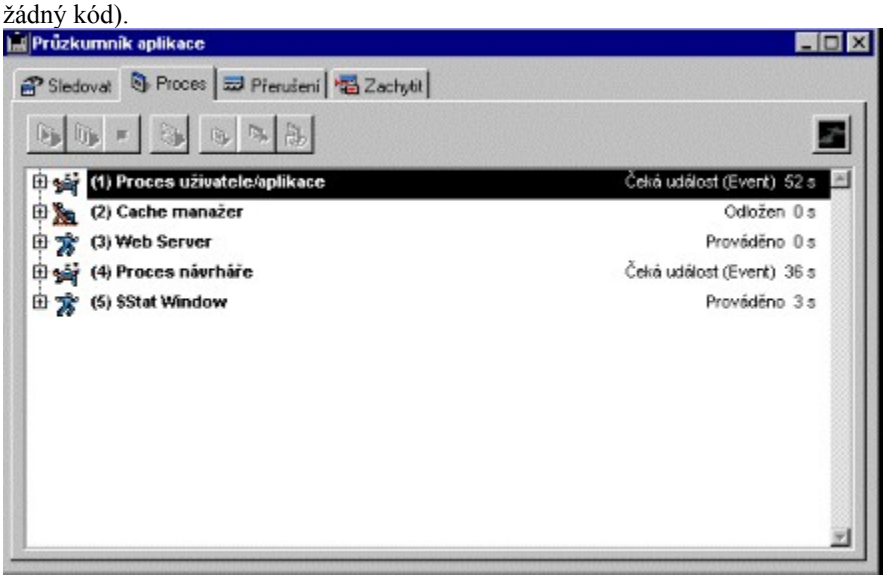
4D obsahuje ještě jednu techniku jak začít proces krokovat, aniž by byl proces "viditelný" nebo prováděl kód.

1. Pokud v něm nejste, přepněte se do prostředí návrháře.

2. Vyberte položku **Průzkumník provádění** z nabídky **Nástroje**.

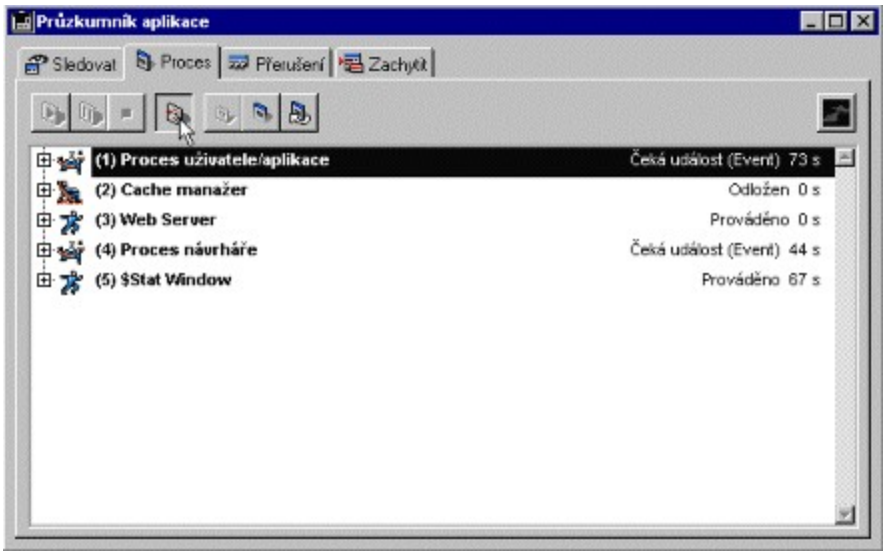


3. Přejděte na stránku Proces. Tato stránka ukazuje všechny procesy, které nyní probíhají (nemusí mít spuštěný žádný kód).



4. Označte proces který chcete krokovat.

5. Klepněte na tlačítko **Krokovat** v paletě nátrojů.



Důležité je, že 4D si teď pamatuje žádost na krokování:
• Pokud proces zrovna začne provádět nějaký kód, objeví se ihned okno Ladění.
• Pokud proces neprovádí žádný kód (např. čeká na událost), objeví se okno Ladění při prvním pokusu o provedení nějakého kódu.

Tip: Můžete chtít krokovat metodu objektu pro tlačítko, na které jste právě klepli. Alt-Klepnutí (Windows) nebo Option-Klepnutí (Macintosh) zde nemusí být dostatečné, protože záleží na "pohotovosti" při klenutí, Jednodušší je vybrat položku **Krokovat** v nabídce proces a pak když klepnete na tlačítko tak se vám ihned objeví okno ladění. Druhý způsob je vložit na začátek metody objektu příkaz TRACE.

Dále si přečtěte

Ladění, TRACE, Okno chyby syntaxe, Proč Ladění?.

Příkazy a odkazy pro Ladění

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Táhnout a vsadit

Příkazy a odkazy pro Táhnout a vsadit

Verze 6.0

Ve 4D v6 jsou obsaženy vestavěné funkce pro přetahování objektů ve formuláři. Můžete přetáhnout a vsadit jeden objekt na druhý v jednom okně nebo v různých oknech. Jinými slovy může být přetažení a položení provedeno v jednom procesu nebo mezi procesy.

4D verze 6 neobsahuje možnosti přetažení z plochy nebo z jiných aplikací. Nicméně je tato možnost dostupná pomocí zásuvných modulů (plug-in) vyvinutých ACI Partnery.

Poznámka: Na začátek předpokládejme, že přetažení je potřeba jen pro přesunutí dat z jednoho místa na druhé. Dále pak uvidíte, že přetažení může sloužit i jako způsob spouštění jakékoliv operace.

Táhnutelné a Vsaditelné vlastnosti objektů

K přetažení a vsazení jednoho objektu do druhého musíte označit vlastnost **Táhnutelné** v okně Vlastnosti objektu. V operaci táhnout a vsadit je tažený objekt **zdrojovým objektem**.

Aby byl objekt cílovým objektem pro tažení a vsazení, musíte mu označit vlastnost **Vsaditelné** v okně Vlastnosti objektu. V operaci táhnout a vsadit je objekt, který data přijme ,**cílovým objektem**.

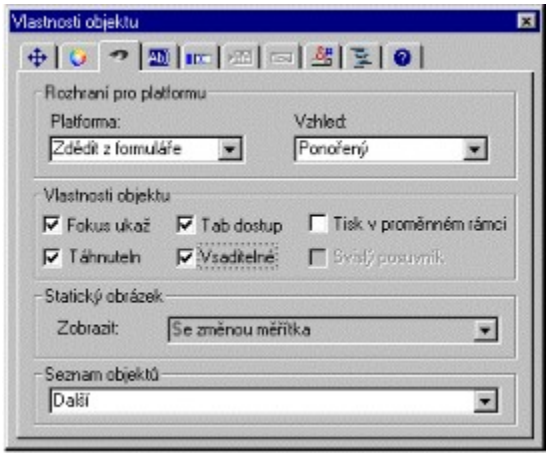
Jako výchozí nejsou objekty ani táhnutelné ani vsaditelné. Je pouze na vás, aby jste tyto možnosti púři vytváření objektu nastavili.

Všechny objekty ve vstupním formuláři nebo v dialogu mohou být táhnutelné a vsaditelné. Jednotlivé prvky array (posuvného seznamu) nebo položky hierarchického seznamu mohou být přetaženy a vsazeny. A naopak můžete táhnout a vsadit objekt na jednotlivé prvky array nebo položky hierarchického seznamu. Nemůžete však táhnout a vsadit objekty z oblasti obsahu výstupního formuláře.

Vytvořit uživatelské rozhraní pro tažení a vsazení je velmi jednoduché, protože 4D přijímá pro tyto operace všechny typy aktivních objektů (pole nebo proměnné) jako zdrojový nebo cílový objekt. Například můžete přetáhnout a vsadit tlačítko.

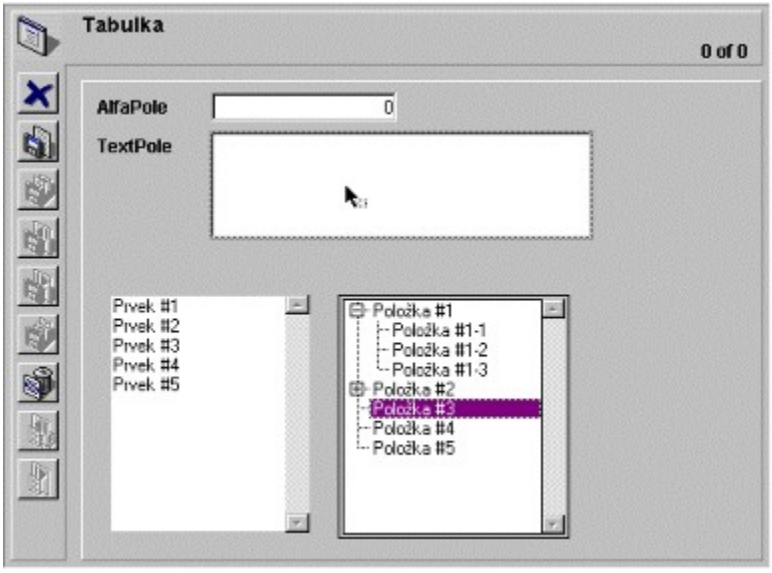
Poznámka: Objekt který má vlastnosti jak táhnutelné tak vsaditelné, může být vsazen také sám do sebe, pokud tuto operaci nevynecháte. Jestli chcete vědět více informací, přečtěte si následující část.

Následující obrázek ukazuje okno Vlastnosti objektu s možnostmi Táhnutelné a vsaditelné.



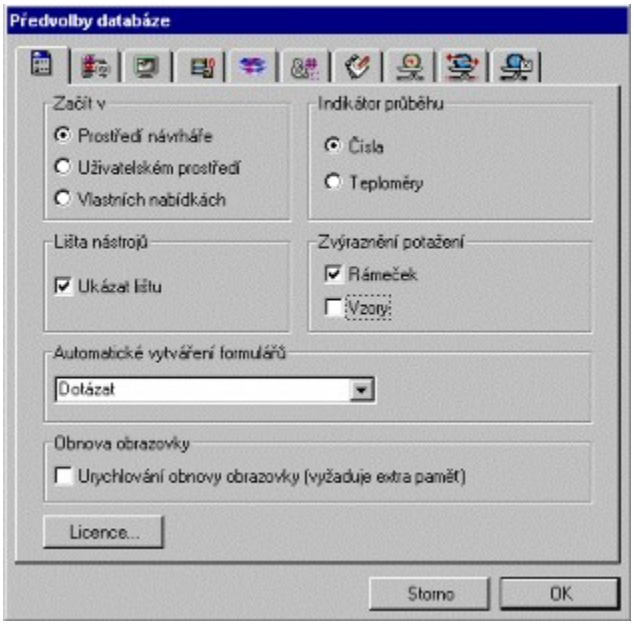
Ovládání uživatelského rozhraní Táhnout a Vsadit

4th Dimension obsahuje jako součást uživatelského rozhraní možnost táhnout a vsadit. Pokud klepnete na táhnutelný objekt a táhnete myší, 4D bude táhnout objekt: na obrazovce se tato operace objeví jako tečkovaný obdélník, který bude následovat tažení myši. V následujícím obrázku je tažena položka hierarchického seznamu do textového pole.



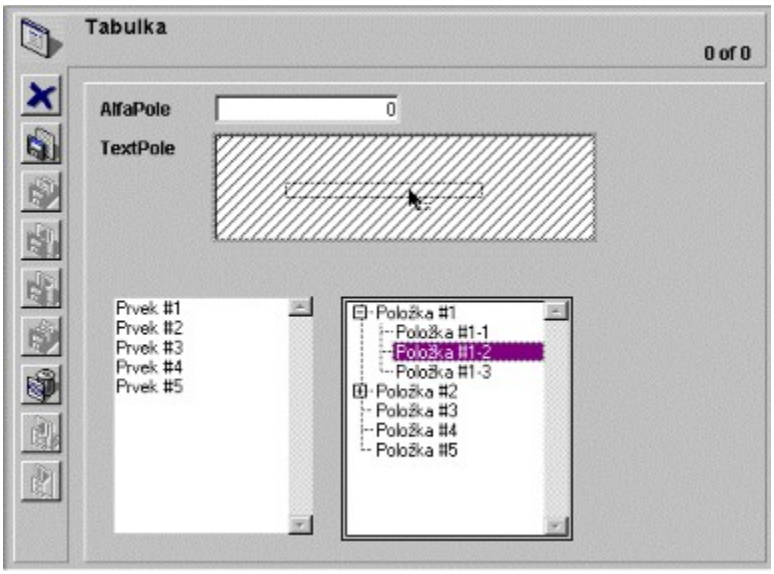
Všiměte si že kolem textového pole je šedý rámeček. Zvýraznění označuje cílový objekt (v tomo případě textové pole). Pokud v tomto okamžiku pustíte tlačítko myši, 4D přiřadí k vsaditelnému objektu hodnotu kterou táhnete ze zdrojového objektu.

V dialogovém okně **Vlastnosti objektu** můžete nastavit zvýraznění tažení a vsazení jako rámeček nebo vzor (nebo obojí):

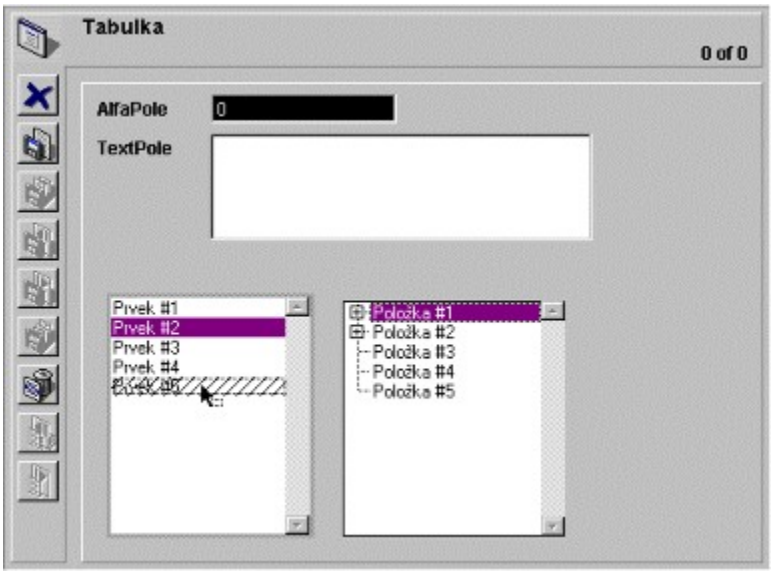


Výchozí zvýraznění je **Rámeček**. Je to obdélníkové, šedé zvýraznění kolem vsaditelného objektu. Pokud používáte barevné pozadí nebo rámeček objektu, může být zvýraznění matoucí. V těchto případech je výhodnější použít zvýraznění **Vzor**, který vyplní cílový objekt šikmými čárami, jak je ukázáno.

Zde je tažena položka hierarchického seznamu na textové pole:



Zde je tažen prvek array na vlastní array:



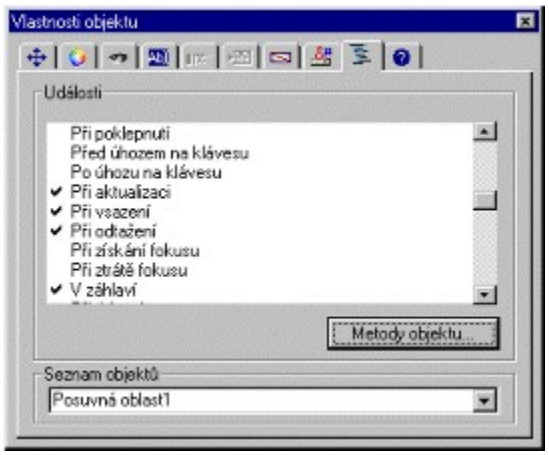
Můžete také vybrat oba typy zvýraznění.

Poznámka: Zvýraznění cílového objektu "následuje" prvek nebo položku hierarchického seznamupokud je cílový objekt array nebo hierarchický seznam.

Programové ovládání Tažení a Předání

4th Dimension umožňuje také programové ovládání tažení a vsazení. Aby to bylo možné, máte k dispozici dvě události: Při vsazení a Při odtažení. Obě události jsou poslány do **cílového objektu**. Během tažení a vsazení není nikdy spuštěna metoda zdrojového objektu.

Pro přijetí událostí Při vsazení a Při odtažení, musí mít cílový objekt tyto události aktivované, jak je ukázáno na následujícím obrázku:



Při odtažení (On Drag over)

Událost Při odtažení je opakovaně posílána cílovému objektu jakmile je kurzor myši přesunut přes tento objekt. Zároveň s touto událostí obvykle:

• Voláte příkaz DRAG AND DROP PROPERTIES který vám dá informace o zdrojovém objektu.

• Podle podstaty a typu cílového (jehož metoda byla spuštěna) a zdrojového objektu **přijmete** nebo **odmítnete** tažení a vsazení.

K přijetí tažení musí metoda cílového objektu vrátit 0 (nula) tak že napíšete $0:=0. K odmítnutí tažení musí metoda objektu vrátit -1 (mínus 1) tak, že napíšete $0:=-1. Během události Při odtažení, 4D zachází s metodou objektu jako s funkcí. Pokud není vrácen žádný výsledek, 4D bude předpokládat, že je tažení přijato.

Pokud přijmete přetažení, je cílový objekt zvýrazněn. Pokud odmítnete tažení, není cílový objekt zvýrazněn. Přijetí přetažení ještě neznamená, že cílový objekt musí přijmout přetahovaná data. Pouze to znamená, že pokud nad tímto objektem pustíte tlačítko myši, tak se cílový objekt pokusí data přijmout.

Pokud neprovedete událost Při odtažení, je cílový objekt zvýrazněn vždy, bez rozdílu podstaty nebo typu tažených dat a taženého objektu.

Událost Při odtažení je první úroveň kontroly pro dokončení tažení a vsazení. Při této události nemusíte testovat pouze typ dat zdrojového objektu a jeho kompatibilitu s cílovým objektem; můžete testovat také uživatele který operaci provádí, protože zvýraznění cílového objektu je pouze na vaší volbě.

Kód pod událostí Při odtažení by měl být krátký, protože se posílá i několikrát za sebou podle pohybů myši.

UPOZORNĚNÍ: Pokud je přetažení a vsazení **meziprocesové** což znamená že zdrojový objekt je v jiném procesu (okně) než cílový objekt, je metoda cílového objektu prováděna v **kontextu zdrojového procesu** (proces zdrojového objektu) a **ne** procesu cílového objektu. Toto je jediný případ ve kterém se objevuje takový typ provádění. Popis tohoto typu provádění je na konci této části.

Při vsazení

Událost Při vsazení je poslána jednou cílovému objektu, když uživatel pustí tlačítko myši nad cílovým objektem. Tato událost je druhá fáze operace tažení a vsazení, ve které provádíte operaci podle akce uživatele.

Tato událost nebude poslána, pokud nebylo přetažení schváleno při události Při odtažení. Pokud jste během provedení události testovali data ve zdrojovém a cílovém objektu a přetažení bylo schváleno, nemusíte již tento test provádět během události Při vsazení. Nyní již víte, že data jsou vhodná pro cílový objekt.

Zajímavé na přetažení a vsazení ve 4D je to, že 4D vám umožní dělat cokoli chcete.

Příklady:

• Pokud je přetažena položka hierarchického seznamu do textového pole, můžete text z této položky vložit na začátek, konec nebo do středu textového pole.
• Váš formulář obsahuje obrázkové tlačítko s dvěmy nastaveními, která znamenají prázdný nebo plný koš. Vsazení objektu na toto tlačítko může znamenat (z hlediska uživatele) "vymaž zdrojový objekt a vlož jej do koše." Zde tažení a vsazení netransportuje data z jednoho místa na druhé, ale provádí akci.
• Přetažení prvku array z plovoucího okna seznamu zákazníků do objektu ve formuláři může znamenat "v tomto okně mi ukaž záznam Zákazníka, jehož název byl právě přetažen a vsazen z plovoucí palety seznamu zákazníků v databázi."
• A tak dále.

Tažení a vsazení ve 4D je víceúčelová funkce, pomocí které můžete splnit libovolný požadavkek na rozhraní uživatele pro který se rozhodnete.

Příkazy Tažení a vsazení

Příkaz DRAG AND DROP PROPERTIES vrací:

• ukazatel na tažený objekt (proměnnou nebo pole)
• číslo prvku, pokud je tažený objekt prvek array nebo seznamu
• číslo zdrojového procesu

Příkaz Drop position vrací číslo prvku či umístění položky cílového prvku nebo položky seznamu, jestliže je cílový objekt array (např. posuvná oblast) nebo hierarchický seznam.

Příkaz RESOLVE POINTER a Type jsou použitelné pro testování podstaty a typu zdrojového objektu.

Pokud je operace tažení a vsazení připravena kopírovat tažená data, funkčnost těchto příkazů zavisí na tom kolik procesů je dotčeno:

• Pokud je přetažení a vsazaní omezeno na jeden proces, použijte tyto příkazy k provedení požadované akce (jednoduché přiřazení zdrojového objektu do cílovému objektu)
• Pokud je přetažení a vsazení meziprocesní, musíte si dávat pozor při přístupu k taženým datům: musíte zpřístupnit data ze zdrojového procesu. Pokud jsou data uložena v proměnné, použijte příkaz GET PROCESS VARIABLE k získání správné hodnoty. Pokud jsou tažená data z pole, nezapomeňte, že v jiném procesu může být jiný platný záznam a vy potřebujete přístup ke správnému záznamu.

V posledním případě je možné několik řešení:
• Jestliže je událost Při odtažení prováděna v kontextu zdrojovém procesu, můžete zkopírovat data a číslo záznamu do meziprocesové proměnné, kterou použijete při události Při vsazení.
• Můžete přenést data pomocí meziprocesové komunikace během události Při vsazení.

Pokud není tažení a vsazení použito pro přenos dat, ale pouze jako součást rozhraní uživatele, můžete provádět cokoli.

Dále si přečtěte

DRAG AND DROP PROPERTIES, Drop position, Form event, GET PROCESS VARIABLE, Is a list, RESOLVE POINTER, Type.

Příkazy a odkazy pro Táhnout a vsadit

4th Dimension 6.5, Seznam témat příkazů

Drop position (Umístění vsazení)

Příkazy a odkazy pro Táhnout a vsadit

Verze 6.0

Drop position Číslo

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Číslo	←	Číslo prvku nebo umístění položky nebo -1 pokud bylo umístění provedeno za poslední prvek array nebo položku seznamu

Popis

Příkaz Drop position vrací číslo prvku nebo pozici položky na kterou byl objekt přetažen a vsazen.

Typické použití Drop position je pokud objekt, do kterého vsadíte tažený, je array nebo hierarchický seznam.

Pokud je cílový objekt array, příkaz vrátí číslo prvku. Jestliže cílový objekt je hierarchický seznam, příkaz vrátí pozici položky v seznamu. V obou případech vrátí příkaz -1 pokud byl objekt vsazen za poslední prvek nebo položku.

Pokud voláte Drop position při události, která se netýká tažení a vsazení, ale která se děje na array nebo hierarchickém seznamu, vrátí příkaz -1.

Důležité: Objekt formuláře přijme tažená data, pokud má označenu vlastnost **Vsaditelné**. Metoda tohoto objektu musí být spuštěna při události Při odtažení a nebo Při vsazení, podle pořadí provedení těchto událostí.

Příklad

Podívejte se na příklad u příkazu DRAG AND DROP PROPERTIES.

Dále si přečtěte

Táhnout a vsadit, DRAG AND DROP PROPERTIES.

Příkazy a odkazy pro Táhnout a vsadit

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DRAG AND DROP PROPERTIES (VLASTNOSTI TAŽENÍ A VSAZENÍ)

Příkazy a odkazy pro Táhnout a vsadit

Verze 6.0

DRAG AND DROP PROPERTIES (zdrObjekt; zdrPrvek; zdrProces)

Parametr	Typ		Popis
zdrObjekt	Ukazatel	→	Ukazatel na taženžý objekt
zdrPrvek	Číslo	←	Číslo taženého prvku array, nebo číslo položky v hierarchickém seznamu, nebo -1 pokud objekt není ani array ani seznam
zdrProces	Číslo	←	Číslo zdrojového procesu

Popis

Příkaz DRAG AND DROP PROPERTIES vám umožní získat informace o zdrojovém objektu při provedení událostí Při vsazení a Při odtažení.

Typické použití příkazu DRAG AND DROP PROPERTIES je v metodě objektu, která se spustí při událostech Při vsazení a Při odtažení.

Důležité: Objekt formuláře přijme tažená data, pokud má označenu vlastnost **Vsaditelné**. Metoda tohoto objektu musí být spuštěna při události Při odtažení a nebo Při vsazení, podle pořadí provedení těchto událostí.

Po provedení příkazu:

• Parametr *zdrObjekt* je ukazatel na zdrojový objekt (objekt, který je tažený a vsazený). Nezapomeňte, že cílový objekt (objekt, pro který se aktivují události Při vsazení a Při odtažení) může být jak ten samý objekt tak i zcela jiný objekt. Přetažení a vsazení z a do stejného objektu je použitelné pro array a hierarchické seznamy - je to jednoduchý způsob jak umožnit uživateli setřídit si prvky v array nebo položky v seznamu.
• Pokud je přetažený a vsazený objekt prvek array, vrací parametr *zdrPrvek* číslo prvku v array. Jinak pokud je zdrojový objekt seznam, vrací tento parametr pozici položky v seznamu. Pokud není zdrojový objekt ani array ani seznam vrátí tento parametr hodnotu -1.
• Tažení a vsazení může probíhat i mezi procesy. Parametr *zdrProces* vrací číslo procesu, ve kterém je zdrojový objekt. Je důležité testovat tento parametr. Na tažení a vsazení ve stejném procesu můžete odpovědět jednoduchým kopírováním dat zdrojového objektu do cílového objektu. Jinak, pokud provádíte meziprocesové tažení a vsazení, by jste měli použít příkaz GET PROCESS VARIABLE k získání dat ze zdrojového objektu. Pokud jsou zdrojová data typu pole, musíte použít meziprocesovou komunikaci nebo řídit tento případ z metody, která se spustí po události Při odtažení (čtěte dále).

Nicméně většinou provádíte tažení a vsazení ze zdrojových proměnných (array, seznamy) a ne z oblastí pro vstup dat (pole a proměnné).

Pokud voláte DRAG AND DROP PROPERTIES a neprovádíte žádné přetahování, vrátí *zdrObjekt* prázdný ukazatel, *zdrPrvek* vrátí -1 a *zdrProces* vrátí 0.

Tip: 4th Dimension automaticky ovládá grafické aspekty tažení a vsazení. Na událost pak musíte odpovědět patřičným způsobem. V následujícím příkladu je odpovědí kopírování dat, která byla přetažena. Případně můžete vložit složitější rozhraní uživatele, kde například, přetažení a vsazení prvku array z plovoucího okna vyplní cílové okno (okno kde je umístěn cílový objekt) strukturovanými daty (např. poli ze záznamu, který je jednoznačně definovan prvkem přetaženým z array).

Příkaz DRAG AND DROP PROPERTIES používáte při události Při odtažení, aby jste získali informace o zdrojovém objektu a zjištění jestli jej cílový objekt přijme (nebo z jiných důvodů). Pokud přijímáte tažení a vsazení, musí metoda objektu vrátit $0:=0. Pokud nepřijímáte tažení a vsazení, musí metoda objektu vrátit $0:=-1. Přijetí nebo odmítnutí se projeví na obrazovce - cílový objekt je zvýrazněný vybraným způsobem, pokud tento objekt a

jeho data přijímá.

Tip: Během události Při odtažení je metoda objektu cílového objektu provedena v procesu zdrojového objektu. Pokud je zdrojový objekt meziprocesního tažení a vsazení pole, můžete využít tuto metodu ke zkopírování dat do meziprocesové proměnné. Pokud to uděláte, nebudete muset aktivovat meziprocesní komunikaci se zdrojovým procesem pro získání hodnoty taženého objektu. Pokud meziprocesní tažení a vsazení používá proměnnou jako zdrojový objekt, můžete potom použít příkaz GET PROCESS VARIABLE během události Při vsazení.

Příklady

1. V mnoha vašich formulářích budete pravděpodobně používat posuvné oblasti, ve kterých budete chtít měnit ručně pořadí pouhým potažením a vsazením. Spíše než psát specielní kód pro každý případ, je jednodušší napsat generickou metodu projektu která bude ovládat libovolno z těchto posuvných oblastí. Můžete napsat něco takového:

```
   ` Metoda projektu Potažení a vsazení na sebe sama
   ` Potažení a vsazení na sebe sama ( Ukazatel ) → Logické
   ` Potažení a vsazení na sebe sama ( → Array ) → Bylo tažení a vsazení do téhož array
Case of
   ÷ (Form event=On Drag Over)
      DRAG AND DROP PROPERTIES($vpSrcObj;$vlSrcElem;$vlPID)
      If ($vpSrcObj=$1)
            ` Přijmout tažení a vsazení pokud je to z téhož array
         $0:=0
      Else
         $0:=-1
      End if
   ÷ (Form event=On Drop)
         ` Vzít informace o zdrojovém objektu
      DRAG AND DROP PROPERTIES($vpSrcObj;$vlSrcElem;$vlPID)
         ` Vzít číslo cílového prvku
      $vlDstElem:=Drop position
         ` Pokud nebyl prvek položen sám na sebe
      If ($vlDstElem    # $vlSrcElem)
            ` Uložit tažený prvek do prvku 0
         $1→{0}:=$1→{$vlSrcElem}
            ` Vymazat tažený prvek
         DELETE ELEMENT($1→;$vlSrcElem)
            ` Pokud byl cílový prvek za taženým prvkem
         If ($vlDstElem>$vlSrcElem)
               ` Snížení čísla cílového prvku
            $vlDstElem:=$vlDstElem-1
         End if
            ` Pokud bylo tažení a vsazení provedeno za posledním prvkem
         If ($vlDstElem=-1)
               ` Nastavit číslo cílového prvku na nový prvek
               ` na konci array
            $vlDstElem:=Size of array($1→)+1
         End if
            ` Vložit nový prvek
         INSERT ELEMENT($1→;$vlDstElem)
            ` Nastavit hodnotu která byla předtím uložena v prvku nula
         $1→{$vlDstElem}:=$1→{0}
            ` Prvek se stane novým označeným prvkem v array
```

```
            $1→:=$vlDstElem
        End if
    End case
```

Jakmile předáte tuto metodu,můžete ji použít následujícím způsobem:

```
    ` Metoda objektu posuvná oblast nějakéArray
    Case of
        `...
    ÷ (Form event=On Drag Over)
        $0:= Potažení a vsazení na sebe sama (Self)
    ÷ (Form event=On Drop)
        Potažení a vsazení na sebe sama (Self)
            ` ...
    End case
```

2. V některých formulářích můžete mít textovou oblast do které budete chtít přetahovat a vsazovat hodnoty z různých zdrojů. Lepší než psát specialní kód pro každý případ, je lepší napsat generickou metodu projektu, která bude ovládat každou z těchto textových oblastí. Můžete napsat něco jako toto:

```
    ` Metoda projektu Vsazení do textové oblasti
    ` Vsazení do textové oblasti ( ukazatel )
    ` Vsazení do textové oblasti ( → Textová nebo řetězcová proměnná )
    Case of
        ` Použijte tuto událost pro přijetí nebo odmítnutí tažení a vsazení
    ÷ (Form event=On Drag Over)
            ` Nastavit $0 na odmítnutí
        $0:=-1
            ` Vzít informace o zdrojovém objektu tažení a vsazení
        DRAG AND DROP PROPERTIES($vpSrcObj;$vlSrcElem;$vlPID)
            ` V tomto příkladu umožňujeme tažení a vsazení z objektu do sebe sama
        If ($vpSrcObj   # $1)
                ` Vzít typ dat, která se přetahují
            $vlSrcType:=Type($vpSrcObj→)
            Case of
                ÷ ($vlSrcType=Is Alpha Field)
                        ` Alfanumerické pole je OK
                    $0:=0
                        ` Zkopírovat hodnotu do meziprocesní proměnné
                    <>vtDraggedData:=$vpSrcObj→
                ÷ ($vlSrcType=Is Text)
                        ` Textové pole nebo proměnná je OK
                    $0:=0
                    RESOLVE POINTER($vpSrcObj;$vsPromNazev;$vlTabulkaNum; $vlPoleNum)
                        ` Pokud je to pole
                    If (($vlTabulkaNum>0) & ($vlPoleNum>0))
                            ` Zkopírovat hodnotu do meziprocesní proměnné
                        <>vtDraggedData:=$vpSrcObj→
                    End if
                ÷ ($vlSrcType=Is String Var)
                        ` Řetězcová proměnná je OK
```

```
            $0:=0
        ÷ (($vlSrcType=String array) | ($vlSrcType=Text array))
                ` Textový a Řetězcový array je OK
            $0:=0
        ÷ (($vlSrcType=Is LongInt) | ($vlSrcType=Is Real)
            If (Is a list($vpSrcObj→))
                    ` Hierarchický seznam je OK
                $0:=0
            End if
        End case
    End if
        ` Použít tuto událost k provedení akce táhnout a vsadit
÷ (Form event=On Drop)
    $vtDraggedData:=""
        ` Vzít informace o zdrojovém objektu
    DRAG AND DROP PROPERTIES($vpSrcObj;$vlSrcElem;$vlPID)
    RESOLVE POINTER($vpSrcObj;$vsPromNazev;$vlTabulkaNum;$vlPoleNum)
        ` Pokud je to pole
    If (($vlTabulkaNum>0) & ($vlPoleNum>0))
            ` Vzít data z meziprocesní proměnné vytvořené během události Při odtažení
        $vtDraggedData:=<>vtDraggedData
    Else
            ` Vzít typ proměnné, která je přetažena
        $vlSrcType:=Type($vpSrcObj→)
        Case of
                ` Jesltiže to je array
            ÷ (($vlSrcType=String array) | ($vlSrcType=Text array))
                If ($vlPID   # Current process)
                        ` Přečíst prvek z výskytu proměnné ve zdrojovém procesu
                    GET PROCESS VARIABLE($vlPID; $vpSrcObj→{$vlSrcElem};$vtDraggedData)
                Else
                        ` Zkopírovat prvek array
                    $vtDraggedData:=$vpSrcObj→{$vlSrcElem}
                End if
            ÷ (($vlSrcType=Is LongInt) | ($vlSrcType=Is Real)
                    ` Pokud je to hierarchický seznam
                If (Is a list($vpSrcObj→))
                        ` Pokud je to seznam z jiného procesu
                    If ($vlPID   # Current process)
                            ` Vzít odkaz na seznam z jiného procesu
                        GET PROCESS VARIABLE($vlPID;$vpSrcObj→;$vlList)
                    Else
                        $vlList:=$vpSrcObj→
                    End if
                        ` Vzít text z položky seznamu, která byla získána
                    GET LIST ITEM($vlList;$vlSrcElem;$vlPolozkaRef;$vsPolozkaText)
                    $vtDraggedData:=$vsPolozkaText
                End if
            Else
                    ` Je to textová nebo řetězcová proměnná
                If ($vlPID   # Current process)
                    GET PROCESS VARIABLE($vlPID;$vpSrcObj→;$vtDraggedData)
```

```
                    Else
                        $vtDraggedData:=$vpSrcObj→
                    End if
                End case
            End if
                ` Pokud je co vsadit (zdrojový objekt může být prázdný)
            If ($vtDraggedData   # "")
                    ` Testovat jestli délka proměnné nepřesáhla   32,000 znaků.
                If ((Length($1→)+Length($vtDraggedData))<=32000)
                    $1→:=$1→+$vtDraggedData
                Else
                    BEEP
                    ALERT("Operace nemohla být dokončena, protože text je příliš dlouhý.")
                End if
            End if
        End case
```

Jakmile předáte tuto metodu do databáze, můžete ji použít následujícím způsobem:

```
        ` Metoda objektu [NějakáTabulka]Textové pole
    Case of
            ` ...
        ÷ (Form event=On Drag Over)
            $0:= Vsazení do textové oblasti (Self)
        ÷ (Form event=On Drop)
            Vsazení do textové oblasti (Self)
                ` ...
    End case
```

Déle si přečtěte

Táhnout a vsadit, Drop position, Form event, GET PROCESS VARIABLE, Is a list, RESOLVE POINTER.

Příkazy a odkazy pro Táhnout a vsadit

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ACCEPT (PŘIJMOUT)

Příkazy a odkazy pro Kontrola vstupu

Verze 3

ACCEPT

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz ACCEPT je používán v metodě formuláře nebo objektu, aby provedl:
• přijmutí nového nebo upraveného záznamu nebo podzáznamu, pro který byl vstup zahájen příkazem ADD RECORD, MODIFY SELECTION, ADD SUBRECORD nebo MODIFY SUBRECORD.
• potvrzení dialogu zobrazeného pomocí příkazu DIALOG.
• opuštění formuláře pro zobrazení seznamu záznamů s otevřeného použitím DISPLAY SELECTION nebo MODIFY SELECTION.

Příkaz ACCEPT provede stejnou akci jako když uživatel stiskne klávesu Enter. Po přijetí formuláře je systémová proměná OK nastavena na 1.

ACCEPT je často používán jako akce po vybrání určité položky nabídky. Používá se také pro akci tlačítka, které má nastavenu automatickou akci "Žádná akce".

Používá se také pro metodu uzavíracího políčka okna oevřeného příkazem Open window. Pokud existuje políčko řízení pro okno, mohou být volány příkazy ACCEPT nebo CANCEL v metodě, která se spustí po poklepání na políčko řízení nebo po klepnutí na políčko uzavření.

ACCEPT se nemůže bufferovat. Výsledek spuštění dvou příkazů ACCEPT po sobě, uvnitř jedné metody, je stejný jako volání jen jedenoho příkazu.

Dále si přečtěte

CANCEL.

Příkazy a odkazy pro Kontrola vstupu

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CANCEL (ZRUŠIT)

Příkazy a odkazy pro Kontrola vstupu

Verze 3

CANCEL

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz CANCEL je používán v metodě formuláře nebo objektu, aby provedl:

• zrušení/storno nového nebo upraveného záznamu nebo podzáznamu, pro který byl vstup zahájen příkazem ADD RECORD, MODIFY SELECTION, ADD SUBRECORD nebo MODIFY SUBRECORD.

• zrušení dialogu zobrazeného pomocí příkazu DIALOG.

• opuštění formuláře pro zobrazení seznamu záznamů otevřeného použitím DISPLAY SELECTION nebo MODIFY SELECTION.

CANCEL provede stejnou akci jako když uživatel stiskne kombinaci kláves zrušení/ storno (Ctrl-tečka na Windows nebo Command-tečka na Macintoshi). Po provedení příkazu CANCEL je systémová proměnná OK nastavena na 0.

CANCEL se obvykle používá jako výsledek po vybrání položky nabídky. Je také používán jako metoda objektu tlačítka, pro které byla zvolena automatická akce "Žádná akce".

Používá se také pro metodu uzavíracího políčka okna otevřeného příkazem Open window. Pokud existuje políčko řízení pro okno, mohou být volány příkazy ACCEPT nebo CANCEL v metodě, která se spustí po poklepání na políčko řízení, nebo po klepnutí na políčko uzavření.

CANCEL se nemůže bufferovat. . Výsledek spuštění dvou příkazů CANCEL po sobě je stejný jako provedení jednoho příkazu.

Dále si přečtěte

ACCEPT.

Příkazy a odkazy pro Kontrola vstupu

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Keystroke (Stisknuta klávesa)

Příkazy a odkazy pro Kontrola vstupu

Verze 6.0

Keystroke → řetězec

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	řetězec	←	znaky předáné uživatelem

Popis

Keystroke vrací znaky zadané uživatelem do pole nebo proměnné.

Většinou budete používat příkaz Keystroke v metodě formuláře, nebo objektu při zachytávání událostí, Při úhozu na klávesu. K detekování události Při úhozu na klávesu, použijte příkaz Form event.

K nahrazení znaků zadaných uživatelem použijte příkaz FILTER KEYSTROKE.

Poznámka: Funkce Keystroke nefunguje v podformuláři.

DŮLEŽITÁ POZNÁMKA: Pokud chcete provést nějaké operace "za běhu" podle platné hodnoty v dostupné oblasti, nebo podle toho jakou klávesu uživatel stiskne, nezapomeňte, že hodnota textu kterou právě vidíte na obrazovce NENÍ hodnota dat zdrojového pole, nebo proměnné pro oblast kterou upravujete. Ke zdroji dat pole nebo proměnné je přiřazena zadaná hodnota po potvrzení vstupu (přechod tabelátorem do jiné oblasti, klepnutí na tlačíto, atd.). Musíte nejdříve přesunout vstup dat do jiné proměnné a pak s touto "stínovou" proměnnou pracovat. Jestliže potřebujete vědět platnou textovou hodnotu pro provedení jiné akce nemáte jinou možnost jak ji zjistit.

Příkaz Keystroke můžete použít pro:

- filtrování znaků podle vlastních pravidel
- filtrování vstupu dat způsobem který nemůžete udělat pomocí vstupních filtrů
- předání dynamického prohlížení, nebo psaní před přijetím vstupu do oblasti.

Příklady

Podívejte se na příklad u příkazu FILTER KEYSTROKE.

2. Když zpracováváte událost Při úhozu na klávesu, pracujete se zadávací textovou oblastí (oblast ve které je kurzor), ne s budoucí hodnotou zdroje dat (pole nebo proměnné) pro tuto oblast. Metoda *Ovládání úhozů* z dalšího příkladu vám umožní přesunout data ze zadávací oblasti do jiné stínové proměnné, ve které můžete provádět akce se znaky zadávanými do oblasti. Předáte do ní jako parametry ukazatel na zdrojová data a druhý bude ukazatel na stínovou proměnnou. Metoda vrací novou hodnotu pro textovou oblast do stínové proměnné a vrací True, pokud je hodnota současně zadaného znaku odlišná od předchozího posledního zadaného znaku.

```
    ` Metoda projektu Ovládání úhozů
    ` Ovládání úhozů ( Ukazatel ; Ukazatel ) → Logické
    ` Ovládání úhozů ( → zdrOblast ; → platnáHodnota ) → Je nová hodnota
C_POINTER ($1;$2)
C_TEXT ($vtNovyZnak)
    ` Vezme rozsah výběru textu v dostupné oblasti
GET HIGHLIGHT ($1→;$vlStart;$vlKonec)
    ` Začít pracovat s platnou hodnotou
$vtNovyZnak:=$2→
    ` Podle stisknuté klávesy nebo zadaného znaku,
```

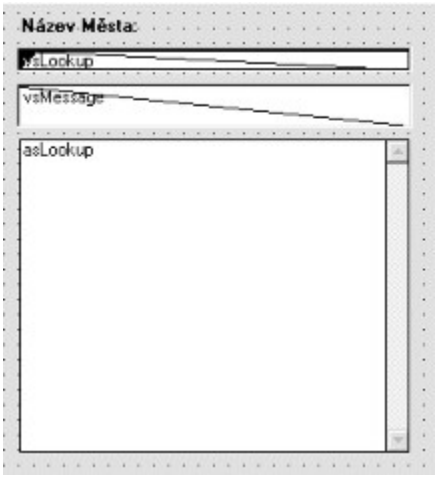
```
    ` provést patřičnou akci
Case of
        ` Byla stisknuta klávesa Backspace (Delete)
    ÷ (Ascii (Keystroke)=Backspace )
            ` Smazat označené znaky nebo znak nalevo od kurzoru
        $vtNovyZnak:=Substring ($vtNovyZnak;1;$vlStart-1- Num ($vlStart=$vlKonec))
                        +Substring($vtNovyZnak;$vlKonec)
            ` Byl předán použitelný znak
    ÷ (Position (Keystroke;"abcdefghjiklmnopqrstuvwxyz -0123456789")>0)
        If ($vlStart#$vlKonec)
                ` jeden nebo více znaků bylo označeno,
                ` úhoz na klávesu je přepíše
            $vtNovyZnak:=Substring($vtNovyZnak;1;$vlStart-1)+Keystroke
                            + Substring($vtNovyZnak;$vlKonec)
        Else
                ` Textový výběr je textový kurzor
            Case of
                        ` Textový kurzor je na začátku textu
                ÷ ($vlStart<=1)
                        ` Vložit znak na začátek textu
                    $vtNovyZnak:=Keystroke+$vtNovyZnak
                        ` Textový kurzor je na konci textu
                ÷ ($vlStart>=Length($vtNovyZnak))
                        ` Připojit znak na konec textu
                    $vtNovyZnak:=$vtNovyZnak+Keystroke
            Else
                    ` Textový kurzor je někde v textu, vložit nový znak
                $vtNovyZnak:=Substring($vtNovyZnak;1;$vlStart-1)+Keystroke
                                +Substring($vtNovyZnak;$vlStart)
            End case
        End if
            ` Byla stisknuta šipka
            ` Neudělat nic, ale přijmout stisknutí klávesy
    ÷ (Ascii(Keystroke)=Left Arrow Key )
    ÷ (Ascii(Keystroke)=Right Arrow Key )
    ÷ (Ascii(Keystroke)=Up Arrow Key )
    ÷ (Ascii(Keystroke)=Down Arrow Key )
        ` ...
Else
    ` nepřijmout jiné znaky než písmena, číslice, mezera a pomlčka
    FILTER KEYSTROKE ("")
End case
    ` Je nyní hodnota odlišná?
$0:=($vtNovyZnak#$2→)
    ` vrátit hodnotu pro další ovládání úhozu na klávesu
$2→:=$vtNovyZnak
```

Po uložení této metody do databáze, ji můžete použít následovně:

```
    ` Metoda dostupného objektu mujObjekt
Case of
    ÷ (Form event=On Load)
        mujObjekt:=""
```

```
        mujStinovyObjekt:=""
    ÷ (Form event=On Keystroke)
        If (Ovládání úhozů (→mujObjekt;→mujStinovyObjekt))
            ` Provést patřičnou akci s použitím hodnoty uložené v mujStinovyObjekt
        End if
End case
```

Prozkoumejme následující část formuláře:



Je složen z následujících objektů: dostupná oblast vtLookup, nedostupná oblast vsMessage a posuvná oblast asLookup. Ve chvíli, kdy předáte znaky do vsLookup, metoda pro tento objekt provede hledání v tabulce [PSČ] a umožní tak uživateli nalézt města pouhým napsáním jedné číslice.
Metoda objektu vsLookup vypadá následovně:

```
        ` Metoda objektu vsLookup
    Case of
        ÷ (Form event=On Load )
            vsLookup:=""
            vsResult:=""
            vsMessage:="Zadejte první znaky města které hledáte."
            CLEAR VARIABLE(asLookup)
        ÷ (Form event=On Keystroke )
            If (Ovládání úhozů (→vsLookup;→vsResult))
                If (vsResult#"")
                    QUERY([PSČ];[PSČ]Město=vsResult+"@")
                    MESSAGES OFF
                    DISTINCT VALUES([PSČ]Město;asLookup)
                    MESSAGES ON
                    $vlVysledek:=Size of array(asLookup)
                    Case of
                        ÷ ($vlVysledek=0)
                            vsMessage:="Nebylo nalezeno žádné město."
                        ÷ ($vlVysledek=1)
                            vsMessage:="Jedno město nalezeno."
                    Else
                        vsMessage:=String($vlVysledek)+" měst nalezeno."
                    End case
                Else
                    DELETE ELEMENT(asLookup;1;Size of array(asLookup))
                    vsMessage:=" Zadejte první znaky města, které hledáte."
                End if
```

```
        End if
    End case
```

Zde je formulář v prostředí uživatele:

S pužitím meziprocesní komunikace, můžete jednoduše vytvořit uživatelské rozhraní, ve kterém jsou tyto možnosti umístěny do plovoucího okna, které komunikuje s ostatními procesy.

Dále si přečtěte

FILTER KEYSTROKE, Form event.

Příkazy a odkazy pro Kontrola vstupu

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

FILTER KEYSTROKE (FILTR ÚHOZU NA KLÁVESU)

Příkazy a odkazy pro Kontrola vstupu

Verze 6.0

FILTER KEYSTROKE (ZnakFiltru)

Parametr	Typ		Popis
ZnakFiltru	Řetězec	→	Znak filtrovaného úhozu na klávesu nebo prázdný řetězec k zrušení úhozu

Popis

FILTER KEYSTROKE vám umožní nahradit znaky zadané uživatelem do pole nebo dostupného objektu prvním znakem zadaným do parametru *ZnakFiltru*.

Pokud předáte prázdný řetězec, je úhoz na klávesu zrušen a ignorován.

Obvykle budete volat FILTER KEYSTROKE v metodě formuláře nebo objektu při události Při úhozu na klávesu. K detekování události Při úhozu na klávesu použijte příkaz Form event. K zachycení aktuální stisknuté klávesy použijte příkaz Keystroke.

DŮLEŽITÁ POZNÁMKA: FILTER KEYSTROKE vám umožní zrušit nebo nahradit znaky zadané uživatelem jiným znakem. Pokud chcete vložit více než jeden znak pro specielní klávesu, nezapomeňte, že hodnota textu kterou právě vidíte na obrazovce NENÍ hodnota dat zdrojového pole nebo proměnné pro oblast, kterou upravujete. Ke zdroji dat pole nebo proměnné je přiřazena zadaná hodnota až po potvrzení vstupu (přechod tabelátorem do jiné oblasti, klepnutí na tlačítko, atd.). Musíte nejdříve přesunout vstup dat do jiné proměnné a pak s touto "stínovou" proměnnou pracovat. Jestliže potřebujete vědět platnou textovou hodnotu pro provedení jiné akce nemáte jinou možnost jak ji zjistit.

Příkaz FILTER KEYSTROKE můžete použít pro:

- filtrování znaků podle vlastních pravidel
- filtrování vstupu dat způsobem který nemůžete udělat pomocí vstupních filtrů
- předání dynamického prohlížení nebo psaní před přijetím vstupu do oblasti.

UPOZORNĚNÍ: Pokud voláte příkaz Keystroke po volání příkazu FILTER KEYSTROKE je vrácen znak který je již filtrován místo znaků které byly skutečně předány.

Příklady

1. Použití následujícího kódu:

```
    ` Metoda dostupnéh objektu mujObjekt
Case of
   ÷ (Form event=On Load )
      mujObjekt:=""
   ÷ (Form event=On Keystroke)
      If(Position( Keystroke;"0123456789")>0)
         FILTER KEYSTROKE("*")
      End if
End case
```

Všechny číslice předáné do oblasti mujObjekt jsou převedeny na hvězdičky.

2. Tento kód nahrazuje chování dostupné oblasti pro Hesla ve kterém jsou všechny zadané znaky nahrazeny (na obrazovce) náhodnými znaky:

```
    ` Metoda objektu vsHeslo dostupná oblast
Case of
```

```
    ÷ (Form event=On Load )
        vsHeslo:=""
        vsActualHeslo:=""
    ÷ (Form event=On Keystroke)
        Ovládání úhozů (→vsHeslo;→vsActualHeslo)
        If (Position( Keystroke;Char(Backspace )+Char(Left Arrow Key )
            +Char(Right Arrow Key ) +Char(Up Arrow Key ) +Char(Down Arrow Key ))=0)
            FILTER KEYSTROKE(Char(65+( Random%26)))
        End if
End case
```

Po té co byl vstup dat potvrzen, pošlete aktuální heslo do proměnné vsActualHeslo.

Poznámka: Metoda *Ovládání úhozů* byla vytvořena v příkladu u příkazu **Keystroke**.

3. Ve vaší alikaci máte nějaké textové oblasti, do kterých zadáte nějaké věty. Vaše aplikace také obsahuje tabulku slovníku s nejčastěji používanými slovy ve vaší databázi. Ve chvíli, kdy upravujete textovou oblast, můžete potřebovat rychle obdržet a vložit vstupy do slovníku založené na základě označených znaků v textu. Máte dva způsoby jak to udělat:

- Vytvořit tlačítka s přiřazenými klávesami, nebo
- Zachytit speciální klávesy během upravování textové oblasti

Tento příklad předvádí druhý způsob, založený na klávese Nápověda.

Jak je popsáno výše, během upravování textové oblasti, bude ke zdroji dat pro tuto oblast přiřazena zadaná hodnota až po té co potvrdíte vstup. Aby jste mohli do textové oblasti, ve chvíli kdy je tato oblast upravována, poslat a obdržet v ní záznamy ze slovníku, potřebujete stínovat vstup dat. Jako první dva parametry předáte ukazatele k zdrojové oblasti a ke stínové proměnné a pak předáte řetězec "zakázaných" znaků jako třetí parametr. Nezáleží na tom, jak bude znak zadán, metoda vrátí originální znak. Zakázané znaky jsou ty, které nechcete vložit do dostupné oblasti a chcete s nimi zacházet jako se specielními znaky.

```
    ` Metoda projektu Stínování úhozů
    ` Stínování úhozů ( Ukazatel ; Ukazatel; Řetězec ) → Řetězec
    ` Stínování úhozů ( → zdrOblast ; → platnáHodnota ; Filtr ) → Stará klávesa
C_STRING(1;$0)
C_POINTER($1;$2)
C_TEXT($vtNovyZnak)
C_STRING(255;$3)
    ` Vrací originální klávesu na kterou bylo klepnuto
$0:=Keystroke
    ` Vzít rozsah výběru textu v dostupné oblasti
GET HIGHLIGHT($1→;$vlStart;$vlKonec)
    ` Začít pracovat s platnou hodnotou
$vtNovyZnak:=$2→
    ` Podle stisknuté klávesy nebo zadaného znaku,
    ` provést patřičnou akci
Case of
        ` Byla stisknuta klávesa Backspace (Delete)
    ÷ (Ascii($0)=Backspace )
            ` Vymazat označené znaky nebo znak nalevo od kurzoru
        $vtNovyZnak:=Mazat text ($vtNovyZnak;$vlStart;$vlKonec)
            ` Byla stisknuta šipka
            ` Neudělat nic, ale přijmout klávesu
    ÷ (Ascii($0)=Left Arrow Key )
    ÷ (Ascii($0)=Right Arrow Key )
    ÷ (Ascii($0)=Up Arrow Key )
```

```
    ÷ (Ascii($0)=Down Arrow Key )
          ` Byl zadán přijatelný znak
    ÷ (Position($0;$3)=0)
       $vtNovyZnak:=Vložit text ($vtNovyZnak;$vlStart;$vlKonec;$0)
  Else
       ` Znak nebyl uznán
    FILTER KEYSTROKE("")
  End case
    ` Vrátit hodnotu pro další ovládání kláves
  $2→:=$vtNovyZnak
```

Tato metoda používá následující dvě podmetody:

```
    ` Metoda projektu Mazat text
    ` Mazat text ( Řetězec ; Long ; Long ) → Řetězec
    ` Mazat text ( → Text ; VýbStart ; VýbKonec ) → Nový text
  C_TEXT($0;$1)
  C_LONGINT($2;$3)
  $0:=Substring($1;1;$2-1- Num ($2=$3))+Substring($1;$3)

    ` Metoda projektu Vložit text
    ` Vložit text (Řetězec; Long ; Long ; String ) → Řetězec
    ` Vložit text ( → zdrText ; výbStart ; výbKonec ; Text k předání ) → Nový text
  C_TEXT($0;$1;$4)
  C_LONGINT($2;$3)
  $0:=$1
  If ($2#$3)
    $0:=Substring($0;1;$2-1)+$4+Substring($0;$3)
  Else
    Case of
      ÷ ($2<=1)
        $0:=$4+$0
      ÷ ($2> Length ($0))
        $0:=$0+$4
    Else
      $0:=Substring($0;1;$2-1)+$4+Substring($0;$2)
    End case
  End if
```

Po předání těchto metod projektu je můžete použít následujícím způsobem:

```
    ` Metoda objektu vsPopis dostupná oblast
  Case of
    ÷ (Form event=On Load )
      vsPopis:=""
      vsStinovyPopis:=""
        ` Ustanovit seznam "zakázaných" znaků zobrazených jako specielní znaky
        ` ( zde, v tomto příkladu, je filtrována pouze klávesa Nápověda)
      vsSpecialKeys:=Char(HelpKey)`klávesa Nápověda
    ÷ (Form event=On Keystroke )
      $vsKey:= Stínování úhozů (→vsPopis;→vsStinovyPopis ;vsSpecialKeys)
      Case of
        ÷ (Ascii($vsKey)=Help Key )
```

```
                ` Udělat něco když je tato klávesa stisknuta
                ` Zde, v tomto příkladu, musí být nalezen a předán vstup ve slovníku
            PROHLEDAT SLOVNIK (→vsPopis;→vsStinovyPopis)
        End case
    End case
```

Metoda projektu PROHLEDAT SLOVNIK je zobrazena níže. Její účel je použít stínovou proměnnou pro přeřazení dostupné oblasti která se upravuje:

```
    ` Metoda projektu PROHLEDAT SLOVNIK
    ` PROHLEDAT SLOVNIK ( Ukazatel ; Ukazatel)
    ` PROHLEDAT SLOVNIK ( → Dostupná oblast ; →StínováProměnná )
C_POINTER($1;$2)
C_LONGINT($vlStart;$vlKonec)
    ` Vzít rozsah výběru textu v dostupné oblasti
GET HIGHLIGHT($1→;$vlStart;$vlKonec)
    ` Vzít vybraný text nebo slovo nalevo od kurzoru
$vtVybranyText:= Ziskat vybrany text ($2→;$vlStart;$vlKonec)
    ` Je co hledat?
If ($vtVybranyText#"")
        ` Pokud výběr textu byl kurzor,
        ` výběr začne na slově předcházející kurzor
    If ($vlStart=$vlKonec)
        $vlStart:=$vlStart-Length ($vtVybranyText)
    End if
        ` Podívat se po prvním dostupném vstupu ve slovníku
    QUERY([Slovnik];[Slovnik]Vstup=$vtVybranyText+"@")
        ` Je to jeden?
    If (Records in selection([Slovnik])>0)
            ` Pokud ano, vložit to do stínového textu
        $2→:=Vložit text ($2→;$vlStart;$vlKonec;[Slovnik]Vstup)
            ` Kopírovat stínový text do oblasti která se upravuje
        $1→:=$2→
            `Nastavit výběr pouze po vstupu do slovníku
        $vlKonec:=$vlStart+Length([Slovnik]Vstup)
        HIGHLIGHT TEXT(vsKomentare;$vlKonec;$vlKonec)
    Else
            ` Není žádný shodný záznam ve slovníku
        BEEP
    End if
Else
        ` Není žádný označený text
    BEEP
End if
```

Metoda Ziskat vybrany text je zobrazena zde:

```
    ` Metoda projektu Ziskat vybrany text
    ` Ziskat vybrany text ( Řetězec ; Long ; Long ) → Řetězec
    ` Ziskat vybrany text ( Text ; VýbStart ; VýbKonec ) → označený text
C_TEXT($0;$1)
C_LONGINT($2;$3)
```

```
If ($2<$3)
   $0:=Substring($1;$2;$3-$2)
Else
   $0:=""
   $2:=$2-1
   Repeat
      If ($2>0)
         If (Position($1[[$2]];"    ,.!?:;()-___")=0)
            $0:=$1[[$2]]+$0
            $2:=$2-1
         Else
            $2:=0
         End if
      End if
   Until ($2=0)
End if
```

Dále si přečtěte

Form event, Keystroke.

Příkazy a odkazy pro Kontrola vstupu

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GOTO AREA (JÍT NA OBLAST)

Příkazy a odkazy pro Kontrola vstupu

Verze 3

GOTO AREA ({*; }objekt)

Parametr	Typ		Popis
*	*	→	Pokud definováno = objekt je název objektu Pokud vynecháno = objekt je pole nebo proměnná
objekt	Pole | Proměnná	→	Název objektu (pokud *) nebo pole nebo proměnná (* vynechána), na které přejít

Popis

Příkaz GOTO AREA je použit k vybrání objektu pro vstup dat *objekt*, jako aktivního objektu formuláře. Má stejnou funkci jako když uživatel na tento objekt (pole, proměnnou) klepne nebo přejde tabelátorem.

Pokud definujete volitelný parametr *, v parametru *objekt* definujete objekt názvem objektu (řetězec). Pokud tento parametr vynecháte, předáváte v parametru *objekt* název pole nebo proměnné.V tomto případě předáváte odkaz na pole nebo proměnnou (pouze objekty polí nebo proměnných), místo řetězce názvu objektu formuláře. Jestli chcete vědět více informací o názvech objektů přečtěte si část Vlastnosti objektů.

Poznámka: Tento příkaz funguje pouze ve vstupním formuláři. Nemá žádný vliv na oblasti v podformuláři.

Příklady

1. Příkaz GOTO AREA může být použit oběma způsoby:

```
GOTO AREA ([Lidé]Jméno) ` Odkaz na pole
GOTO AREA (*; "OblastVěk") ` Název objektu
```

2. Podívejte se na příklad u příkazu REJECT.

Dále si přečtěte

REJECT.

Příkazy a odkazy pro Kontrola vstupu

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

REJECT (ODMÍTNOUT)

Příkazy a odkazy pro Kontrola vstupu

Verze 3

REJECT {(pole)}

Parametr	Typ		Popis
pole	Pole	→	Pole k odmítnutí

Popis

Příkaz REJECT má dva způsoby použití. První je bez parametru. To odmítne celý vstup dat a donutí uživatele zůstat ve formuláři. Druhý způsob odmítne pouze vstup do pole a donutí uživatele zůstat v poli.

Poznámka: Před použitím tohoto příkazu by jste si měli promyslet vestavěné nástroje pro potvrzení dat.

První způsob použití REJECT zabrání uživateli potvrdit záznam dokud není kompletní. Stejného cíle můžete dosáhnout bez použití příkazu REJECT - k tlačítku s akcí Žádná akce přiřadíte klávesu Enter a použijete příkazy ACCEPT (po té co byla schválena kontrola správného zadání polí) nebo CANCEL, k potvrzení nebo zrušení záznamu. Je doporučeno použít tuto druhou techniku a nepoužívat první způsob použití REJECT.

Pokud použijete první způsob, zabráníte uživateli přijmout záznam spuštěním REJECT, většinou z důvodu že nebyl záznam dokončen nebo má nesprávné údaje. Pokud se uživatel pokusí nekompletní záznam přijmout, spustí se příkaz REJECT a zabrání uživateli záznam uložit: formulář zůstane zobrazený na obrazovce. Uživatel bude muset pokračovat v zadávání dokud nebude záznam přijatelný, nebo jej musí zrušit.

Nejlepší umístění tohoto příkazu je v metodě objektu pro tlačítko Přijmout přiřazeného ke klávese Enter. Tímto způsobem proběhne přijetí pouze pokud je záznam přijatelný a je potvrzen a uživatel nemůže obejít potvrzení stisknutím klávesy Enter.

Druhý způsob použití příkazu REJECT je jeho spuštění s parametrem pole. Kurzor zůstane v poli. Tento způsob použití REJECT donutí uživatele zadat správnou hodnotu do pole. Musí být použit ihned po upravení pole. Zda byl záznam změněn můžete zjistit pomocí příkazu Modified. REJECT lze také umístit do metody objektu pro dostupnou oblast. Tento příkaz nemá žádný vliv na pole v podformuláři.

Jak jeden, tak i druhý způsob použití REJECT musí být umístěn do metody formuláře nebo do metody aktuálně měněného objektu formuláře. Pokud používáte REJECT pro podformulář pro celostránkový vstup, umístěte jej do metody formuláře nebo objektu tohoto formuláře.

Můžete použít příkaz HIGHLIGHT TEXT k označení dat v poli, které bylo odmítnuto.

Příklady

1. Následující příklad je pro záznam bankovní transakce. Ukazuje první způsob použití příkazu REJECT v metodě objektu tlačítka Přijmout. Klávesa Enter byla přiřazena k tomuto tlačítku. To znamená že pokud uživatel stiskne Enter, je provedena metoda tohoto tlačítka. Pokud byla transakce prověřena, pak musí existovat číslo šeku. Pokud toto číslo neexistuje, je potvrzení zrušeno:

```
Case of
        ` Pokud bylo prověřeno bez čísla...
   ÷ (([Operace]Transakce="Šek") & ([Operace]Číslo šeku = ""))
      ALERT ("Prosím vyplňte číslo šeku.") ` Upozornit uživatele
      REJECT ` Odmítnout vstup
      GOTO AREA ([Operace]Číslo šeku) ` Jít na pole Číslo šeku
End case
```

2. Následující příklad je částí metody pro pole [Zaměstnanci]Plat. Metoda objektu testuje pole [Zaměstnanci]Plat a

odmítne potvrzení pokud je menší než 2500 Kč. Stejného cíle dosáhnete vyplněním minimální hodnoty pro pole v editoru formulářů:

```
If ([Zaměstnanci]Plat <2500)
   ALERT ("Plat musí být větší než 2 500,-Kč")
   REJECT ([Zaměstnanci]Plat)
End if
```

Dále si přečtěte

ACCEPT, CANCEL, GOTO AREA.

Příkazy a odkazy pro Kontrola vstupu

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Form event (Událost formuláře)

Příkazy a odkazy pro Události formuláře

Verze 6.0

Form event → Číslo

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Číslo	←	Číslo události formuláře

Popis

Funkce Form event vrací číselnou hodnotu udávající typ události formuláře, která se právě stala. Většinou použijete Form event v metodě objektu nebo formuláře.

4th Dimension obsahuje následující předdefinované konstanty:

Konstanta	*Č.*	*Událost*	*Popis*
On Load	*1*	*Při zavedení*	*Formulář bude zobrazen nebo tištěn*
On Unload	*24*	*Při vyvedení*	*Formulář bude zavřen nebo opuštěn*
On Validate	*3*	*Při potvrzení vstupu*	*Potvrzování vstupu dat*
On Clicked	*4*	*Při klepnutí*	*Bylo klepnuto na objekt*
On Double Clicked	*13*	*Při poklepnutí*	*Bylo poklepáno na objekt*
On Before Keystroke	*17*	*Před úhozem na klávesu*	*Před předáním znaku do objektu který má focus Get edited text vrací text bez nového znaku*
On After Keystroke	*28*	*Po úhozu na klávesu*	*Po předání znaku do objektu který má focus Get edited text vrací text včetně nového znaku*
On Getting Focus	*15*	*Při získání fokusu*	*Objekt formuláře získává focus*
On Losing Focus	*14*	*Při ztrátě fokusu*	*Objekt formuláře ztrácí focus*
On Activate	*11*	*Při aktivaci*	*Okno formuláře se stane oknem na popředí*
On Deactivate	*12*	*Při deaktivaci*	*Okno formuláře přestane být oknem na popředí*
On Outside Call	*10*	*Při vnějším volání*	*Formulář obdržel volání CALL PROCESS*
On Drop	*16*	*Při vsazení*	*Data byla vsazena do objektu*
On Drag Over	*21*	*Při odtažení*	*Data mohou být vsazena do objektu*
On Menu Selected	*18*	*Při volbě z nabídky*	*Byla vybrána položka nabídky*
On Data Change	*20*	*Při aktualizaci*	*Data objektu byla změněna*
On Plug in Area	*19*	*V oblasti plug in*	*Externí objekt si vyžaduje spuštění metody objektu*
On Header	*5*	*V záhlaví*	*Bude zobrazena nebo tištěna oblast záhlaví formuláře*
On Printing Detail	*23*	*Při tisku obsahu*	*Bude tištěna oblast obsahu formuláře*
On Pronting Break	*6*	*Při tisku zlomu*	*Jeden ze zlomů formuláře se bude tisknout*
On Printing Footer	*7*	*Při tisku zápatí*	*Bude se tisknout zápatí formuláře*
On Close Box	*22*	*Při zavření okna*	*Bylo klepnuto na zavírací políčko formuláře*
On Display Detail	*8*	*Při zobrazení obsahu*	*Záznam bude zobrazen v seznamu*
On Open Detail	*25*	*Při otevíraní obsahu*	*Je poklepáno na záznam a ten bude otevřen ve vstupním formuláři*
On Close Detail	*26*	*Při uzavření obsahu*	*Opouští se vstupní formulář a vrací se do výstupního*
On Resize	*29*	*Při změně velikosti*	*Okno formuláře mění velikost*
On Timer	*27*	*Při časování*	*Byl dosažen počet tiků definovaný SET TIMER.*

Události a metody

Když se uděje událost formuláře, provede 4th Dimension následující akce:

• Nejdříve projde všechny objekty formuláře a volá všechny metody objektů v této události zahrnuté, pro něž byla zatržena tato událost ve vlastnostech objektu.

• Pak zavolá metodu formuláře, jestliže je pro něj ve vlastnostech formuláře zatržena tato událost.

Nepředpokládejte, že metody objektů, jsou-li, budou volány v nějakém určitém pořadí. Jediné pravidlo je, že metody objektů jsou volány vždy předmetodou formuláře. Jestliže je objekt podformulář se seznamem záznamů v hlavním formuláři, jsou nejdříve volány metody objektů a pak metoda předáného formuláře. Poté je zavolána metoda hlavního/ rodičovského formuláře. Jinými slovy je-li objekt v podformuláři, používá 4D pro podformulář totéž pravidlo pro objekty a formulář jako pro normální formuláře.

Včetně události *Při zavedení* (On Load) a *Při vyvedení* (On Unload), jestliže není daná událost zatržena u formuláře a je u objektu bude metoda objektu při události volána. Jinými slovy zatržení či nezatržení události u formuláře nemá vliv na volání metod objektů při určité události.

Počet objektů zahrnutých do události závisí na povaze a druhu události:
• *Při zavedení* – Budou spuštěny všechny metody objektů, u nichž je ve vlastnostech objektu tato událost zatržena. Pak je-li tato událost zatržena ve vlastnostech formuláře bude spuštěna metoda formuláře.
• *Při aktivaci* a *Při změně velikosti* – nebude spuštěna žádná metoda formuláře, protože tato událost se týká formuláře jako celku a nikoliv jednotlivých objektů. Jestliže bude tato událost zatržena u formuláře bude spuštěna pouze metoda formuláře.
• *Při odtažení* – do této události mohou být zahrnuty pouze vsaditelné objekty, které mají toto událost zatrženu ve Vlastnostech formuláře. Metoda formuláře nebude volána.
• *Při časování* – tato událost je generována pouze jestliže metoda formuláře obsahuje předchozí volání příkazu SET TIMER.. Jestliže je u formuláře zatržena událost Při časování bude volána pouze metoda formuláře, žádný metoda objektu není do události zahrnuta.

Upozornění: Rozdílně oproti všem dalším událostem je metoda cílového objektu pro libovolný objekt provedena v kontextu procesu taženého zdrojového objektu a nikoliv v procesu cílového objektu, do nějž je vsazováno. Více informací viz příkaz DRAG AND DROP PROPERTIES a Drop position.

Následující tabulka ukazuje jak jsou volány metody objektu a formuláře podle jednotlivých typů:

Událost	***Metoda objektu***	***Metoda formuláře***	***Jaký objekt***
Při zavedení	*Ano*	*Ano*	*Všechny objekty*
Při vyvedení	*Ano*	*Ano*	*Všechny objekty*
Při potvrzení vstupu	*Ano*	*Ano*	*Všechny objekty*
Při klepnutí	*Ano (poklepatelné) (*)*	*Ano*	*Vybrané objekty*
Při poklepnutí	*Ano (poklepatelné) (*)*	*Ano*	*Vybrané objekty*
Před úhozem na klávesu	*Ano (pokud dostupné) (*)*	*Ano*	*Vybrané objekty*
Po úhozu na klávesu	*Ano (pokud dostupné) (*)*	*Ano*	*Vybrané objekty*
Při získání fokusu	*Ano (poklepatelné) (*)*	*Ano*	*Vybrané objekty*
Po ztrátě fokusu	*Ano (poklepatelné) (*)*	*Ano*	*Vybrané objekty*
Při aktivaci	*Nikdy*	*Ano*	*Žádné*
Při deaktivaci	*Nikdy*	*Ano*	*Žádné*
Při vnějším volání	*Nikdy*	*Ano*	*Žádné*
Při vsazení	*Ano (vsaditelné) (*)*	*Ano*	*Vybrané objekty*
Při odtažení	*Ano (vsaditelné) (*)*	*Nikdy*	*Vybrané objekty*
Při volbě z nabídky	*Nikdy*	*Ano*	*Žádné*
Při aktualizaci	*Ano (měnitelné) (*)*	*Ano*	*Vybrané objekty*
V oblasti plug in	*Ano*	*Ano*	*Vybrané objekty*
V záhlaví	*Ano*	*Ano*	*Všechny objekty*

Při tisku obsahu	*Ano*	*Ano*	*Všechny objekty*
Při tisku zlomu	*Ano*	*Ano*	*Všechny objekty*
Při tisku zápatí	*Ano*	*Ano*	*Všechny objekty*
Při uzavření okna	*Nikdy*	*Ano*	*Žádné*
Při zobrazení obsahu	*Ano*	*Ano*	*Všechny objekty*
Při otevření obsahu	*Nikdy*	*Ano*	*Žádné*
Při uzavření obsahu	*Nikdy*	*Ano*	*Žádné*
Při změně velikosti	*Nikdy*	*Ano*	*Žádné*
Při časování	*Nikdy*	*Ano*	*Žádné*

(*) Jestli chcete vědět více informací, přečtěte si část Události, Objekty a Vlastnosi která následuje.

DŮLEŽITÉ: Vždy si pamatujte, že událost, kterou označíte pro objekt nebo formulář vždy spustí metodu přiřazenou k objektu nebo formuláři. Důležité při označování metod v prostředí návrháře (v oknech Vlasntosti objektu nebo formuláře) je, že odznačením můžete omezit počet volání metod, a tím ovlivnit rychlost provádění vašeho formuláře.

UPOZORNĚNÍ: Události On Load (Při zavedení) a On Unload (Při vyvedení) jsou spuštěny pro objekty jestliže události jsou aktivovány jak v objektu tak ve formuláři. Pokud jsou události aktivovány pouze pro objekty, nebudou spuštěny; tyto dvě události musí být aktivovány na úrovni formuláře.

Události, Objekty a Vlastnosi

Metoda objektu je volána pokud se událost může pro objekt nastat, podle jejího typu a nastavení. Následující část popisuje události které mohou pro objekty nastat.

Poklepatelné objekty

Objekty na které lze klepnout jsou ovládány myší. Obsahují:

- Logické dostupné pole a proměnné
- Obrázkové pole nebo proměnné, jejichž formát zobrazení je nastaven na Na pozadí
- Tlačítka, výchozí tlačítka, přepínače, zaškrtávací tlačítka a tlačítka na síti
- 3D tlačítka, 3D přepínače a 3D zaškrtávací políčka
- Rozevírací nabídky, rozevírací hierarchické seznamy a obrázkové seznamy
- Rozevírací seznamy, nabídka/rozevírací seznam
- Posuvné oblasti, hierarchické seznamy
- Neviditelná tlačítka, zvýrazněná tlačítka, obrázkové přepínače
- Teploměry, pravítka, výseče (známé jako posuvné objekty)
- Ovládací karty
- Dělič

Po označení událostí On Clicked (Při klepnutí) a On Double Clicked (Při poklepnutí) pro jeden z těchto objektů, můžete detekovat a ovládat klepnutí či poklepnutí v nebo na objekt s použitím příkazu Form event, který vrátí událost Při klepnutí nebo Při poklepnutí podle případu.

Pro všechny tyto objekty nastane událost Při klepnutí jakmile je puštěno tlačítko myši. Nicméně jsou zde dvě výjimky:

- Neviditelné tlačítko - Událost Při klepnutí vznikne ihned po klepnutí a nečeká se až bude tlačítko myši puštěno.
- Posuvné objekty (teploměr, pravítko, výseč) - Pokud nastavení objektu vyžaduje, aby byla metoda provedena ihned po klepnutí, nastane událost Při klepnutí ihned po stisknutí myši.

Poznámka: Některé z těchto objektů mohou být aktivovány z klávesnice. Například jakmile má zaškrtávací políčko fokus, může být zadáno stisknutím Mezerníku. I v tomto případě je aktivována událost Při klepnutí.

UPOZORNĚNÍ: Text se seznamem není poklepatelný objekt. Text se seznamem musí být brán jako textová oblast, jejíž rozevírací seznam je pouze seznam výchozích hodnot. Texty se seznamem můžete ovládat pomocí událostí Před úhozem na klávesu (On Before Keystroke), Po úhozu na klávesu (On After Keystroke) a Při aktualizaci (On Data Change).

Objekty dostupné z klávesnice

Objekty dostupné z klávesnice jsou objekty, do kterých můžete zadávat data pomocí klávesnice, a ve kterých můžete filtrovat data na nižší úrovni než pomocí událostí Před úhozem na klávesu (On Before Keystroke) a Po úhozu na klávesu (On After Keystroke). Možnosti těchto událostí využijete především příkazem Get edited text.

Objekty dostupné z klávesnice jsou:

- Všechny dostupné objekty polí (mimo Obrázkových, Podformulářů a BLOBů)
- Všechny dostupné proměnné (mimo Obrázkových, BLOB, Ukazatelů a Array)
- Text se seznamem
- Hierarchické seznamy

Po označení událostí Před úhozem na klávesu (On Before Keystroke) a Po úhozu na klávesu (On After Keystroke) pro jeden z těchto objektů, můžete úhozy na klávesy v objektu detekovat a ovládat s použitím příkazu Form event, který vrátí Před úhozem na klávesu a potom Po úhozu na klávesu (Jestli chcete vědět více informací přečtěte si popis u příkazu Get edited text).

Poznámka: Příkaz POST KEY vytváří události Před úhozem na klávesu a Po úhozu na klávesu.

Měnitelné objekty

Měnitelné objekty jsou objekty které mají zdroj dat který může být změněn myší nebo klávesnicí; nejsou stejné jako ovladače uživatelského rozhraní ovládané událostí Při klepnutí. Obsahují:

- Všechny dostupné objekty polí (mimo Podformulářů a BLOBů)
- Všechny dostupné proměnné (mimo BLOB, Ukazatelů a Array)
- Text se seznamem
- Externí objekty (pro které je plné zadávání dat akceptováno 4D Extensions)

Tyto objekty vrací události Při aktualizaci (On Data Change). Po zatržení události Při aktualizaci pro jeden z těchto objektů, můžete detekovat a ovládat změny v jeho zdroji dat s použitím příkazu Form event, který vrátí Při aktualizaci.

Objekty dostupné tabelátorem

Objekty dostupné tabelátorem jsou ty, které získají fokus pokud na ně přejdete Tabelátorem a nebo na ně klepnete myší. Objekt který má fokus je schopen přijímat znaky (zapsané z klávesnice), které nejsou akcelerátory (Windows) nebo zkratky (Macintosh) k položkám nabídky nebo k objektům jako jsou Tlačítka.

Všechny objekty jsou dostupné tabelátorem MIMO NÁSLEDUJÍCÍCH:

- Nedostupné pole a proměnné
- Tlačítka (pokud jsou použita na MacOS)
- Tlačítka na síti
- 3D tlačítka, 3D přepínače a 3D zaškrtávací tlačítka
- Rozevírací nabídky, hierarchické rozevírací nabídky
- Nabídka/Seznam (pokud je použita na MacOS)
- Obrázkové nabídky
- Posuvné oblasti
- Neviditelná tlačítka, zvýrazněná tlačítka, přepínače
- Diagramy
- Externí objekty (pro které plný vstup dat není přijmut 4D Extensions)
- Ovládací karty

• Děliče

Po označení událostí Při získání fokusu a/nebo Při ztrátě fokusu pro objekty dostupné tabelátorem, můžete detekovat nebo řídit změnu focusu těchto objektů s použitím příkazu From event, který vrátí Při získání focusu nebo Při ztrátě focusu, podle případu.

Kategorie událostí

Události formuláře mohou být rozděleny do následujících kategorií:

• Základní události
Při zavedení, Při vyvedení, Při potvrzení vstupu, Při zobrazení obsahu, Při otevírání obsahu, Při uzavření obsahu
• Události patřící pouze formuláři
Při aktivaci, Při deaktivaci, Při vnějším volání, Při zavření okna, Při volbě z nabídky, Při časování, Při změně velikosti
• Události vztažené k akci uživatele
Při klepnutí, Při poklepání, Před úhozem na klávesu, Po úhozu na klávesu, Při získání fokusu, Při ztrátě fokusu, Při aktualizaci, V oblasti plug-in
• Události tažení a vsazení
Při vsazení, Při odtažení
• Události tisku
V záhlaví, Při tisku obsahu, Při tisku zlomu, Při tisku zápatí.

Kompatibilita mezi V6 a V3

Následující tabulka ukazuje ekvivatenty mezi událostmi V6 a prováděcími cykly V3.

Událost V6	***Prováděcí cyklus formátu V3***	***Příkaz V3***
Při zavedení	*Before fáze*	*Before*
Při vyvedení	*Žádný prováděcí cykl*	*Žádný*
Při potvrzení vstupu	*After fáze*	*After*
Při klepnutí	*Generická During fáze*	*During*
Při poklepnutí	*Generická During fáze*	*During*
Před klepnutím na klávesu	*Žádný prováděcí cykl*	*Žádný*
Po klepnutí na klávesu	*Žádný prováděcí cykl*	*Žádný*
Při získání fokusu	*Žádný prováděcí cykl*	*Žádný*
Při ztrátě fokusu	*Žádný prováděcí cykl*	*Žádný*
Při aktivaci	*Activated fáze*	*Activated*
Při deaktivaci	*Deactivated fáze*	*Deactivated*
Při vnějším volání	*Outside call fáze*	*Outside call*
Při vsazení	*Žádný prováděcí cykl*	*Žádný*
Při odtažení	*Žádný prováděcí cykl*	*Žádný*
Při výběru z nabídky	*Generická During fáze*	*During plus Menu selected*
Při aktualizaci	*Generická During fáze*	*During*
V oblasti plug in	*Generická During fáze*	*During*
V záhlaví	*Printing header fáze*	*In header*
Při tisku obsahu	*Generická During fáze*	*During*
Při tisku zlomu	*Printing break fáze*	*In break*
Při tisku zápatí	*Printing footer fáze*	*In footer*

Pokud objekt (pole nebo proměnná) má označenu možnost V3 Script only if modified, předvolby událostí jsou omezeny na všechny odpovídající During operace, které jsou možné jako náhrada za příkaz V3:

Událost V6	***Prováděcí cyklus formátu V3***	***Příkaz V3***
Při klepnutí	*Generická During fáze*	*During*
Při poklepnutí	*Generická During fáze*	*During*
Při aktualizaci	*Generická During fáze*	*During*
V oblasti plug in	*Generická During fáze*	*During*

Jakmile jednou začnete upravovat formulář a jeho objekty ve V6, jsou události formuláře a objektů, jako výchozí nastaveny na stejné schéma. Aby jste využili nové možnosti událostí V6, vyberte v prostředí návrháře možnosti událostí pro formulář a objekty a upravte metody objektu a formuláře pomocí příkazu Form event.

Nové události bez odpovídajících cyklů ve V3 jsou:

Událost V6	***Prováděcí cyklus formátu V3***	***Příkaz V3***
Při vyvedení	*Žádný prováděcí cyklus*	*Žádný*
Před klepnutím na klávesu	*Žádný prováděcí cyklus*	*Žádný*
Po klepnutí na klávesu	*Žádný prováděcí cyklus*	*Žádný*
Při získání fokusu	*Žádný prováděcí cyklus*	*Žádný*
Při ztrátě fokusu	*Žádný prováděcí cyklus*	*Žádný*
Při vsazení	*Žádný prováděcí cyklus*	*Žádný*
Při odtažení	*Žádný prováděcí cyklus*	*Žádný*
Při uzavření okna	*Žádný prováděcí cyklus*	*OPEN WINDOW (se zavíracím políčkem)*

Nové události které vám umožní provést akce lépe podle založení událostí:

Událost V6	***Prováděcí cyklus formátu V3***	***Příkaz V3***
Při klepnutí	*Generická During fáze*	*During*
Při poklepnutí	*Generická During fáze*	*During*
Při výběru z nabídky	*Generická During fáze*	*During plus Menu selected*
Při aktualizaci	*Generická During fáze*	*During*
V oblasti plug in	*Generická During fáze*	*During*
Při tisku obsahu	*Generická During fáze*	*During*
Při zobrazení obsahus	*Before a During fáze*	*Before & During*
Při tisku obsahu	*Generická During fáze*	*During*
Při tisku obsahu	*Generická During fáze*	*During*

Příklady

Ve všech těchto příkladech se počítá s tím, že budou patřičné události pro objekty a formuláře zaškrtnuté.

1. Tento příklad třídí výběr podzáznamů pro podtabulku [Rodiče]Děti před tím, než je formulář pro tabulku [Rodiče] zobrazen na obrazovku:

```
   ` Metoda formuláře pro tabulku [Rodiče]
Case of
   ÷ (Form event=On Load)
      ORDER SUBRECORDS BY ([Rodiče]Děti; [Rodiče]Děti'Jméno;>)
      ` ...
End case
```

2. Tento příklad ukazuje použití události Při potvrzení vstupu k automatickému přiřazení datumu změny(do pole), když byl záznam upraven:

```
   ` Metoda formuláře
Case of
      ` ...
   ÷ (Form event=On Validate)
      [aTabulka]Poslední změna:=Current date
End case
```

3. V tomto příkladu je ukázáno kompletní vytvoření a ovládání rozevíracího seznamu (inicializace, klepnutí uživatele a opuštění objektu):

```
   ` Metoda objektu asBurgerVelikost
Case of
   ÷ (Form event=On Load)
      ARRAY STRING(31;asBurgerVelikost;3)
      asBurgerVelikost{1}:="Malý"
```

```
        asBurgerVelikost{2}:="Střední"
        asBurgerVelikost{3}:="Velký"
    ÷ (Form event=On Clicked)
        If (asBurgerVelikost#0)
            ALERT("Vybral jste si "+asBurgerVelikost{asBurgerVelikost}+" hamburger.")
        End if
    ÷ (Form event=On Unload)
        CLEAR VARIABLE(asBurgerVelikost)
End case
```

4. Tento příklad ukazuje jak, v metodě objektu, přijmout a později ovládat operaci tažení a vsazení pro objekt pole, který přijímá pouze obrázky:

```
    ` metoda objektu [aTabulka]aObrazek dostupné obrázkové pole
Case of
    ÷ (Form event=On Drag Over)
            ` Operace tažení a vsazení byla započata a myš je nad polem
            ` Vzít informace o zdrojovém objektu
        DRAG AND DROP PROPERTIES ($vpSrcObject;$vlZdrPrvek; $lZdrProcess)
            ` Nezapomeňte že nepotřebujeme testovat číslo zdrojového procesu,
            ` pro metodu objektu, protože ta je vyjímečně prováděna v obsahu toho
                ` procesu
        $vlTypDat:=Type ($vpZdrObjekt→)
            ` Jsou zdrojová data obrázek (fiepoleld, proměnná nebo array) ?
        If (($vlTypDat=Is Picture) | ($vlTypDat=Picture Array))
                ` Pokud ano, přijmou tažení.
                ` Nezapomeňte že tlačítko myši je stisknuto, pouze na efekt
                ` přijetí tažení 4D zvýrazní objekt aby uživatel
                ` věděl že zdrojová data mohou být vsazena
            $0:=0
        Else
                ` Pokud ne odmítnout
            $0:=-1
                ` V tomto případě není objekt zvýrazněn
        End if
    ÷ (Form event=On Drop)
            ` Zdrojová data byla vsazena do objektu, proto je potřebujeme kopírovat
            ` Vzít informace o zdrojovém objektu
        DRAG AND DROP PROPERTIES ($vpZdrObjekt;$vlZdrPrvek; $lZdrProcess)
        $vlTypDat:=Type ($vpZdrObjekt→)
        Case of
                ` Zdrojový objekt je obrázkové pole nebo proměnná
            ÷ ($vlTypDat=Is Picture)
                    ` Je zdrojový objekt ze stejného procesu?
                If ($lZdrProcess=Current process)
                        ` Pokud ano, pouze kopírovat zdrojovou hodnotu
                    [aTabulka]aObrazek:=$vpZdrObjekt→
                Else
                        ` Pokud ne, je zdrojový objekt proměnná?
                    If (Is a variable ($vpZdrObjekt))
                            ` pokud ano, vzít její hodnotu ze zdrojového procesu
```

```
                    GET PROCESS VARIABLE ($lZdrProcess;$vpZdrObjekt→;$vgDraggedPict)
                    [aTabulka]aObrazek:=$vgDraggedPict
                Else
                    ` Pokud ne, použijte CALL PROCESS k získání hodnoty
                    ` pole ze zdrojového procesu
                End if
            End if
            ` zdrojový objekt je array obrázků
        ÷ ($vlTypDat=Picture Array)
            ` Je zdrojový objekt ze stejného procesu?
            If ($lZdrProcess=Current process)
                ` Pokud ano, pouze zkopírovat hodnotu
                [aTabulka]aObrazek:=$vpZdrObjekt→{$vlZdrPrvek}
            Else
                ` Pokud ne, získat hodnotu ze zdrojového procesu
                GET PROCESS VARIABLE ($lZdrProcess;
                $vpZdrObjekt → {$vlZdrPrvek}; $vgDraggedPict)
                [aTabulka]aObrazek:=$vgDraggedPict
            End if
        End case
    End case
```

Poznámka: Pro další příklady na události tažení a vsazení si přečtěte příklady u příkazu DRAG AND DROP PROPERTIES.

5. Tento příklad je vzor pro metodu formuláře. Ukazuje každou z možných událostí která se může stát pokud určitá souhrnná zpráva používá formulář jako výstupní formulář:

```
` Metoda formuláře použitá jako výstupní formulář pro souhrnnou zprávu
$vpFormTabulky:=Current form table
Case of
    ` ...
  ÷ (Form event=On Header)
        ` Bude se tisknout oblast záhlaví
    Case of
      ÷ (Before selection($vpFormTabulky→))
            ` kód pro první zlom záhlaví proběhne zde
      ÷ (Level = 1)
            ` Kód pro zlom záhlaví úrovně 1 proběhne zde
      ÷ (Level = 2)
            ` Kód pro zlom záhlaví úrovně 2 proběhne zde
            ` ...
    End case
  ÷ (Form event=On Printing Details)
        ` Bude se tisknout záznam
        ` Kód pro každý záznam proběhne zde
  ÷ (Form event=On Printing Break)
        ` Oblast zlomu se bude tisknout
    Case of
      ÷ (Level = 0)
            ` Kód pro zlom úrovně 0 proběhne zde
      ÷ (Level = 1)
            ` Kód pro zlom úrovně 1 proběhne zde
```

```
                ` ...
        End case
    ÷ (Form event=On Printing Footer)
        If(End selection($vpFormTabulky→))
                ` Kód pro poslední zápatí proběhne zde
        Else
                ` Kód pro zápatí proběhne zde
        End if
End case
```

6. Tento příklad je vzor metody formuláře, která pracuje s událostmi, které mohou nastat při zobrazení formuláře s použitím příkazu DISPLAY SELECTION nebo MODIFY SELECTION. Ze cvičných důvodů zobrazí podstatu události v titulku okna formuláře.

```
    ` Metoda formuláře
Case of
    ÷ (Form event=On Load)
        $vsTheEvent:="Formulář je zobrazen"
    ÷ (Form event=On Unload)
        $vsTheEvent:="Výstupní formulář je opuštěn a mizí z obrazovky"
    ÷ (Form event=On Display Details)
        $vsTheEvent:="Zobrazení záznamu #"+String(Selected record number([TheTable]))
    ÷ (Form event=On Menu Selected)
        $vsTheEvent:="Byla vybrána položka nabídky"
    ÷ (Form event=On Header")
        $vsTheEvent:="Vykreslí se oblast záhlaví"
    ÷ (Form event=On Clicked")
        $vsTheEvent:="Na záznam bylo klepnuto"
    ÷ (Form event=On Double Clicked")
        $vsTheEvent:="Na záznam bylo poklepnuto"
    ÷ (Form event=On Open Details)
        $vsTheEvent:="Na záznam #"+String(Selected record number([TheTable]))+"bylo poklepáno"
    ÷ (Form event=On Close Details)
        $vsTheEvent:="Vrací se zpět do výstupního formuláře"
    ÷ (Form event=On Activate)
        $vsTheEvent:="Okno formuláře se stalo oknem na popředí"
    ÷ (Form event=On Deactivate)
        $vsTheEvent:="Okno formuláře již není okno na popředí"
    ÷ (Form event=On Outside cal)
        $vsTheEvent:="Bylo zachyceno volání odjinud"
Else
    $vsTheEvent:="Co se děje? Událost #"+String(Form event)
End case
SET WINDOW TITLE ($vsTheEvent)
```

7. Příklady použití událostí Před úhozem na klávesu a Po úhozu na klávesu si přečtěte příklady u příkazů Get edited text, Keystroke a FILTER KEYSTROKE.

8. Tento příklad ukazuje jak zacházet v posuvné oblastí s klepnutím a poklepáním stejným způsobem:

```
    ` Metoda objektu asVyber posuvná oblast
Case of
    ÷ (Form event=On Load)
        ARRAY STRING (...;asVyber;...)
```

```
            ` ...
        asVyber:=0
    ÷ ((Form event=On Clicked) | (Form event=On Double Clicked))
        If (asVyber #0)
            ` Bylo klepnuto na položku, zde něco udělejte
            ` ...
        End if
            ` ...
End case
```

9. Tento příklad ukazuje jak zacházet s klepnutím a poklepáním zvlášť. Všiměte si použití prvku nula pro zacházení s označeným prvkem.

```
    ` Metoda objektu asVyber posuvná oblast
Case of
    ÷ (Form event=On Load)
        ARRAY STRING (...;asVyber;...)
            ` ...
        asVyber:=0
        asVyber{0}:="0"
    ÷ (Form event=On Clicked)
        If (asVyber #0)
            If (asVyber # Num( asVyber))
                    ` Bylo klepnuto na novou položku, zde něco udělejte
                    ` ...
                    ` Uložit novou označenou položku pro příště
                asVyber{0}:=String (asVyber)
            End if
        Else
            asVyber:=Num(asVyber{0})
        End if
    ÷ (Form event=On Double Clicked)
        If (asVyber#0)
            ` Na položku bylo poklepáno, udělejte zde něco jiného
        End if
            ` ...
End case
```

10. Tento příklad ukazuje jak udržovat stav textové oblasti pomocí metody formuláře a událostí Při získání fokusu a Při ztrátě fokusu:

```
    ` Metoda formuláře [Kontakty];"VstupDat"
Case of
    ÷ (Form Event=On Load)
        C_TEXT(vtStavovaOblast)
        vtStavovaOblast:=""
    ÷ (Form Event=On Getting Focus)
        RESOLVE POINTER (Last object;$vsPromNazev;$vlTabulkaNum;$vlPoleNum)
        If (($vlTabulkaNum#0) & ($vlPoleNum#0))
            Case of
                ÷ ($vlPoleNum=1) ` Pole příjmení
                    vtStavovaOblast:="Napište příjmení kontaktu, bude automatiky zvětšen"
                        ` ...
```

```
            ÷ ($vlPoleNum=10) ` Pole PSČ
                vtStavovaOblast:="Zadejte 5 znaků PSČ, bude automaticky vyzkoušen a potvrzen"
                    ` ...
            End case
        End if
    ÷ (Form Event=On Losing Focus)
        vtStavovaOblast:=""
            ` ...
End case
```

11. Tento příklad ukazuje jak odpovědět na událost zavření okna z formuláře pro vstup záznamů:

```
    ` Metoda pro formulář pro vstup dat
$vpFormTabulky:=Current form table
Case of
        ` ...
    ÷ (Form Event=On Close Box)
        If (Modified record($vpFormTabulky→))
            CONFIRM ("Tento záznam byl změněn. Uložit změny?")
            If (OK=1)
                ACCEPT
            Else
                CANCEL
            End if
        Else
            CANCEL
        End if
            ` ...
End case
```

12. Tento příklad ukazuje jak zvětšit písmena v textovém nebo alfanumerickém poli pokaždé, když se jejich data změní:

```
        ` Metoda objektu [Kontakty]Jméno
Case of
        ` ...
    ÷ (Form event=On Data Change)
        [Kontakty]Jméno:= Uppercase(Substring([Kontakty]Jméno;1;1))
                          +Lowercase(Substring([Kontakty]Jméno;2))
            ` ...
End case
```

Dále si přečtěte

CALL PROCESS, Current form table, DRAG AND DROP PROPERTIES, FILTER KEYSTROKE, Get edited text, Keystroke, SET TIMER.

Příkazy a odkazy pro Události formuláře

4th Dimension 6.5, Seznam témat příkazů

Before (Před)

Příkazy a odkazy pro Události formuláře

Verze 3

Poznámka kompatibility

Tento příkaz byl do 4D předán z důvodů kompatibility. Nově ve verzi 6 můžete použít příkaz Form event a zachytávat, jestli nevrátí událost Při aktualizaci (On Data Change).

Before → Logické

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Před tím než provedete spuštění Before cyklu, ujistěte se, že máte ve formuláři a v objektu zaškrtnutou událost Při zavedení.

Dále si přečtěte

Form event.

Příkazy a odkazy pro Události formuláře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

During (Během)

Příkazy a odkazy pro Události formuláře

Verze 3

Poznámka Kompatibility

Tento příkaz byl do 4D předán z důvodů kompatibility. Nově ve verzi 6 můžete použít příkaz Form event a zachytávat, jestli nevrátí událost jako třeba Při klepnutí (On Clicked).

During → Logické

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Před tím než provedete During cyklus, ujistěte se že máte označeny patřičné události, jako Při klepnutí, pro formulář a nebo objekt.

Dále si přečtěte

Form event.

Příkazy a odkazy pro Události formuláře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

After (Po)

Příkazy a odkazy pro Události formuláře

Verze 3

Poznámka Kompatibility

Tento příkaz byl do 4D předán z důvodů kompatibility. Nově ve verzi 6 můžete použít příkaz Form event a zachytávat, jestli nevrátí událost Při potvrzení vstupu (On Validate).

After → Logické

Parametr	Typ	Popis
Tento příkaz nevyžaduje žádné parametry.		

Popis

Před tím než provedete After cyklus, ujistěte se že máte označenu událost Při potvrzení vstupu pro formulář a nebo objekt.

Dále si přečtěte

Form event.

Příkazy a odkazy pro Události formuláře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

In header (V záhlaví)

Příkazy a odkazy pro Události formuláře

Verze 3

Poznámka Kompatibility

Tento příkaz byl do 4D předán z důvodů kompatibility. Nově ve verzi 6 můžete použít příkaz Form event a zachytávat, jestli nevrátí událost V záhlaví (On header).

In header → Logické

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Před tím než provedete In header cyklus, ujistěte se že máte označenu událost V záhlaví pro formulář a nebo objekt.

Dále si přečtěte

During, In break, In footer.

Příkazy a odkazy pro Události formuláře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

In break (Ve zlomu)

Příkazy a odkazy pro Události formuláře

Verze 3

Poznámka Kompatibility

Tento příkaz byl do 4D předán z důvodů kompatibility. Nově ve verzi 6 můžete použít příkaz Form event a zachytávat, jestli nevrátí událost Při tisku zlomu (On printing break).

In break → Logické

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Před tím než provedete In break cyklus, ujistěte se že máte označenu událost Při tisku zlomu pro formulář a nebo objekt.

Dále si přečtěte

During, In footer, In header.

Příkazy a odkazy pro Události formuláře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

In footer (V zápatí)

Příkazy a odkazy pro Události formuláře

Verze 3

Poznámka Kompatibility

Tento příkaz byl do 4D předán z důvodů kompatibility. Nově ve verzi 6 můžete použít příkaz Form event a zachytávat, jestli nevrátí událost Při tisku zápatí (On printing footer).

In footer → Logické

Parametr	Typ	Popis
Tento příkaz nevyžaduje žádné parametry.		

Popis

Před tím než provedete In footer cyklus, ujistěte se, že máte označenu událost Při tisku zápatí pro formulář a nebo objekt.

Dále si přečtěte

During, In break, In header.

Příkazy a odkazy pro Události formuláře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Activated (Aktivované)

Příkazy a odkazy pro Události formuláře

Verze 3

Poznámka Kompatibility

Tento příkaz byl do 4D předán z důvodů kompatibility. Nově ve verzi 6 můžete použít příkaz Form event a zachytávat, jestli nevrátí událost Při aktivaci (On Activate).

Activated → Logické

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Logické	←	Vrací TRUE pokud je prováděcí cyklus aktivace

Popis

Příkaz Activated vrací True do metody formuláře pokud okno které obsahuje formulář se stalo oknem na popředí v procesu na popředí.

UPOZORNĚNÍ: Nedávejte příkazy jako TRACE a ALERT do fáze formuláře Activated, nebo způsobíte nekonečnou smyčku.

Poznámka: Před tím než provedete Activated cyklus, ujistěte se že máte označenu událost Při aktivaci pro formulář a nebo objekt. Je to automaticky provedeno při převodu databáze.

Dále si přečtěte

Deactivated, Form event.

Příkazy a odkazy pro Události formuláře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Deactivated (Deaktivované)

Příkazy a odkazy pro Události formuláře

Verze 3

Poznámka Kompatibility

Tento příkaz byl do 4D předán z důvodů kompatibility. Nově ve verzi 6 můžete použít příkaz Form event a zachytávat, jestli nevrátí událost Při deaktivaci (On Deactivate).

Deactivated → Logické

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Logické	←	Vrací TRUE pokud je prováděcí cyklus deaktivace

Popis

Příkaz Deactivated vrací True do metody formuláře pokud okno na popředí v procesu na popředí které obsahuje formulář se přesunulo na pozadí.

Před tím než provedete Deactivated cyklus, ujistěte se že máte označenu událost Při deaktivaci pro formulář anebo objekt.

Dále si přečtěte

Activated, Form event.

Příkazy a odkazy pro Události formuláře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Outside call (Vnější volání)

Příkazy a odkazy pro Události formuláře

Verze 3

Poznámka Kompatibility

Tento příkaz byl do 4D předán z důvodů kompatibility. Nově ve verzi 6 můžete použít příkaz Form event a zachytávat, jestli nevrátí událost Při vnějším volání (On Outside call).

Outside call → Logické

Parametr	Typ	Popis
Tento příkaz nevyžaduje žádné parametry.		

Popis

Před tím než provedete Outside cyklus, ujistěte se že máte označenu událost Při vnějším volání pro formulář a nebo objekt.

Dále si přečtěte

CALL PROCESS, Form event.

Příkazy a odkazy pro Události formuláře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Get edited text (Získat upravovaný text)

Příkazy a odkazy pro Události formuláře

Verze 6.5

Get edited text → Text

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Text	←	Text který byl předán

Popis

Příkaz Get edited text je hlavně používán s událostí Po úhozu na klávesu k získání textu tak jak byl zadán. Může být také použit s událostí Před úhozem na klávesu. Jestli chcete vědět více informací o těchto událostech, přečtěte si popis u příkazu Form event.

Poznámka: Kvůli nové události Po úhozu na klávesu (nově ve verzi 6.5) byla stará událost Při úhozu na klávesu přejmenována na Před úhozem na klávesu.

Při použití s jiným vstupem než textovým objektem ve formuláři, vrátí tato funkce prázdný řetězec.

Příklady

1. Následující příkad převede právě předáné znaky na velká písmena:

```
If (Form event=On After Keystroke)
   [Trips]Agencies:=Uppercase(Get edited text)
End if
```

2. Zde je příklad jak zacházet s právě zadanými znaky. Účel je umístit právě zadaná slova do jiného textového pole (nazvaného "Slova"). Aby jste toho docílili, vložte následující metodu do metody objektu pro pole:

```
If (Form event=On After Keystroke)
   $SkutečnyVstup:=Get edited text
   PLATFORM PROPERTIES($platform)
   If ($platform#3) ` MacOS
      Repeat
         $RozloženaVěta:=Replace string($SkutečnyVstup; Char(32);Char(13))
      Until (Position(" ";$RozloženaVěta)=0)
   Else ` Windows
      Repeat
         $RozloženaVěta:= Replace string ($SkutečnyVstup;Char(32);Char(13)+Char(10))
      Until (Position(" ";$RozloženaVěta)=0)
   End if
   [Příklad]Slova:=$RozloženaVěta
End if
```

Poznámka: Tento příklad není úplný, protože jsme předstírali že slova jsou jednotlivě oddělena mezerami (Char(32)). Pro kompletní dokončení by jste potřebovali další filtry k získání slov (oddělených tečkou, středníkem, apostrofem, atd.).

Dále si přečtěte

Form event.

Příkazy a odkazy pro Události formuláře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET TIMER (NASTAVIT ČASOVAČ)

Příkazy a odkazy pro Události formuláře

Verze 6.5

SET TIMER (početTiků)

Parametr	Typ		Popis
PočetTiků	LongInt	→	Počet tiků

Popis

Příkaz SET TIMER vám umožní aktivovat událost Při časování, a pro platný proces nastavit počet tiků, které proběhnou mezi každou událostí Při časování, parametrem *početTiků*.

Poznámka: Jestli chcete vědět více informací o této nové události, přečtěte si popis k příkazu Form event.

Pokud je tento příkaz volán v místě kde nemá k dispozici formulář, neprovede nic.

Web server 4D při odesílání formulářů 4D může využít možnosti tohoto příkazu stejně jako událost Při časování. Tato možnost vám umožní získat HTML stránky obnovované v reálném čase při ukládání záznamu. Obnovení záznamu v tomto případě neprobíhá automaticky; musíte volat příkaz REDRAW. Můžete optimalizovat váš systém, aby volal REDRAW pouze v případě, že se změní data.

Pouze prohlížeče které umožňují JavaScript vám umožní automaticky překreslit stránky. Čas zadaný pomocí SET TIMEOUT bude použit prohlížečem a Web procesem. Zadaný čas by měl být několik sekund (5 je praktická hodnota). Pro další informace si přečtěte druhý příklad u tohoto příkazu.

Pro procedurální zrušení události Při časování, nastavte do parametru početTiků 0.

Příklady

1. Uvažujme že chcete aby počítač pípnul každé tři vteřiny po zobrazení formuláře na obrazovku. Aby jste to udělali, napište následující:

```
If (Form event=On Load)
   SET TIMER(60*3)
End if
If (Form event=On Timer)
   BEEP
End if
```

2. Uvažujme že chcete každých 5 sekund obnovit formulář 4D zobrazený Web prohlížečem. Aby jste to udělali, napište následující metodu formuláře:

```
If (Form event=On Load)
   SET TIMER(60*5)
End if
If (Form event=On Timer)
      ... ` Zde můžete umístit test, jestli byla data upravena a pokud ano,
      `spusťte následující řádek.
   REDRAW
End if
```

Dále si přečtěte

Form event, REDRAW.

Příkazy a odkazy pro Události formuláře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Stránky formuláře

Příkazy a odkazy pro Stránky formuláře

Verze 6.0

Příkazy v této části vám umožní pracovat se stránkami formuláře.

Automatické akce tlačítek provádějí stejné úkoly jako příkazy FIRST PAGE, LAST PAGE, NEXT PAGE a PREVIOUS PAGE. Ve verzi 6 je nově předána automatická akce, ekvivalentní příkazu GOTO PAGE, kterou můžete použít třeba na Ovládací karty, rozevírací seznamy, atd. Pokud to bude možné, používejte automatické akce místo těchto příkazů.

Příkazy stránek mohou být použity ve vstupním formuláři a ve formuláři zobrazeném jako dialog. Výstupní formulář používá pouze první stránku a proto zde nelze tyto příkazy použít. Formulář má vždy alespoň jednu stránku - první stránku. Nezapomeňte, že na rozdíl od většího počtu stránek, má každý formulář pouze jednu metodu formuláře.

K získání čísla právě zobrazené stránky, použijte příkaz Current form page.

Poznámka: Při **navrhování** formuláře můžete pracovat se stránkami 1 až N stejně jako se stránkou 0, na kterou umístíte objekty, které se budou objevovat na všech stránkách. Při **používání** formuláře můžete pracovat se stránkami 1 až N, ale stránka 0 je automaticky kombinována se všemi stránkami a nemůžete ji použít samotnou.

Příkazy a odkazy pro Stránky formuláře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

FIRST PAGE (PRVNÍ STRÁNKA)

Verze 3

FIRST PAGE

Parametr Typ Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz FIRST PAGE mění aktuální zobrazenou stránku formuláře na první stránku. Pokud není formulář zobrazen nebo je již zobrazena první stránka, příkaz FIRST PAGE neudělá nic.

Příklad

Následující příklad je jednořádková metoda volaná z položky nabídky. Zobrazí první stránku formuláře.

```
FIRST PAGE
```

Dále si přečtěte

Current form page, GOTO PAGE, LAST PAGE, NEXT PAGE, PREVIOUS PAGE.

Příkazy a odkazy pro Stránky formuláře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

LAST PAGE (POSLEDNÍ STRÁNKA)

Příkazy a odkazy pro Stránky formuláře

Verze 3

LAST PAGE

Parametr Typ Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz LAST PAGE mění aktuální zobrazenou stránku formuláře na poslední stránku. Pokud není formulář zobrazen nebo je již zobrazena poslední stránka, příkaz LAST PAGE neudělá nic.

Příklad

Následující příklad je jednořádková metoda volaná z položky nabídky. Zobrazí poslední stránku formuláře.

```
LAST PAGE
```

Dále si přečtěte

Current form page, GOTO PAGE, FIRST PAGE, NEXT PAGE, PREVIOUS PAGE.

Příkazy a odkazy pro Stránky formuláře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

NEXT PAGE (DALŠÍ STRÁNKA)

Verze 3

NEXT PAGE

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz NEXT PAGE mění aktuální zobrazenou stránku formuláře na následující stránku. Pokud není formulář zobrazen nebo je již zobrazena poslední stránka, příkaz NEXT PAGE neudělá nic.

Příklad

Následující příklad je jednořádková metoda volaná z položky nabídky. Zobrazí stránku formuláře která následuje za aktuálně zobrazenou.

```
NEXT PAGE
```

Dále si přečtěte

Current form page, GOTO PAGE, FIRST PAGE, LAST PAGE, PREVIOUS PAGE.

Příkazy a odkazy pro Stránky formuláře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

PREVIOUS PAGE (PŘEDCHOZÍ STRÁNKA)

Verze 3

PREVIOUS PAGE

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz PREVIOUS PAGE mění aktuální zobrazenou stránku formuláře na předchozí stránku. Pokud není formulář zobrazen nebo je již zobrazena první stránka, příkaz PREVIOUS PAGE neudělá nic.

Příklad

Následující příklad je jednořádková metoda volaná z položky nabídky. Zobrazí stránku formuláře která je před aktuálně zobrazenou.

```
PREVIOUS PAGE
```

Dále si přečtěte

Current form page, GOTO PAGE, LAST PAGE, NEXT PAGE, FIRST PAGE.

Příkazy a odkazy pro Stránky formuláře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GOTO PAGE (JÍT NA STRÁNKU)

Příkazy a odkazy pro Stránky formuláře

Verze 3

GOTO PAGE (číslo stránky)

Parametr	Typ		Popis
Číslo stránky	Číslo	→	Číslo stránky formuláře která se má zobrazit

Popis

GOTO PAGE mění aktuální zobrazenou stránku formuláře na stránku zadanou do parametru *číslostránky*.

Pokud není formulář zobrazen, GOTO PAGE neudělá nic. Pokud je číslo stránky větší než počet stránek formuláře, je zobrazena poslední stránka a pokud je číslo stránky menší než počet stránek ve formuláři, je zobrazena první stránka.

Příklady

Následující příklad je příklad metody objektu tlačítka, které zobrazí specifickou stránku, stránku 3.

```
GOTO PAGE (3)
```

Dále si přečtěte

Current form page, FIRST PAGE, LAST PAGE, NEXT PAGE, PREVIOUS PAGE.

Příkazy a odkazy pro Stránky formuláře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Current form page (Platná stránka formuláře)

Příkazy a odkazy pro Stránky formuláře

Verze 3

Current form page → Číslo

Parametr Typ Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz Current form page vrací číslo aktuálně zobrazené stránky formuláře.

Příklad

Pokud ve formuláři vyberete položku nabídky nebo pokud formulář obdrží volání z jiného procesu, můžete provést různé akce podle aktuálně zobrazené stránky formuláře. V tomto příkladu napíšete:

```
` Metoda formuláře [MojeTabulka];"MůjFormulář"
Case of
   ÷ (Form event=On Load)
      ` ...
   ÷ (Form event=On Unload)
      ` ...
   ÷ (Form event=On Menu selected)
      $vlMenuNumber:=Menu Selected >> 16
      $vlItemNumber:=Menu Selected & 0xFFFF
      Case of
         ÷ ($vlMenuNumber=...)
            Case of
               ÷ ($vlItemNumber=...)
               ÷ (Current form page=1)
                  ` Provést akci pro stránku 1
               ÷ (Current form page=2)
                  ` Provést akci pro stránku 2
                     ` ...
               ÷ ($vlItemNumber=...)
                     ` ...
            End case
         ÷ ($vlMenuNumber=...)
            ` ...
      End case
   ÷ (Form event=On Outside call)
      Case of
         ÷ (Current form page=1)
            ` Provést akci pro stránku 1
         ÷ (Current form page=2)
            ` Provést akci pro stránku 2
      End case
            ` ...
End case
```

Dále si přečtěte

FIRST PAGE, GOTO PAGE, LAST PAGE, NEXT PAGE, PREVIOUS PAGE.

Příkazy a odkazy pro Stránky formuláře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GRAPH (DIAGRAM)

Příkazy a odkazy pro Diagram

Verze 6.0 (Upraveno)

Poznámka verze 6: Od verze6 jsou diagramy vytvářeny plug-inem 4D Chart, který je předány do 4th Dimension. Příkaz Graph z předchozí verze je přesměrován ke 4D Chart. K použití dodatečných příkazů 4D Chart pro upravení oblasti diagramu umístěné ve formuláři, použijte parametr *obastgraf* (popsaný v této části), jako odkaz externí oblasti k příkazům 4D Chart. Jestli chcete vědět více informací o příkazech 4D Chart si přečtěte *Příručku 4D Chart*.

Příkaz GRAPH je navržen pro použití s oblastí digramu ve formuláři. Musí být použit v metodě formuláře nebo v metodě objektu jednoho z objektů ve formuláři. Může být také použit v metodě volané jednou z těchto metod.

GRAPH (oblastdiagram; diagramčíslo; xpopis; yprvky {; yprvky2; ...; yprvkyN})

Parametr	Typ		Popis
oblastdiagram	proměnná	→	oblast diagramu ve formuláři
diagramčíslo	číslo	→	číslo typu diagramu
xpopis	array | podpole	→	popisy na ose x
yprvky	array | podpole	→	data k vykreslení na osu y - maximálně osm kategorií

Popis

Příkaz GRAPH vykreslí diagram pro oblast diagramu ve formuláři. Data mohou být z array nebo podpolí.

Parametr *oblastdiagram* je název oblasti diagramu ve kterém má být diagram zobrazen. Oblast diagramu je vytvořena ve formuláři editorem formulářů s použitím objektu typu diagram. Název diagramu je název předány jako název proměnné. Jestli chcete vědět více informací o vytváření diagramů přečtěte si *Příručku návrháře 4th Dimension.*

Parametr *diagramčíslo* definuje typ diagramu který bude vytvořen. Musí to být číslo mezi 1 a 8. Typy diagramů jsou popsány v příkadu 1. Jakmile byl diagram vytvořen, můžete změnit jeho typ změnou parametru *diagramčíslo* a opětovným zavoláním příkazu GRAPH.

Parametr *xpopis* definuje popis který bude použit pro osu x (spodní část diagramu). Tato data mohou být typu řetězec, datum čas nebo číslo. Měl by být stejný počet podzáznamů nebo prvků array v *xpopis* jako podzáznamů nebo počet prvků v každém parametru *yprvky*.

Data definovaná parametrem *yprvky* jsou data diagramu. Musí být číselného typu. Může být zobrazeno až 8 datových nastavení. Diagram typu koláč zobrazí pouze první *yprvky*.

Příklady

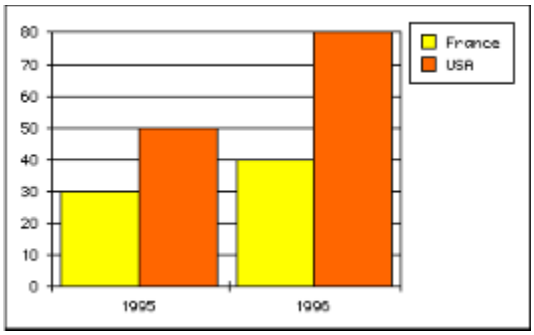
1. Následující příklad ukazuje jak použít array k vytvoření diagramu. Kód může být předán do metody formuláře nebo objektu. Není zamýšleno aby byl realistický, proto jsou data konstantní:

```
ARRAY STRING (4; X; 2) ` Vytvořit array pro osu x
X{1}:="1995" ` X popis #1
X{2}:="1996" ` X popis #2
ARRAY REAL (A; 2) ` Vytvořit array pro osu y
A{1}:=30 ` vložit nějaká data
A{2}:=40
ARRAY REAL (B; 2) ` Vytvořit array pro osu y
B{1}:=50 ` vložit nějaká data
B{2}:=80
GRAPH (vGraph;vType; X; A; B) ` vykreslit diagram
```

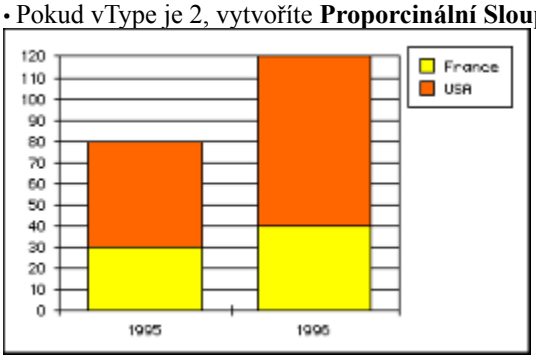
GRAPH SETTINGS (vGraph;0;0;0;0;**False**;**False**;**True**;"Francie";"USA")
` Nastavit legendu pro diagram

Následující obrázky ukazují výsledný diagram.

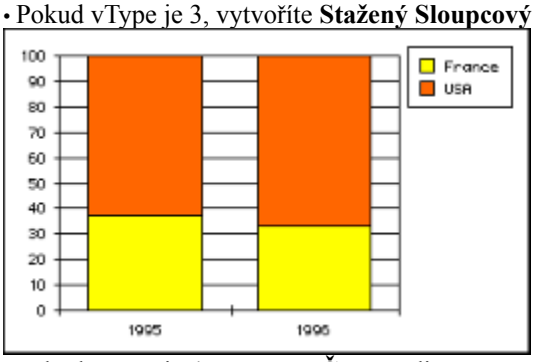
• Pokud vType je 1, vytvoříte **Sloupcový** diagram



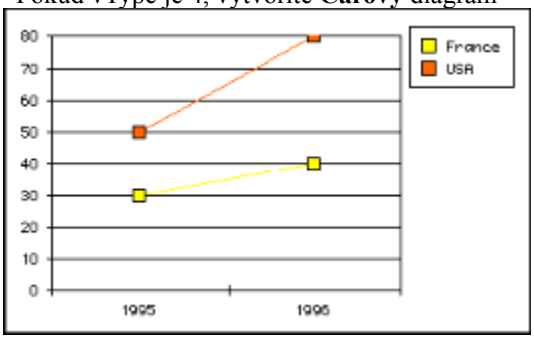
• Pokud vType je 2, vytvoříte **Proporcinální Sloupcový** diagram



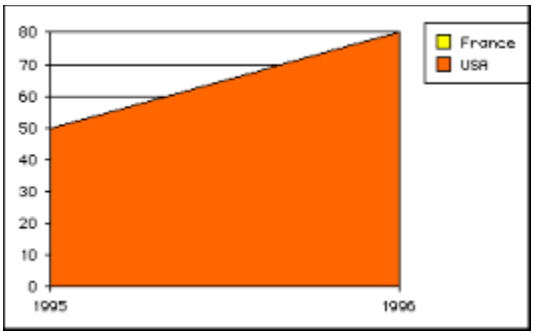
• Pokud vType je 3, vytvoříte **Stažený Sloupcový** diagram



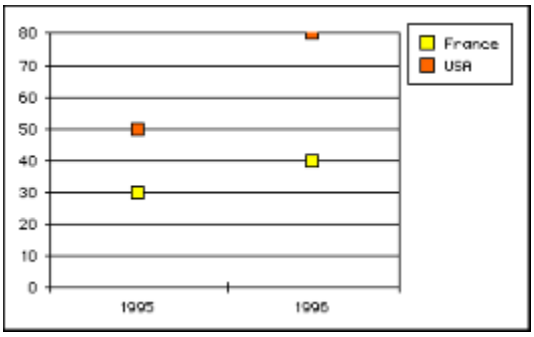
• Pokud vType je 4, vytvoříte **Čárový** diagram



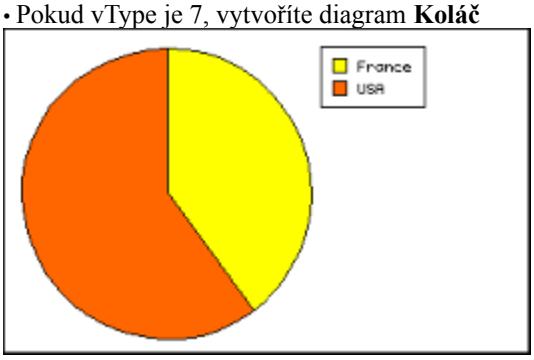
• Pokud vType je 5, vytvoříte diagram **Oblast**



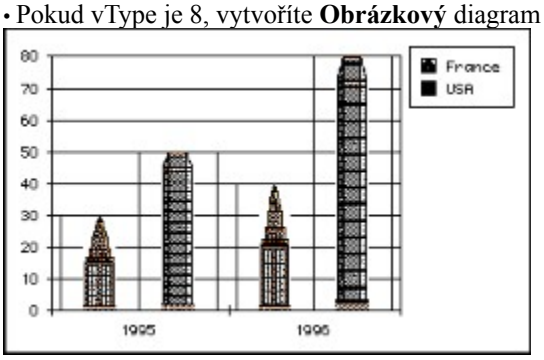
• Pokud vType je 6, vytvoříte **XY** diagram



• Pokud vType je 7, vytvoříte diagram **Koláč**



• Pokud vType je 8, vytvoříte **Obrázkový** diagram



2. Následující příklad vytvoří diagram prodejů v dolarech pro prodejce v podtabulce. Podtabulka má tři pole: Jméno, PosleníRokCelkem a TentoRokCelkem. Diagram zobrazí prodeje prodejců za poslední dva roky:

GRAPH (vDiagram;1;[Zaměstnanci]Prodeje`Jméno;[Zaměstnanci]Prodeje`PosledníRokCelkem;
[Zaměstnanci]Prodeje` TentoRokCelkem)

Dále si přečtěte

GRAPH SETTINGS, GRAPH TABLE.

Příkazy a odkazy pro Diagram

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GRAPH SETTINGS (NASTAVENÍ DIAGRAMU)

Příkazy a odkazy pro Diagram

Verze 6.0 (Upraveno)

Poznámka verze 6: Od verze6 jsou diagramy vytvářeny plug-inem 4D Chart, který je předáný do 4th Dimension. Příkaz GRAPH z předchozí verze je přesměrován ke 4D Chart. K použití dodatečných příkazů 4D Chart pro upravení oblasti diagramu umístěného ve formuláři, použijte parametr *oblastdiagram* (popsaný v předchozí části) jako odkaz externí oblasti k příkazům 4D Chart. Jestli chcete vědět více informací o příkazech 4D Chart přečtěte si *Příručku 4D Chart*.

Příkaz GRAPH je navržen pro použití s oblastí Diagram ve formuláři. Musí být použit v metodě formuláře nebo v metodě objektu jednoho z objektů ve formuláři. Může být také použit v metodě volané jednou z těchto metod.

GRAPH SETTINGS (diagram; xmin; xmax; ymin; ymax; xprop; xsíť; ysíť; titul {; titul2; ...; titulN})

Parametr	Typ		Popis
diagram	Proměnná	→	Název oblasti diagramu
xmin	číslo | datum | čas	→	Minimální hodnota x-osy pro proporcionální diagram (pouze čáry nebo dělený)
xmax	číslo | datum | čas	→	Maximální hodnota x-osy pro proporcionální diagram (pouze čáry nebo dělený)
ymin	číslo	→	Minimální hodnota y-osy
ymax	číslo	→	Maximální hodnota y-osy
xprop	logické	→	TRUE pro proporcionální x-osu; FALSE pro normální x-osu (čáry či dělené)
xsíť	logické	→	TRUE pro síť na x-osu; FALSE nezobrazení sítě (pouze když xprop je TRUE)
ysíť	logické	→	TRUE pro síť na y-osu; FALSE nezobrazení sítě
titul	řetězec	→	popisky pro legendy diagramu

Popis

Příkaz GRAPH SETTINGS mění nastavení diagramu zobrazeného ve formuláři. Diagram musí být vytvořen pomocí příkazu GRAPH. Příkaz GRAPH SETTINGS nemá žádný vliv na diagram typu Koláč.

Parametry *xmin*, *xmax*, *ymin* a *ymax* nastavují minimální a maximální hodnoty pro odpovídající osu v diagramu. Pokud některá dvojice těchto parametrů je nulová (0, ?00:00:00?, nebo !00/00/00!, závisí na typu dat), jsou použity výchozí hodoty.

Parametr *xprop* zapíná proporcionální kreslení pro čárová diagram (typ 4) a pro diagram XY (typ 6). Pokud je hodnota TRUE, vykreslí každý bod na ose x podle jeho hodnoty, pouze pokud jsou hodnoty číselné, čas nebo datum.

Parametr *xsíť* a *ysíť* mohou zobrazit nebo skrýt čáry sítě. Síť pro osu x bude zobrazena pouze když se kreslí diagram proporcionální čárový nebo XY.

Parametr(y) *titul* jsou popisky k legendě.

Poznámka kompatibility (Březen 97): Parametry xmin, xmax a xprop nejsou aktuálně podporovány. Budou podporovány v další verzi 4D Chart.

Příklad

Podívejte se na příklad u příkazu GRAPH.

Dále si přečtěte

GRAPH, GRAPH TABLE.

Příkazy a odkazy pro Diagram

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GRAPH TABLE (VYTVOŘIT DIAGRAM Z TABULKY)

Příkazy a odkazy pro Diagram

Verze 6.0 (Upraveno)

Poznámka verze 6: Od verze 6 jsou diagramy vytvářeny plug-inem 4D Chart, který je předány do 4th Dimension. Příkaz GRAPH TABLE z předchozí verze je přesměrován ke 4D Chart. Jestli chcete vědět více informací o příkazech 4D Chart přečtěte si *Příručku 4D Chart*.

GRAPH TABLE {(tabulka)}

nebo:

GRAPH TABLE ({tabulka;} diagramTyp; x pole; y pole {; y pole2; ...; y poleN})

Parametr	Typ		Popis
tabulka	tabulka	→	Tabulka, pro kterou vytvořit diagram Výchozí tabulka, není-li uvedeno nic
diagramtyp	číslo	→	číslo typu diagramu
x pole	pole	→	pole označení osy x
y pole	pole	→	pole hodnot na ose y - maximálně osm

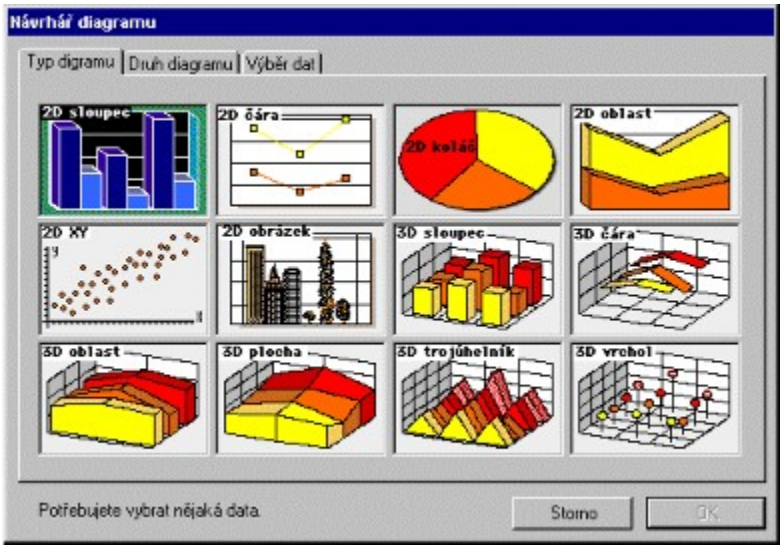
Popis

Příkaz GRAPH TABLE má dva způsoby použití. První zobrazí okno 4D Chart a umožní uživateli vytvořit vlastní diagram z vlastních polí. Druhý způsob určuje pole k zobrazení a nebude zobrazovat okno 4D Chart.

GRAPH TABLE vytvoří diagram z polí tabulky. Jsou zobrazena pouze data z platného výběru platného procesu.

Použití prvního způsobu je shodné s vybráním položky Diagram z nabídky Zprávy v prostředí uživatele.

Následující obrázek ukazuje okno 4D Chart,kde může uživatel definovat vlastní diagram.



Druhý způsob vytvoří diagram z polí definované tabulky *tabulka*.

Parametr *diagramtyp* definuje diagram, který bude zobrazen. Musí to být číslo mezi 1 a 8- Podívejte se na seznam typů diagramů v předchozím příkazu.

Parametr *x pole* definuje popisky které budou použity pro osu x (spodní část diagramu). Pole může být typu Alfa, Integer, Long Integer, Real nebo Datum.

Parametry *y pole* jsou data k zobrazení. Typ pole musí být Integer, Long Integer nebo Real. Může být zobrazeno až 8 y polí, každé oddělené střeníkem.

V obou případech otevře GRAPH TABLE okno diagramu pro práci s právě vytvořenými diagramy. Jestli chcete vědět více informací o okně diagramu, přečtě si *Příručku uživatele 4th Dimension.*

Poznámka: K vytvoření diagramu z polí můžete také použít i Rychlé zprávy.

Příklady

1. Následující příklad ukazuje použití prvního způsobu GRAPH TABLE. Otevře okno pro vytvoření diagramu a umožní uživateli vytvořit vlastní diagram. Kód provede hledání v tabulce, setřídí seznam záznamů a pak zobrazí okno pro vytvoření diagramu:

```
QUERY ([Lidé])
If (OK=1)
   ORDER BY ([Lidé])
   If (OK=1)
      GRAPH TABLE([Lidé])
   End if
End if
```

2.Následující příklad ukazuje druhý způsob použití GRAPH TABLE. Nejdříve nalezne a setřídí záznamy a pak vytvoří diagram platů pro lidi:

```
QUERY([Lidé];[Lidé]Funkce="Manager")
ORDER BY([Lidé];[Lidé]Plat;>)
GRAPH TABLE([Lidé];1;[Lidé]Příjmení;[Lidé]Plat)
```

Dále si přečtěte

Graph.

Příkazy a odkazy pro Diagram

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Load list (Načíst seznam)

Příkazy a odkazy pro Hierarchické seznamy

Verze 6.0

Load list (NázevSeznamu) → ListRef

Parametr	Typ		Popis
NázevSeznamu	Řetězec	→	Název seznamu vytvořeného v prostředí návrháře Editorem seznamů
Výsledek funkce	ListRef	←	Číslo odkazu na nově vytvořený seznam

Popis

Příkaz Load list vytvoří nový hierarchický seznam, jehož obsah je zkopírován ze seznamu předaného parametrem *NázevSeznamu*. Vrátí číslo odkazu na nově vytvořený seznam.

Pokud seznam v parametru *NázevSeznamu* neexistuje, seznam není vytvořen a funkce vrátí 0 (nula).

Nezapomeňte, že seznam je kopie seznamu vytvořeného v Prostředí návrháře. Změny v novém seznamu nebudou mít žádný vliv na již existující seznam. A naopak nebudou mít změny v seznamu vytvořeném v prostředí návrháře žádný vliv na nový seznam vytvořený příkazem Load list.

Pokud provedete nějaké změny do nového seznamu a budete je chtít uložit, zavolejte příkaz SAVE LIST.

Nezapomeňte provést příkaz CLEAR LIST když skončíte práci s tímto seznamem a již jej nebudete potřebovat. Jinak bude tento seznam zůstávat v paměti dokud neskončíte program, nebo dokud nebude skončen proces ve kterém byl vytvořen.

Tip: Pokud přiřadíte nějaký seznam k objektu ve formuláři (hierarchický seznam, ovládací karta nebo hierarchická rozevírací nabídka) s použitím Výběrový seznam v okně Vlastnosti objektu v Editoru formulářů, nemusíte volat příkazy Load list a CLEAR LIST 4th Dimension sama seznam načte a vymaže.

Příklad

Vytvoříte databázi pro mezinárodní obchod a potřebujete přepínat jazyky při používání této databáze. Do formuláře předáte seznam nazvaný hlList který bude obsahovat standardní vlastnosti.V prostředí návrháře si připravíte různé seznamy jako "Std možnosti US" pro Anglickou verzi, "Std možnosti ČR" pro Českou verzi, "Std možnosti FR" pro Francouzskou verzi, atd. Dále si vytvoříte meziprocesní proměnnou <>gsSoučasnýJazyk s dvojmístným značením pro jednotlivé jazyky jako "ČR" pro Českou, "US" pro Anglickou, "FR" pro Francouzskou, atd. Aby jste si byli jisti, že se váš seznam bude objevovat vždy se správným jazykem, můžete napsat:

```
   ` Metoda objektu hlList Hierarchický seznam
Case of
   ÷ (Form event = On Load)
      C_LONGINT (hlList)
      hlList:=Load list("Std možnosti"+<>gsSoučasnýJazyk)
   ÷ (Form event = On Unload)
      CLEAR LIST(hlList;*)
End case
```

Dále si přečtěte

CLEAR LIST, SAVE LIST.

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SAVE LIST (ULOŽIT SEZNAM)

Verze 6.0

SAVE LIST (list; listNázev)

Parametr	Typ		Popis
list	ListRef	→	Číslo odkazu na seznam
listNázev	Řetězec	→	Název seznamu který se objeví v prostředí návrháře v Editoru seznamů

Popis

Příkaz SAVE LIST uloží seznam, jehož číslo odkazu předáte parametrem *list,* pod názvem *listNázev*.

Pokud existuje seznam s tímto názvem je jeho obsah nahrazen.

Dále si přečtěte

Load list.

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

New list (Nový seznam)

Příkazy a odkazy pro Hierarchické seznamy

Verze 6.0

New list → ListRef

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	ListRef	←	Číslo odkazu na seznam

Popis

Příkaz New list vytvoří nový, prázdný hierarchický seznam a navrátí jeho jedinečné číslo odkazu.

UPOZORNĚNÍ: hierarchické seznamy jsou uchovávány v paměti. Jakmile dokončíte práci se seznamem, je důležité uvolnit paměť, kterou zabírají příkazem CLEAR LIST.

Ještě několik jiných příkazů vám umožní vytvořit hierarchické seznamy:
• Copy list duplikuje existující seznam.
• Load list vytvoří seznam z Výběrového seznamu vytvořeného (ručně nebo programově) v Editoru seznamů v Prostředí návrháře.
• BLOB to list vytvoří seznam z obsahu BLOBu, do kterého byl seznam předtím uložen.

Po vytvoření seznamu pomocí New list můžete:
• Přidat položky do seznamu pomocí příkazů APPEND TO LIST nebo INSERT LIST ITEM.
• Vymazat položky ze seznamu s použitím příkazu DELETE LIST ITEM.

Příklad

Podívejte se příklad u příkazu APPEND TO LIST.

Dále si přečtěte

APPEND TO LIST, BLOB to list, CLEAR LIST, Copy list, DELETE LIST ITEM, INSERT LIST ITEM, Load list.

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Copy list (Kopírovat seznam)

Příkazy a odkazy pro Hierarchické seznamy

Verze 6.0

Copy list (list) → ListRef

Parametr	Typ		Popis
list	ListRef	→	Odkaz na seznam který se má kopírovat
Výsledek funkce	ListRef	←	Odkaz na kopii seznamu

Popis

Příkaz Copy list duplikuje seznam jehož odkaz jste zadali do parametru *list* a vrací odkaz na nově vytvořený seznam.

Po dokončení práce s novým seznam jej vymažte pomocí CLEAR LIST.

Dále si přečtěte

CLEAR LIST, Load list, New list.

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CLEAR LIST (ODSTRANIT SEZNAM)

Příkazy a odkazy pro Hierarchické seznamy

Verze 6.0

CLEAR LIST (listRef {; *})

Parametr	Typ		Popis
listRef	ListRef	→	Číslo odkazu na seznam
*		→	odstraní z paměti i všechny podseznamy, jsou-li je-li vynechána, podseznamy nejsou z paměti odstraněny

Popis

Příkaz CLEAR LIST odstraní hierarchický seznam jehož číslo odkazu jste zadali do parametru *ListRef*.

Většinou použijete volitelný parametr *, takže se vymažou i všechny podseznamy přiřazené k hierarchickému seznamu (pokud nějaké jsou).

Seznam přiřazený k objektu pomocí okna Vlastnosti objektu nemusíte odstraňovat. 4D sama načte a vymaže seznam. Ale pokaždé když načítáte, kopírujete nebo získáváte seznam z BLOBu nebo jej vytvářítе programově, proveďte pokud jste skončili práci se seznamem příkaz CLEAR LIST.

K ostranění pouze podseznamu přiřazeného k položce (jakékoli úrovně) seznamu (ten chceme zachovat) zobrazeného ve formuláři, postupujte následovně:

1. Proveďte příkaz GET LIST ITEM na položce rodičovského seznamu k získání čísla odkazu podseznamu.

2. Proveďte příkaz SET LIST ITEM v rodičovském záznamu k odřazení podseznamu od položky před jeho vymazáním.

3. Proveďte příkaz CLEAR LIST k vymazání podseznamu, jehož číslo odkazu jste získali pomocí GET LIST ITEM.

4. Proveďte příkaz REDRAW LIST na seznam ve formuláři k přepočítání a překreslení položek a podseznamů.

Příklady

1. V metodě vymazání, která maže všechny objekty a data, která již nepotřebujete (okno bylo již zavřeno a formulář opuštěn), můžete končit mazáním hierarchického seznamu, který i může být již vymazán, podle akce kterou uživatel provedl v předchozím. Použijte příkaz Is a list ke kontrole, zda je seznamu nutné mazat:

```
   ` Metoda vymazání
If (Is a list(hlList))
   CLEAR LIST(hlList;*)
End if
```

2. Podívejte se například u příkazu Load list.

3. Podívejte se na příklad u příkazu BLOB to list.

Dále si přečtěte

BLOB to list, Load list, New list.

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Count list items (Počet položek seznamu)

Příkazy a odkazy pro Hierarchické seznamy

Verze 6.0

Count list items (list) → Long Integer

Parametr	Typ		Popis
list	ListRef	→	Číslo odkazu na seznam
Výsledek funkce	LongInteger	←	počet prvků viditelných (rozšířených) v seznamu

Popis

Příkaz Count list items vrátí počet položek, které jsou "viditelné" v seznamu, jehož číslo odkazu jste vložili do parametru *list*.

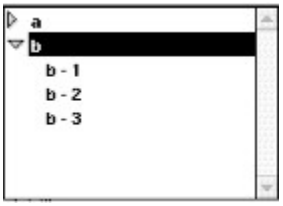
Count list items navrací celkový počet položek v seznamu. Vrací počet položek které jsou viditelné, což záleží na tom jestli jsou nějaké položky nebo podseznamy rozbalené či zabalené.

Tento příkaz se používá na seznamy zobrazené ve formuláři.

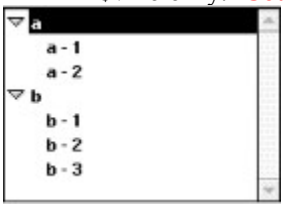
Příklady

1. Zde je seznam, s názvem hList, zobrazen v Prostředí uživatele.

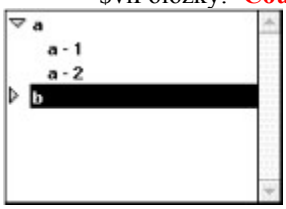
$vlPoložky:=**Count list items** (hList) ` v tomto okamžiku je $vlPoložky 2



$vlPoložky:=**Count list items** (hList) ` v tomto okamžiku je $vlPoložky 5



$vlPoložky:=**Count list items** (hList) ` v tomto okamžiku je $vlPoložky 7



$vlPoložky:=**Count list items** (hList) ` v tomto okamžiku je $vlPoložky 4

Dále si přečtěte

List item position, Selected list item.

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Is a list (Je seznamem)

Příkazy a odkazy pro Hierarchické seznamy

Verze 6.0

Is a list (list) → Logické

Parametr	Typ		Popis
list	Číslo	→	ListRef odkaz na seznam k testování
Výsledek funkce	Logické	←	TRUE pokud seznam je hierarchický seznam FALSE pokud seznam není hierarchický seznam

Popis

Příkaz Is a list vrací TRUE ,pokud je hodnota, kterou jste zadali do parametru list platným odkazem na hierarchický seznam, jinak vrací FALSE.

Příklady

1. Podívejte se na příklad u příkazu CLEAR LIST.

2. Podívejte se na příklad u příkazu DRAG AND DROP PROPERTIES.

Dále si přečtěte

DRAG AND DROP PROPERTIES.

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

REDRAW LIST (PŘEKRESLIT SEZNAM)

Verze 6.0

REDRAW LIST (list)

Parametr	Typ		Popis
list	Proměnná	→	Číslo odkazu na seznam

Popis

Příkaz REDRAW LIST přepočítá pozici všech položek a podseznamů (pokud nějaké jsou) pro seznam, jehož odkaz jste zadali do parametru list.

Tento příkaz MUSÍTE provést alespoň jednou, když změníte některé části seznamu, nebo podseznamy ve formuláři.

Upozornění: Jako parametr vkládejte název seznamu, ne výraz nebo proměnnou. Například, pokud máte ve formuláři seznam hList:

```
   ` Přepočítat seznam po změnách které jsme provedli
REDRAW LIST (hList) ` DOBŘE
   `....
$hList:=hList
   ` ...
   ` Přepočítat seznam po změnách které jsme provedli
REDRAW LIST ($hList) ` ŠPATNĚ
   ` ...
```

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET LIST PROPERTIES (NASTAVIT VLASTNOSTI SEZNAMU)

Příkazy a odkazy pro Hierarchické seznamy

Verze 6.0

SET LIST PROPERTIES (list; vzhled {; ikona {; výškařád}})

Parametr	Typ		Popis
list	číslo	→	číslo odkazu (ListRef) na seznam
vzhled	číslo	→	Grafický vzhled seznamu 1 Hierarchický seznam na Macintoshi 2 Hierarchický seznam na Windows
ikona	číslo	→	ID zdroje "cicn", nebo 0 pro výchozí označení dle platformy
výškařád	číslo	→	minimální výška řádky v bodech

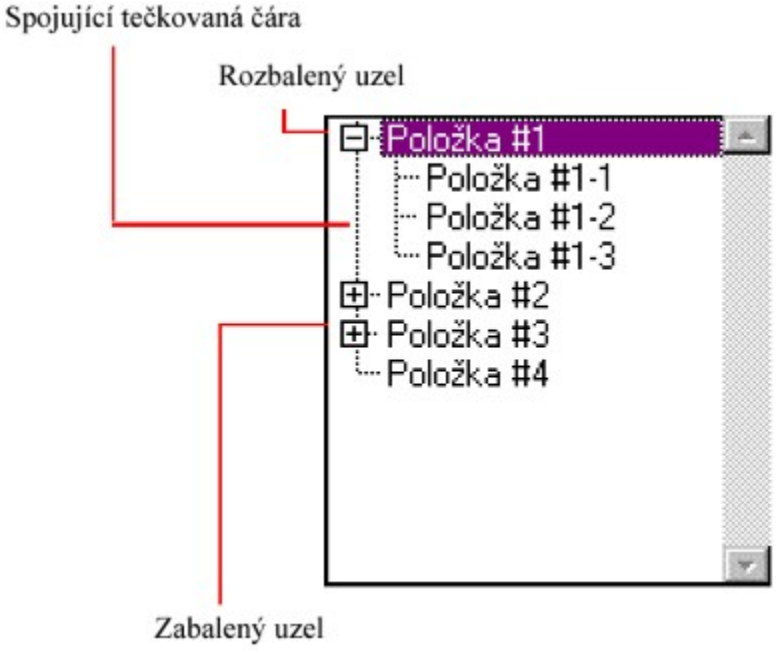
Popis

Příkaz SET LIST PROPERTIES nastaví vzhled hierarchického seznamu,jehož odkaz předáte do parametru list.

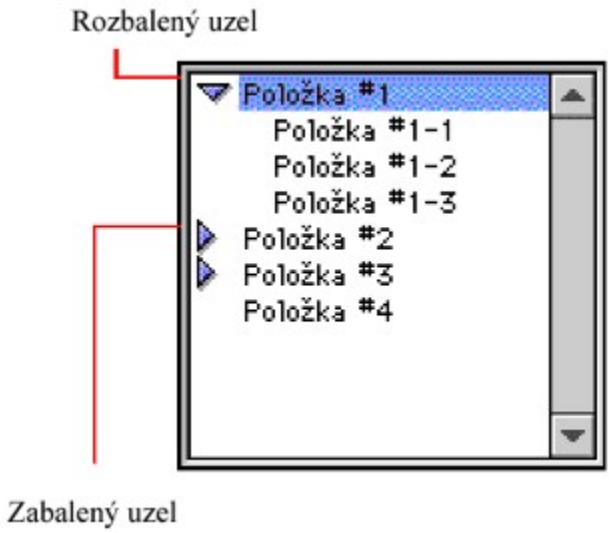
Parametr vzhled může být jedna z předdefinovaných konstant obsažených ve 4th Dimension:

Konstanta	*Typ*	*Hodnota*
ala Macintosh	*Long Integer*	*1*
ala Windows	*Long Integer*	*2*

Při vzhledu Windows má seznam připojené tečkované čáry mezi "uzlem" a potomkem. Jedna ikona označuje zabalený uzel a druhá označuje rozbalený uzel. Zde je výchozí vzhled Windows



Ve vzhledu Macintosh nemá seznam žádné spojovací čáry. Jedna ikona označuje zabalený uzel a druhá označuje rozbalený uzel. Uzly položek bez potomků nejsou zobrazeny vůbec. Zde je výchozí vzhled Macintosh:



Poznámka: Pokud zobrazíte hierarchický seznam bez nastavení předvoleb pomocí SET LIST PROPERTIES, je vzhled seznamu řízen podle nastavení vzhledu platformy ve Vlastnostech objektu.
Parametr *ikona* definuje vzhled ikony pro uzel. Hodnota předáná do parametru *ikona* nastavuje ikonu pro zabalený uzel, *ikona+1* nastavuje ikonu pro rozbalený uzel a *ikona+2* nastavuje ikonu pro uzel bez potomků (pokud je vzhled nastaven na Windows).

Například pokud předáte 15000, barevná ikona cicn' ID=15000 bude zobrazena pro každý zabalený uzel, barevná ikona cicn' ID=15001 bude zobrazena pro každý rozbalený uzel a barevná ikona cicn' ID=15002 bude zobrazena pro každý uzel bez potomků.

Je důležité nejdříve vložit tyto barevné ikony cicn' do zdroje ve vaší struktuře. Pokud chybí zdroj barevné ikony, odpovídající uzly budou zobrazeny bez ikony. (Můžete tuto možnost využít pro zobrazení seznamu bez ikon.)

UPOZORNĚNÍ: Při vytváření ikon do cicn' zdroje, použijte ID vyšší než nebo rovno 15000. ID zdrojů pod 15000 jsou rezervovány pro 4th Dimension.

ID výchozích zdrojů uzlů pro Macintosh i Windows jsou přístupné jako následující předdefinované konstanty 4th Dimension:

Konstanta	***Typ***	***Hodnota***
Macintosh node	*Long Integer*	*860*
Windows node	*Long Integer*	*138*

Jinými slovy 4th Dimension obsahuje následující předdefinované cicn' zdroje:

Číslo ID	***Popis***
860	*Zabalený uzel Macintosh*
861	*Rozbalený uzel Macintosh*
138	*Zabalený uzel Windows*
139	*Rozbalený uzel Windows*
140	*Uzel bez potomků na Windows*

Pokud nepředáte parametr ikony, zobrazí se uzly podle původního nastavení a podle zvoleného vzhledu.

Barevné ikony mohou být různých velikostí. Můžete například vytvořit ikonu o velikosti 16x16 nebo 32x32.

Pokud nepředáte parametr *výškařád*, je výška řádku nastavena podle velikosti písma a podle písma použitého v objektu. Pokud použijete barevnou ikonu ,která je příliš vysoká nebo dlouhá, objeví se tato ikona oříznutá, nebo bude překryta tečkovanou čárou (pokud bude vzhled Windows) .Stejně jako textem nebo uzlem nad a nebo pod ní.

Vyberte velikost barevné ikony, písmo a velikost písma optimálně, nebo zadejte minimální výšku řádku hierarchického seznamu. Pokud hodnota,kterou zadáte, bude větší než současná výška řádku, bude výška řádku

upravena podle čísla, které zadáte.

Poznámka: Příkaz SET LIST PROPERTIES nastavuje vzhled uzlů pro celý seznam. Pokud chcete nastavit vzhled ikon uzlů pro jednotlivé položky zvlášť, použijte příkaz SET LIST ITEM PROPERTIES.

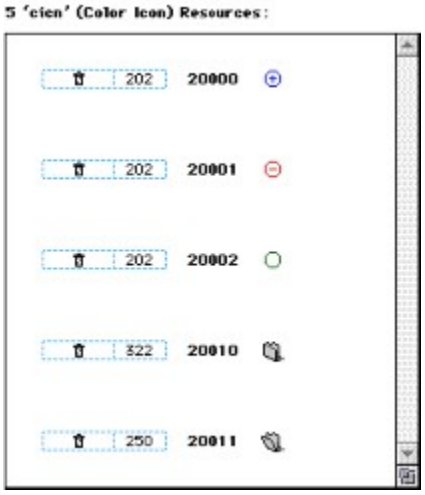
Příklady

Následující hierarchický seznam byl definován v Prostředí návrháře v Editoru seznamů:

Ve formuláři používá objekt hlMěsta tento seznam pmocí následující metody:

```
Case of
   ÷ (Form event=On Load)
      hlMěsta:=Load list("Města")
      SET LIST PROPERTIES(hlMěsta;vlVzhled;vlIcona)
   ÷ (Form event=On Unload)
      CLEAR LIST(hlMěsta;*)
End case
```

Dále byl změněn soubor struktury tak, že nyní obsahuje následující ikony:



1. S následujícím řádkem

```
SET LIST PROPERTIES (hlMěsta; Ala Macintosh; Macintosh node)
```

Hierarchický seznam bude vypadat takto:

2. S následujícím řádkem:

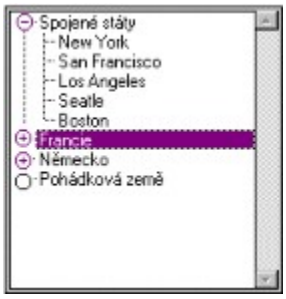
SET LIST PROPERTIES(hlMěsta;ala Windows;Windows node)

hierarchický seznam bude vypadat následovně:

3. S následujícím řádkem:

SET LIST PROPERTIES(hlMěsta;ala Windows;20000)

hierarchický seznam bude vypadat následovně:



4. S následujícím řádkem:

SET LIST PROPERTIES(hlMěsta;ala Macintosh;20000)

hierarchický seznam bude vypadat následovně:

5. S následujícím řádkem:

SET LIST PROPERTIES(hlMěsta;ala Macintosh;20010)

hierarchický seznam bude vypadat následovně:

Zdroje cicn' ikon ukázané a následně předáné do souboru struktury:

6. S následujícím řádkem:

SET LIST PROPERTIES(hlMěsta;ala Windows;20020;32)

hierarchický seznam bude vypadat následovně:

Dále si přečtěte

GET LIST ITEM PROPERTIES, GET LIST PROPERTIES, SET LIST ITEM PROPERTIES.

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

GET LIST PROPERTIES (ZÍSKAT VLASTNOSTI SEZNAMU)

Příkazy a odkazy pro Hierarchické seznamy

Verze 6.0

GET LIST PROPERTIES (list; vzhled {; ikona {; výškařád}})

Parametr	Typ		Popis
list	ListRef	→	Číslo odkazu na seznam
vzhled	Číslo	←	Grafický styl seznamu 1- Macintosh, 2 - Windows
ikona	Číslo	←	ID zdroje cicn' pro ikonu
výškařád	Číslo	←	Minimální výška řádku v bodech

Popis

Příkaz GET LIST PROPERTIES vrací informace o seznamu, jehož odkaz zadáte do parametru *list*.

Parametr *vzhled* vrací grafický styl seznamu.

Parametr *ikona* vrací ID zdroje, ze kterého se berou ikony zobrazené v seznamu.

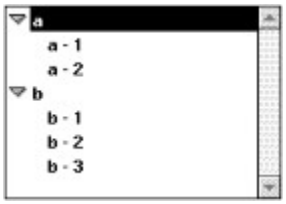
Parametr *výškařád* vrací minimální výšku řádku.

Tyto vlastnosti mohou být použity příkazem SET LIST PROPERTIES a nebo v prostředí návrháře v Editoru seznamů, jestliže byl seznam vytvořen, nebo uložen pomocí příkazu SAVE LIST.

Pro kompletní popis těchto parametrů se podívejte na příkaz SET LIST PROPERTIES.

Příklad

Existující seznam s názvem dList zobrazený v prostředí uživatele (ve vzhledu Macintosh):



Metoda objektu pro tlačítko:

```
` Metoda objektu bMacNeboWin tlačítko
GET LIST PROPERTIES(hList;$vlVzhled;$vlIcona;$vlLH)
If ($vlVzhled=Ala Macintosh)
   $vlVzhled:=Ala Windows
   $vlIcona:=Windows node
   $vlLH:=20
Else
   $vlVzhled:=Ala Macintosh
   $vlIcona:=Macintosh node
   $vlLH:=0
End if
SET LIST PROPERTIES(hList;$vlVzhled;$vlIcon;$vlLH)
REDRAW LIST(hList) ` NEZAPOMEŇTE provést REDRAW LIST jinak nebude seznam obnoven
```

alternativně zobrazí seznam ukázány výše(ve vzhledu Windows):

Dále si přečtěte

SET LIST PROPERTIES.

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SORT LIST (TŘÍDIT SEZNAM)

Příkazy a odkazy pro Hierarchické seznamy

Verze 6.0

SORT LIST (list {; > nebo <})

Parametr	Typ		Popis
list	List ref	→	Odkaz na seznam
> nebo <		→	Pořadí třídění: > vzestupné nebo < sestupné

Popis

Příkaz SORT LIST setřídí seznam, jehož odkaz předáte parametrem *list*.

K třídění ve vzestupném pořadí vložte >. K třídění v sestupném pořadí vložte <. Pokud je vynecháno, třídí se ve vzestupném pořadí.

Příkaz SORT LIST setřídí všechny úrovně seznamu. Nejdříve třídí položky seznamu,pak třídí položky podseznamů (pokud nějaké jsou), atd. přes všechny úrovně seznamu. Proto budete většinou používat SORT LIST na seznamy ve formáři. Třídění podseznamů je méně důležité, protože se pořadí mění při každém volání na vyšší úroveň.

Příkaz SORT LIST nemění označenou položku, ani stav rozbalení/zabalení položek. Nicméně, protože třídění může přesunout označenou položku, může Selected list item vrátit jiné číslo pozice před a po třídění.

Příklad

Existující seznam s názvem dList zobrazený v prostředí uživatele (ve vzhledu Macintosh):

Po spuštění této metody:

```
` třídit seznam a jeho podseznamy ve vzestupném pořadí
SORT LIST(hList;>)
` NEZAPOMEŇTE provést REDRAW LIST jinak nebude seznam obnoven
REDRAW LIST(hList)
```

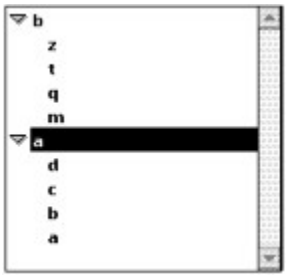
bude vypadat takto:

Po spuštění této metody:

```
` třídit seznam a jeho podseznamy v sestupném pořadí
```

SORT LIST(hList;>)
` NEZAPOMEŇTE provést REDRAW LIST jinak nebude seznam obnoven
REDRAW LIST(hList)

bude seznam vypadat takto:



Dále si přečtěte

Selected list item.

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

APPEND TO LIST (PŘIPOJIT K SEZNAMU)

Příkazy a odkazy pro Hierarchické seznamy

Verze 6.0

APPEND TO LIST (list; PoložkaText; PoložkaRef {; podlist {; rozšiř}})

Parametr	Typ		Popis
list	ListRef	→	Číslo odkazu seznamu
PoložkaText	Řetězec	→	Text další položky seznamu (max 31 znaků)
PoložkaRef	Číslo	→	Nové jedinečné číslo odkazu na položku seznam
podlist	ListRef	→	Nové číslo odkazu, připojení podseznamu k položce
rozšiř	Logické	→	Je-li podseznam připojen: TRUE = podseznam bude rozšířen FALSE = podseznam nebude rozšířen

Popis

Příkaz APPEND TO LIST připojí novou položku k hierarchickému seznamu, jehož číslo odkazu předáte parametrem *list*.

Text položky předáte do parametru *PoložkaText*. Můžete vložit řetězec, nebo textový výraz v délce maximálně 31 znaků. Pokud předáte delší hodnotu, bude oříznuta.

Jedinečnou hodnotu čísla odkazu předáte do parametru *PoložkaRef*. Ačkoli jsme definovali odkaz položce jako jedinečný, můžete vložit hodnotu ,jakou chcete. Přečtěte si následující část Číslo odkazu k položce.

Pokud chcete aby položka měla podseznam, vložte platné číslo odkazu potomka do parametru *podlist*. K rozbalení nebo zabalení poseznamu vložte TRUE nebo FALSE do parametru *rozšiř*.

Odkaz seznamu, který předáte do *podlist* ,musí být odkaz na existující seznam. Existující seznam může být prázdný, mít jednu úroveň, nebo mít podseznamy. Pokud nechcete přiřadit potomka k nové položce, vynechte tento parametr nebo vložte 0. Pokud přiřadíte parametr *podlist* a nedefinujete parametr *rozšiř*, podseznam jako výchozí nebude rozšířen. I když jsou oba volitelné, jak parametr *podlist* ,tak *rozšiř*, musí být předány současně.

Tipy

• K předání nové položky do seznamu, použijte příkaz INSERT LIST ITEM. K upravení existující položky nebo připojeného seznamu použijte příkaz SET LIST ITEM.
• K upravení vzhledu nově připojeného podseznamu použijte příkaz SET LIST ITEM PROPERTIES.

UPOZORNĚNÍ: Pokud přidáváte položku k seznamu, který je zobrazen ve formuláři,nebo k seznamu přiřazenému k položce (přes jeden nebo více úrovní), jejíž seznam je zobrazen, MUSÍTE použít příkaz REDRAW LIST; 4D přepočítá seznam a zobrazí změny, které jste provedli. Pravidlo je jednoduché: pokaždé, když budete měnit seznam, nebo položky na jakékoli úrovni, proveďte příkaz REDRAW LIST k zobrazení změn, které jste provedli.

Číslo odkazu k položce: Co s ním dělat?

Každá položka v hierarchickém seznamu má své Long Integer číslo odkazu. Je to číslo pro vaše použití: 4th Dimension jej pouze přenáší. Zde jsou nějaké možnosti co s tímto číslem můžete dělat:

1. Nepotřebujete jednotlivě identifikovat položky (úroveň začátečníka)

• První příklad: Programově vytvoříte Ovládací kartu například adresové knihy. Vzhledem k tomu, že Ovládací karta bude vracet číslo označené karty, obvykle nebudete potřebovat více informací. V tomto případě se nemusíte starat o čísla položek seznamu, vložte 0 do parametru PoložkaRef. Nezapomeňte, že pro Ovládací kartu knihy adres budete

chtít předdefinovat v prostředí návrháře seznamy A, B, ...Z. Nicméně je možná budete chtít vytvořit programově, aby jste zabránili vytvoření karet pro písmena ,pro která neexistují záznamy (například pro písmeno X pravděpodobně nebudete mít záznamy).

• Druhý příklad: Při vytváření databáze průběžně vytváříte seznam klíčových slov. Budete ukládat seznam po každém skončení práce pomocí příkazu SAVE LIST nebo LIST TO BLOB a znovu jej načístat na začátku práce s použitím příkazů Load list a BLOB to list. Zobrazíte si tento seznam v plovoucím okně. Jakmile klepnete na položku, vloží se klíčové slovo do platné dostupné oblasti ve formuláři v procesu na popředí. Můžete také použít tažení a vsazení. Důležité je, že pokaždé pracujete s označenou položkou (ta na kterou klepnete nebo jí přetáhnete), protože příkazy Selected list item (klepnutí) a DRAG AND DROP PROPERTIES, vám poskytnou pozici položky kterou vyberete. S použitím pozice můžete získat její text pomocí příkazu GET LIST ITEM. To je ono. Tak nepotřebujete jednotlivě rozlišovat položky podle jejich čísel; můžete do parametru položkaRef vložit 0.

2. Potřebujete částečnou identifikaci položek (střední úroveň)

Použijete čísla odkazu položek k uložení vyžadovaných informací, když pracujete s položkou; to je popsáno v dalším příkladu.V tomto příkladu použijeme čísla odkazu na položku pro uchovávání čísel záznamů. Nicméně musíme být schopni odlišit jednotlivé položky podle záznamů tabulky [Oddělení], které souhlasí se záznamy [Zaměstnanci]. Podívejte se na příklady k tomuto příkazu aby jste zjistili, jak na to.

3. Potřebujete jedinečně rozlišit položky seznamu (složitá úroveň).

Programujete složité ovládání hierarchického seznamu, kde potřebujete jedinečně odlišit každou položku seznamu na všech úrovních. Jednoduchý způsob, jak toho docílit, je vytvořit si vlastní čítač. Předpokládejme, že vytvoříte seznam hList s použitím příkazu New list. V tomto okamžiku nastavíte čítač vlhČítač na 0. Pokaždé, když provedete APPEND TO LIST , nebo INSERT LIST ITEM, zvětšíte hodnotu čítače (vlhČítač:=vlhČítač+1) a přiřadíte jeho hodnotu jako číslo odkazu na položku. Trik je v tom, že nebudete snižovat čítač při vymazání položek, čítač může pouze vzrůstat. Pokud to uděláte, zajistíte tím jedinečnost každého čísla odkazu v celém hierarchckém seznamu. Protože číslo odkazu na položku je Long Integer, můžete přidávat položky mnohokrát. (Nezapomeňte, že když používáte tisíce položek, bude lepší použít tabulku a ne seznam.)

Poznámka: Jestliže používáte Bitwise operátory , můžete použít čísla odkazu na položku k uložení informací, které se vejdou do Long Integer hodnoty. T.j., 2 Integer hodnoty, 4 bytové hodnoty , nebo 32 Logických hodnot.

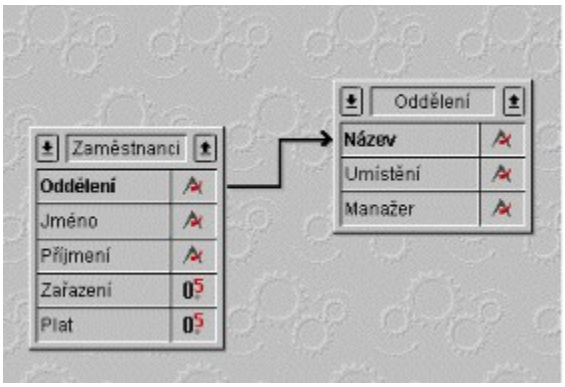
Proč potřebujete jedinečné číslo odkazu na položku?

Ve většině případů, když používáte hierarchické seznamy pro rozhraní uživatele a když pouze pracujete s označenými položkami (ta na kterou bylo klepnuto, nebo byla tažena), nebudete potřebovat čísla odkazů na položky. S použitím Selected list item a GET LIST ITEM máte vše ,co potřebujete k práci s označenou položkou. Dále příkazy jako INSERT LIST ITEM a DELETE LIST ITEM vám umožní pracovat se seznamem vztaženě k označené položce.

Čísla položek budete potřebovat pro případy ,kdy chcete pracovat s položkami seznamu programově a nestačí vám pouze označené položky.

Příklad

Zde je pohled na strukturu databáze:



Tabulky [Oddělení] a [Zaměstnanci] obsahují následující záznamy:

Oddělení

Název	Umístění	Manažer
Prodej	1 patro	Jan Prachař
Marketing	1 patro, Východ	Petr Skřivan
Služby zákazníkům	2 patro, Západ	Omar Shacker
Prázdninový Management	Někde	K založení

Zaměstnanci

Oddělení	Jméno	Příjmení
Prodej	Jaroslav	Plecháček
Prodej	Filip	Malík
Marketing	Helena	Schalková
Marketing	Aleš	Poťouchlo
Služby zákazníkům	Radek	Štveráček
Prodej	Zuzana	Šottová

Chcete zobrazit hierarchický seznam, nazvaný hList který ukáže Oddělení a pro každé Oddělení přiřadit seznam, který bude obsahovat Zaměstnance pracující v Oddělení. Metoda objektu pro hList je:

```
Metoda objektu hList - Hierarchický seznam
Case of
   ÷ (Form event=On Load)
      C_LONGINT(hlList;$hPodlist;$vlOddělení;$vlEmployee)
         ` Vytvořit nový prázdný hierarchický seznam
      hlList:=New list
         ` Vybrat všechny záznamy z tabulky [Oddělení]
      ALL RECORDS([Oddělení])
         ` Pro každé Oddělení
      For ($vlOddělení;1;Records in selection([Oddělení]))
            ` Označit zaměstnance z tohoto oddělení
         RELATE MANY([Oddělení]Název)
            ` Kolik jich je?
         $vlNbZaměstnanci:=Records in selection([Zaměstnanci])
            ` Je alespoň jeden zaměstnanec v Oddělení?
         If ($vlNbZaměstnanci>0)
               ` Vytvořit podseznam pro oddělení
            $hPodlist:=New list
```

```
            ` Pro každého zaměstnance
        For ($vlEmployee;1;Records in selection([Zaměstnanci]))
                ` Přidat položku Zaměstnance do poseznamu
                ` Všiměte si že číslo záznamu [Zaměstnanci]
                ` je přiřazen jako číslo odkazu na položku
            APPEND TO LIST($hPodlist;[Zaměstnanci]Jméno+", "
            +[Zaměstnanci]Příjmení;Record number([Zaměstnanci]))
                ` Jít na další záznam [Zaměstnanci]
            NEXT RECORD([Zaměstnanci])
        End for
    Else
            ` Žádný zaměstnanec, Žádný podseznam pro Oddělení
        $hPodlist:=0
    End if
        ` Přidat položku Oddělení do hlavního seznamu
        ` Všiměte si že číslo záznmu [Oddělení]
        ` Je přiřazeno jako číslo odkazu na položku. Bit #31
        ` číslo odkazu na záznam je vnuceno a tím budeme moci odlišit
        ` položku Oddělení od položky Zaměstnance. Podívejte se na poznámku
        ` dále aby jste zjistili proč tento bit můžeme pužít jako dodatkovou
        ` informaci o položce.
    APPEND TO LIST(hlList;[Oddělení]Název;0x80000000
        | Record number([Oddělení]);$hPodlist;$hPodlist    # 0)
        ` Nastavíme položky oddělení Tučně, aby bylo snažší se vyznat
    SET LIST ITEM PROPERTIES(hlList;0;False;Bold;0)
        ` Jít na další oddělení
    NEXT RECORD([Oddělení])
End for
    ` Třídit celý seznam ve vzestupném pořadí
SORT LIST(hlList;>)
    ` Zobrazit okno v typu Windows
    ` a vnutit mimální výšku 14 Pts
SET LIST PROPERTIES(hlList;ala Windows;Windows node;14)
÷ (Form event=On Unload)
    ` Seznam již není potřeba, nezapomeňte jej odstranit!
CLEAR LIST(hlList;*)
÷ (Form event=On Double Clicked)
        ` Bylo provedeno poklepání
        ` Zjistit pozici označené položky
    $vlPozPolozka:=Selected list item(hlList)
        ` Pouze v případě ověřit pozici
    If ($vlPozPolozka    # 0)
            ` Získat informace o položce
        GET LIST ITEM(hlList;$vlPozPolozka;$vlPolozkaRef;$vsPolozkaText;
                                $vlItemSubList;$vbItemSubExpanded)
            ` Je položka položkou Oddělení?
        If ($vlPolozkaRef ?? 31)
                ` Pokud ano, bylo poklepáno na položku Oddělení
            ALERT("Bylo poklepáno na položku oddělení "+Char(34)+ $vsPolozkaText+Char(34)+".")
        Else
                ` Pokud ne, bylo poklepáno na položku Zaměstnance
                ` Pomocí odkazu na rodičovskou položku nalézt záznam [Oddělení]
```

```
            GOTO RECORD([Oddělení];List item parent(hlList;$vlPolozkaRef) ?-31)
               ` Řekne kde Zaměstnanec pracuje a komu dává zprávy
            ALERT("Poklepal jste na položku zaměstnance"+Char(34) +$vsPolozkaText+Char(34)+
            " který pracuje v Oddělení"+Char(34) +[Oddělení]Název+Char(34)+
            " jeho manažer je "+Char(34) +[Oddělení]Manažer+Char(34)+".")
         End if
      End if
   End case
` Poznámka: 4th Dimension může obsahovat až 16 milionů záznamů na tabulku
` (přesně 16 777 215). Tato hodnota je 2^24 minus jedna. Číslo záznamu
` zabírá 24 bitů. V našem příkladu, používáme bit #31 nepoužitého vyššího bitu
` oddělující položky Zaměstnanci a Oddělení.
```

V tomto příkladu je pouze jeden důvod pro odlišení položek [Oddělení] a [Zaměstnanci]:

1. Jako číslo odkazu na položku používáme číslo záznamu a proto by se nám mohlo stát, že budeme mít stejné číslo odkazu jak u Oddělení tak u Zaměstnanců.

2. Používáme příkaz List item parent k získání rodiče označené položky. Pokud by jsme hledali Zaměstnance s číslem záznamu 10, a existovalo by i Oddělení s číslem záznamu 10, příkaz by našel Oddělení dříve a vrátil by tedy položku Oddělení a ne položku Zaměstnance.

Proto jsme vytvořili číslo odkazu na položku jedinečné, ne proto že by jsme potřebovali jedinečné rozlišení, ale protože jsme potřebovali rozlišit záznamy Zaměstnanců a Oddělení.

V prostředí Uživatele nebo Vlastní nabídky by seznam vypadal následovně:

Poznámka: Tento příklad je použitelný pro rozhraní uživatele pokud používáte relativně malé množství záznamů. Nezapomeňte že seznamy se ukládají do paměti - proto nevytvářejte seznamy s tisíci položek.

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

INSERT LIST ITEM (VLOŽIT POLOŽKU SEZNAMU)

Verze 6.0

INSERT LIST ITEM (list; PředPolRef | *; PoložkaText; PoložkaRef {; podlist {; rozšiř}})

Parametr	Typ		Popis
list	ListRef	→	Číslo odkazu na seznam
PředPolRef | *	Číslo | *	→	Číslo odkazu na položku seznamu, před níž se vloží nová položka, nebo * pro předání před právě vybranou položku
PoložkaText	Řetězec	→	Text pro novou položku seznamu
PoložkaRef	Číslo	→	číslo odkazu na položku seznamu
podlist	ListRef	→	číslo odkazu na podseznam k položce
rozšiř	Logické	→	Je-li podseznam připojen: TRUE - zobrazí se rozšířený

Popis

Příkaz INSERT LIST ITEM vloží novou položku do seznamu jehož číslo odkazu předáte parametrm *list*.

Pokud předáte * jako druhý parametr, je nová položka předána před platnou označenou položku v seznamu. V tomto případě bude nově předáná položka označená.

Jinak pokud chcete vložit položku před secifickou položku, předáte číslo odkazu na položku pro tuto položku. V tomto případě nebude nově předáná položka automaticky označena. Pokud neexistuje položka s takovým číslem odkazu, příkaz neprovede nic.

Text a číslo odkazu na položku předáte do parametrů *PoložkaText* a *PoložkaRef*.

Poznámka: I když jsou oba volitelné, jak parametr *podlist* tak *rozšiř*, musí být předány spojeně.

Příklad

Následující kód vloží položku (bez přiřazeného podseznamu) před označenou položku:

```
vlJedinecnaRef:=vlJedinecnaRef+1
INSERT LIST ITEM(hList;*;"Nová položka";vlJedinecnaRef)
REDRAW LIST(hList)
```

Dále si přečtěte

APPEND TO LIST.

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET LIST ITEM PROPERTIES (NASTAVIT VLASTNOSTI POLOŽKY SEZNAMU)

Příkazy a odkazy pro Hierarchické seznamy

Verze 6.0

SET LIST ITEM PROPERTIES (list; PoložkaRef; dostup; styl; ikona)

Parametr	Typ		Popis
list	ListRef	→	Číslo odkazu na seznam
PoložkaRef	Číslo	→	Číslo odkazu na položku, nebo 0 pro poslední připojenou
dostup	Logické	→	TRUE = položka bude dostupná pro zadávání FALSE = není dostupná
styl	Číslo	→	Styl písma
ikona	Číslo	→	ikona ze zdroje Cicn, nebo 65536 + číslo zdroje PICT, nebo 131072 + číslo odkazu z knihovny obrázků

Popis

Příkaz SET LIST ITEM PROPERTIES upraví položku, jejíž číslo odkazu je předáno *PoložkaRef* v seznamu jehož číslo předáno *list*.

Pokud neexistuje položka s předaným číslem odkazu, příkaz neprovede nic. Při přidání položky pomocí APPEND TO LIST můžete do SET LIST ITEM PROPERTIES v parametru *PoložkaRef* předat 0 k upravě poslední předáné položky.

Pokud pracujete s čísly odkazu na položku, vytvořte seznam tak, aby položky měly jedinečná čísla, jinak nebudete schopni rozlišit jednotlivé položky. Jestli chcete vědět více informací, přečtěte si příkaz APPEND TO LIST.

Poznámka: K změně textu položky seznamu nebo podseznamu použijte příkaz SET LIST ITEM.

Aby byla položka dostupná, vložte TRUE do parametru dostup, jinak vložte FALSE.

Důležité: Aby byla položka dostupná, musí být v seznamu, který je dostupný. Aby jste udělali seznam dostupný, použijte příkaz SET ENTERABLE. K zdostupnění jednotlivých položek použijte příkaz SET LIST ITEM PROPERTIES. Změna dostupnosti na úrovni seznamu nemá žádný vliv na vlastnost dostupnosti položek. Nicméně položka bude fyzicky dostupná pouze tehdy, je-li dostupný seznam.

Styl písma pro položku definujete v parametru *styl*. Vkládáte kombinaci (čísla se sčítají) následujících předdefinovaných konstant:

Konstanta	*Typ*	*Hodnota*
Plain	*Long Integer*	*0*
Bold	*Long Integer*	*1*
Italic	*Long Integer*	*2*
Underline	*Long Integer*	*4*
Outline	*Long Integer*	*8*
Shadow	*Long Integer*	*16*
Condensed	*Long Integer*	*32*
Extended	*Long Integer*	*64*

Poznámka: Na Windows jsou možné pouze kombinace Bold, Italic a Underline a samozřejmě Plain.

K přiřazení ikony k položce, vložte jednu z následujících hodnot:

- N, kde N je ID zdroje Cicn
- Use pict resource+N, kde N je ID zdroje PICT
- Use PicRef+N, kde N je číslo odkazu na obrázek z Knihovny obrázků v Prostředí návrháře

Předejte 0 pokud pro položkuseznamu nechcete žádné obrázky.

Poznámka: Use PICT resource a Use PicRef jsou předdefinované konstanty 4D.

Příklad

Podívejte se na příklad u příkazu APPEND TO LIST.

Dále si přečtěte

GET LIST ITEM PROPERTIES, SET LIST ITEM.

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GET LIST ITEM PROPERTIES (ZÍSKAT VLASTNOSTI POLOŽKY SEZNAMU)

Verze 6.0

GET LIST ITEM PROPERTIES (list; PoložkaRef; dostup {; styl{; ikona}})

Parametr	Typ		Popis
list	ListRef	→	Číslo odkazu na seznam
PoložkaRef	Číslo	→	Číslo odkazu na položku, nebo 0 pro poslední připojenou položku
dostup	Logické	←	TRUE = položka bude dostupná pro zadávání FALSE = není dostupná
styl	Číslo	←	Styl písma
ikona	Číslo	←	ikona ze zdroje Cicn, nebo 65536 + číslo zdroje PICT, nebo 131072 + číslo odkazu z knihovny obrázků

Popis

Příkaz GET LIST ITEM PROPERTIES vrací vlastnosti položky, jejíž číslo jste zadali do parametru *PoložkaRef* , patřící seznamu, jehož číslo odkazu je *list*.

Po provedení příkazu:

• *dostup* vrací TRUE pokud je položka dostupná
• *styl* vrací styl písma položky
• *ikona* vrací číslo ikony nebo obrázeku přiřazeného k položce, 0 pokud není.

Jestli chcete vědět více informací o těchto parametrech, přečtěte si popis k příkazu SET LIST ITEM PROPERTIES.

Pokud neexistuje položka se zadaným číslem odkazu, příkaz ponechá parametry nezměněné.

Pokud pracujete s čísly odkazu na položku, vytvořte seznam, ve kterém položky mají jedinečná čísla, jinak nebudete schopni rozlišit jednotlivé položky. Jestli chcete vědět více informací, přečtěte si příkaz APPEND TO LIST.

Dále si přečtěte

GET LIST ITEM, SET LIST ITEM, SET LIST ITEM PROPERTIES.

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

List item position (Pozice položky v seznamu)

Příkazy a odkazy pro Hierarchické seznamy

Verze 6.0

List item position (list; PoložkaRef) → Číslo

Parametr	Typ		Popis
list	ListRef	→	Číslo odkazu na seznam
PoložkaRef	Číslo	→	Číslo odkazu na položku
Výsledek funkce	Číslo	←	Pozice položky v rozbaleném seznamu

Popis

Příkaz List item position vrací pozici položky, jejíž číslo jste zadali do parametru *PoložkaRef* , patřící do seznamu s číslem odkazu předaném v *list*.

Pozice je vyjádřena relativně ve vztahu k první položce hlavního seznamu s uvážením platných rozbalených/zabalený seznamů a podseznamů.

Výsledek je číslo mezi 1 a výsledkem funkce Count list items.

Pokud není položka viditelná, protože je v zabaleném seznamu, List item position tento seznam rozbalí aby byla položka viditelná.

Pokud položka neexistuje, vrátí příkaz 0.

Dále si přečtěte

Count list items, SELECT LIST ITEM BY REFERENCE.

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

List item parent (Rodič položky seznamu)

Příkazy a odkazy pro Hierarchické seznamy

Verze 6.0

List item parent (list; PoložkaRef) → Číslo

Parametr	Typ		Popis
list	ListRef	→	Číslo odkazu na seznam
PoložkaRef	Číslo	→	Číslo odkazu na položku
Výsledek funkce	Číslo	←	Číslo odkazu na rodičovskou položku - 0 pokud není

Popis

Příkaz List item parent vrací číslo odkazu na rodičovskou položku.

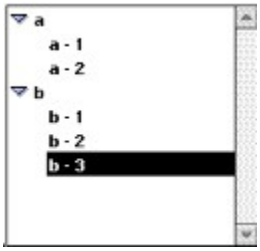
Číslo odkazu na seznam předáte parametrem *list*; číslo odkazu na položku tohoto seznamu předáte parametrem *PoložkaRef*. Za předpokladu že *PoložkaRef* se odkazuje ne existující položku, a že tato položka je v podseznamu (a proto má rodičovskou položku), obdržíte číslo odkazu na rodičovskou položku.

Pokud číslo odkazu na položku které předáte parametrem *PoložkaRef* se neodkazuje na existující položku, List item parent vrátí 0 (nula).

Pokud pracujete s čísly odkazu na položku, vytvořte seznam ve kterém položky mají jedinečná čísla, jinak nebudete schopni rozlišit jednotlivé položky. Jestli chcete vědět více informací, přečtěte si příkaz APPEND TO LIST.

Příklady

Máme seznam nazvaný hList který v Prostředí uživatele vypadá takto:



Čísla odkazů na položky jsou nastavena následovně:

Položka	Číslo odkazu na položku
a	100
a - 1	101
a - 2	102
b	200
b - 1	201
b - 2	202
b - 3	203

• Pokud je v následujícím příkladu označena položka "b - 3", proměnná $vlRodicPolozkaRef bude 200, číslo odkazu na položku "b":

```
$vlPozPolozka:=Selected list item(hList)
```

GET LIST ITEM(hList;$vlPozPolozka;$vlPolozkaRef;$vsPolozkaText)
$vlRodicPolozkaRef:=**List item parent**(hList;$vlPolozkaRef) ` $vlRodicPolozkaRef je 200

• Pokud je označena položka "a - 1", proměnná $vlRodicPolozkaRef bude 100, číslo odkazu na položku "a".

• Pokud je označena položka "a" nebo "b", proměnná $vlRodicPolozkaRef bude 0, protože tyto položky nemají rodičovskou položku

Dále si přečtěte

GET LIST ITEM, List item position, SELECT LIST ITEM BY REFERENCE, SET LIST ITEM.

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DELETE LIST ITEM (VYMAZAT POLOŽKU SEZNAMU)

Příkazy a odkazy pro Hierarchické seznamy

Verze 6.0

DELETE LIST ITEM (List; položkaRef | *{; *})

Parametr	Typ		Popis
list	ListRef	→	Odkaz na seznam
položkaRef	*	Číslo	*	→	Číslo odkazu na položku nebo * pro platnou označenou položku
*		→	Pokud ano, vymazat podseznam (pokud je nějaký) z paměti Pokud ne, podseznam (pokud je nějaký) není vymazán

Popis

Příkaz DELETE LIST ITEM vymaže ze seznamu položku, jejíž odkaz definujete v parametru *položkaRef*.

Pokud předáte * jako druhý parametr, vymaže se platná označená položka v seznamu.

Jinak musíte definovat číslo odkazu na položku, kterou chcete vymazat. Pokud neexistuje položka s tímto číslem odkazu, příkaz neprovede nic.

Pokud pracujete s čísly odkazu na položku, vytvořte seznam, ve kterém položky mají jedinečná čísla, jinak nebudete schopni rozlišit jednotlivé položky. Jestli chcete vědět více informací, přečtěte si příkaz APPEND TO LIST.

Nezáleží na tom kterou položku mažete, ale když předáte volitelný parametr * 4D automaticky vymaže podseznam přiřazený k položce, pokud nějaký existuje. Pokud tento parametr nezadáte, je dobré si před tím zjistit číslo odkazu na podseznam přiřazený k položce, protože jej pak budete moci vymazat použitím příkazu CLEAR LIST.

Příklad

Následující příklad vymaže platnou označenou položku ze seznamu hList. Pokud má tato položka přiřazený podseznam, je tento podseznam také vymazán:

```
DELETE LIST ITEM(hList;*;*)
   ` NEZAPOMEŇTE provést příkaz REDRAW LIST jinak nebude seznam obnoven
REDRAW LIST(hList)
```

Dále si přečtěte

CLEAR LIST, GET LIST ITEM.

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GET LIST ITEM (ZÍSKAT POLOŽKU SEZNAMU)

Příkazy a odkazy pro Hierarchické seznamy

Verze 6.0

GET LIST ITEM (list; PoložkaPoz; PoložkaRef; PoložkaText {; podlist{; rozšiř}})

Parametr	Typ		Popis
list	ListRef	→	Číslo odkazu na seznam
PoložkaPoz	Číslo	→	Pozice položky v rozbaleném seznamu
PoložkaRef	Číslo	←	číslo odkazu na položku seznamu
PoložkaText	Řetězec	←	Text pro novou položku seznamu
podlist	ListRef	←	číslo odkazu na podseznam k položce
rozšiř	Logické	←	Je-li podseznam připojen TRUE - podseznam je rozšířený FALSE - podseznam je zabalený

Popis

Příkaz GET LIST ITEM vrací informace o položce, jejíž pozice je předána parametrem *PoložkaPoz* , ze seznamu definovaném parametrem *list*.

Pozice je relativní, protože závisí na aktuáně zabalených a rozbalených položkách a podseznamech. Pozice je číslo mezi 1 a číslem vraceným funkcí Count list items. Pokud předáte číslo mimo tento rozsah, GET LIST ITEM vrátí parametry nezměněné.

Po provedení příkazu obdržíte:

• Číslo odkazu na položku v parametru *PoložkaRef*.
• Text položky v parametru *PoložkaText*

Pokud předáte i volitelné parametry *podlist* a *rozšiř*:

• podlist bude obsahovat číslo odkazu na podseznam přiřazený k položce. Pokud položka nemá podseznam, bude jeho hodnota nula (0).
• Pokud má položka podseznam, parametr rozšiř vrátí TRUE pokud je tento podseznam rozbalený, jinak vrátí FALSE.

Příklad

hList je seznam, jehož položky mají jedinečná čísla odkazů. Následující kód programově přepíná stav rozbalený a zabalený podseznamu přiřazenému aktuálně označené položce (je-li nějaký):

```
$vlPozPolozka:=Selected list item(hList)
If ($vlPozPolozka>0)
   GET LIST ITEM(hList;$vlPozPolozka;$vlPolozkaRef;$vsPolozkaText; $hPodlist;$vbRozsir)
   If (Is a list($hPodlist))
      SET LIST ITEM(hList;$vlPolozkaRef;$vsPolozkaText;$vlPolozkaRef;$hPodlist; Not($vbRozsir))
      REDRAW LIST(hList)
   End if
End if
```

Dále si přečtěte

GET LIST ITEM PROPERTIES, List item parent, List item position, Selected list item, SET LIST ITEM, SET LIST ITEM PROPERTIES.

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET LIST ITEM (NASTAVIT POLOŽKU SEZNAMU)

Příkazy a odkazy pro Hierarchické seznamy

Verze 6.0

SET LIST ITEM (list; PoložkaRef; NovýPolText; NováPolRef {; podlist{; rozšiř}})

Parametr	Typ		Popis
list	ListRef	→	Číslo odkazu na seznam
PoložkaRef	Číslo	→	Číslo odkazu na položku seznamu
NovýPolText	Řetězec	→	Nový text položky
NováPolRef	Číslo	→	Nové číslo odkazu na položku
podlist	ListRef	→	číslo odkazu na podseznam k položce
rozšiř	Logické	→	Je-li podseznam připojen TRUE - rozšířit podseznam FALSE - zabalit podseznam

Popis

Příkaz SET LIST ITEM upraví položku, jejíž číslo odkazu je předáváno parametrem *PoložkaRef* , ze seznamu předaném parametrem *list*.

Pokud neexistuje položka se zadaným číslem odkazu, příkaz neprovede nic. Do parametru *položkaRef* můžete vložit nulu (0) pro upravení poslední položky připojené příkazem APPEND TO LIST.

Pokud pracujete s čísly odkazu na položku, vytvořte seznam ve kterém položky mají jedinečná čísla, jinak nebudete schopni rozlišit jednotlivé položky. Jestli chcete vědět více informací, přečtěte si příkaz APPEND TO LIST.

Nový text pro položku předáte parametrem *NovýPolText*. K upravení čísla odkazu na položku,vložte novou hodnotu do parametru *NováPolRef*; jinak vložte stejnou hodnotu jaká je v *položkaRef*.

K přiřazení seznamu k položce zadejte číslo odkazu na tento seznam do parametru *podlist*. V tomto případě také definujte parametr *rozšiř*. Pokud jej nastavíte na TRUE, bude podseznam rozbalený a pokud předáte FALSE bude zabalený.

K odřazení podseznamu od položky, vložte do parametru *podlist* nula (0). V tomto případě je dobré mít uložené číslo odkazu na tento seznam pomocí příkazu GET LIST ITEM aby, pokud jej nebudete potřebovat, jste jej později mohli vymazat pomocí příkazu CLEAR LIST.

Pokud nechcete měnit vlastnosti podseznamu, vložte do parametru *podlist* hodnotu -1.

Poznámka: I když jsou oba volitelné, jak parametr *podlist* tak *rozšiř*, musí být předány současně.

Příklad

1. hList je seznam jehož položky mají jedinečná čísla odkazů. Následující metoda objektu pro tlačítko přidá podseznam k platné označené položce

```
$vlPozPolozka:=Selected list item(hList)
If ($vlPozPolozka>0)
   GET LIST ITEM(hList;$vlPozPolozka;$vlPolozkaRef;$vsPolozkaText; $hPodlist;$vbRozsir)
   $vbNovyPodList:=Not(Is a list($hPodlist))
   If ($vbNovyPodList)
      $hPodlist:=New list
   End if
   vlJedinecnaRef:=vlJedinecnaRef+1
```

```
        APPEND TO LIST($hPodlist;"Nová položka";vlJedinecnaRef)
        If ($vbNovyPodList)
            SET LIST ITEM(hList;$vlPolozkaRef;$vsPolozkaText;$vlPolozkaRef;$hPodlist;True)
        End if
        SELECT LIST ITEM BY REFERENCE(hList;vlJedinecnaRef)
        REDRAW LIST(hList)
    End if
```

2. Podívejte se na příklad u příkazu GET LIST ITEM.

3. Podívejte se na příklad u příkazu APPEND TO LIST.

Dále si přečtěte

GET LIST ITEM, GET LIST ITEM PROPERTIES, SET LIST ITEM PROPERTIES.

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Selected list item (Označená položka seznamu)

Příkazy a odkazy pro Hierarchické seznamy

Verze 6.0

Selected list item (list) → Long Integer

Parametr	Typ		Popis
list	ListRef	→	Odkaz na seznam
Výsledek funkce	Long Integer	←	Pozice platné označené položky v rozevřeném seznamu

Popis

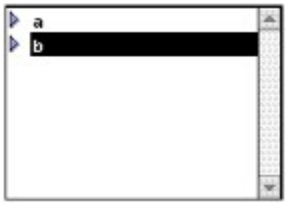
Příkaz Selected list item vrací pozici označené položky v seznamu definovaném parametrem *list*.

Tento příkaz se používá pro seznamy zobrazené ve formuláři k určení uživatelem označené položky.

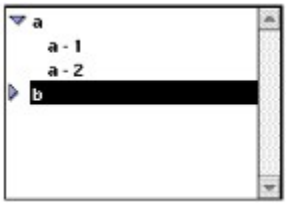
Pokud má seznam podseznam, použije se příkaz na hlavní seznam (ten který je zobrazen ve formuláři) ne na jeho podseznamy. Pozice je vypočítávána relativně podle první položky hlavního seznamu s ohledem na stav rozšíření podseznamů. Pozice vždy záleží na tom, kolik je nad označenou položkou rozevřeno nebo zavřeno podseznamů.

Příklad

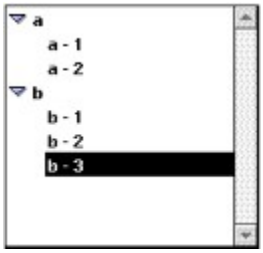
Zde je seznam pojmenovaný hList jak vypadá v prostředí uživatele:



$vlPozPolozka:=**Selected list item**(hList) ` v tomto případě bude $vlPozPolozka 2



$vlPozPolozka:=**Selected list item**(hList) ` v tomto případě bude $vlPozPolozka 4



$vlPozPolozka:=**Selected list item**(hList) ` v tomto případě bude $vlPozPolozka 7

$vlPozPolozka:=**Selected list item**(hList) ` v tomto případě bude $vlPozPolozka 5

Dále si přečtěte

SELECT LIST ITEM, SELECT LIST ITEM BY REFERENCE.

Příkazy a odkazy pro Hierarchické seznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SELECT LIST ITEM (OZNAČIT POLOŽKU SEZNAMU)

Příkazy a odkazy pro Hierarchické seznamy

Verze 6.0

SELECT LIST ITEM (list; PoložkaPoz)

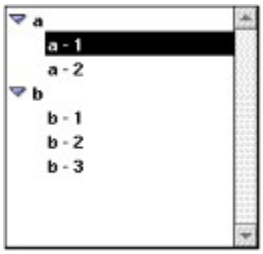
Parametr	Typ		Popis
list	ListRef	→	Odkaz na seznam
PoložkaPoz	Číslo	→	Pozice položky v rozšířeném seznamu

Popis

Příkaz SELECT LIST ITEM označí položku jejíž pozici jste zadali do parametru *PoložkaPoz* a je umístěná v seznamu definovaném parametrem *list*. Parametr *PoložkaPoz* je závislý na aktuálně zabalených/rozevřených seznamech a podseznamech. Hodnota pozice musí být mezi 1 a číslem vráceným příkazem Count list items. Pokud předáte hodnotu mimo tento rozsah, bude jako výchozí označena první položka.

Příklady

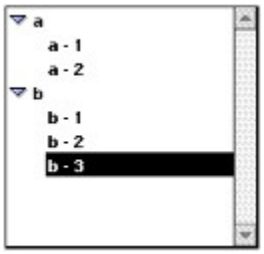
Máme seznam s názvem hList, ukázaný zde v prostředí uživatele:



Po spuštění tohoto kódu:

```
SELECT LIST ITEM(hList;Count list items(hList))
   ` NEZAPOMEŇTE provést příkaz REDRAW LIST jinak nebude seznam obnoven
REDRAW LIST(hList)
```

Bude označena poslední viditelná položka:



Dále si přečtěte

SELECT LIST ITEM BY REFERENCE, Selected list item.

SELECT LIST ITEM BY REFERENCE (OZNAČIT POLOŽKU SEZNAMU PODLE ODKAZU)

Verze 6.0

SELECT LIST ITEM BY REFERENCE (list; PoložkaRef)

Parametr	Typ		Popis
list	ListRef	→	Odkaz na seznam
PoložkaRef	Číslo	→	Odkaz na číslo položky

Popis

Příkaz SELECT LIST ITEM BY REFERENCE označí položku jejíž číslo odkazu jste zadali do parametru *PoložkaRef* a je umístěna v seznamu definovaném parametrem *list*.

Pokud neexistuje položka s tímto číslem odkazu, příkaz neprovede nic.

Pokud není položka viditelná (je umístěna v zavřeném podseznamu), příkaz tento podseznam rozevře a označí nyní již viditelnou položku.

Pokud pracujete s čísly odkazu na položku, vytvořte seznam ve kterém položky mají jedinečná čísla, jinak nebudete schopni rozlišit jednotlivé položky. Jestli chcete vědět více informací, přečtěte si příkaz APPEND TO LIST.

Příklad

hList je seznam jehož položky mají jedinečná čísla odkazů. Následující metoda objektu pro tlačítko označí rodičovskou položku (pokud je nějaká) platné označené položky:

```
` Získat pozici označené položky
$vlPozPolozka:=Selected list item(hList)
   ` Získat číslo odkazu označené položky
GET LIST ITEM(hList;$vlPozPolozka;$vlPolozkaRef;$vsPolozkaText)
   ` Získat číslo odkazu rodičovské položky (pokud existuje)
$vlRodicPolozkaRef:=List item parent(hList;$vlPolozkaRef)
If ($vlRodicPolozkaRef>0)
      ` Označit rodičovskou položku
   SELECT LIST ITEM BY REFERENCE(hList;List item parent (hList;$vlPolozkaRef))
   ` NEZAPOMEŇTE provést příkaz REDRAW LIST jinak nebude seznam obnoven
   REDRAW LIST(hList)
End if
```

Dále si přečtěte

SELECT LIST ITEM, Selected list item.

4th Dimension 6.5, Seznam témat konstant

IMPORT TEXT (IMPORTOVAT TEXT)

Příkazy a odkazy pro Import a Export

Verze 3

IMPORT TEXT ({tabulka;} dokument)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka do které importovat data nebo výchozí tabulka pokud vynecháno
dokument	Řetězec	→	Textový dokument ze kterého importovat data

Popis

Příkaz IMPORT TEXT přečte data z dokumentu *dokument* (Windows nebo Macintosh dokument) do tabulky *tabulka* a vytvoři z něj nové záznamy pro tabulku.

Operace importu je prováděna přes platný vstupní formulář. Import čte do jednotlivých polí a proměnných podle jejich umístění ve formuláři, podle jejich vrstvení. Z tohoto důvodu musíte být velmi opatrní na pořadí textových objektů ve vrstvách. První objekt do kterého mají být data importována musí být na pozadí atd. Pokud počet proměnných a polí ve formuláři nesouhlasí s počtem importovaných polí, jsou přebytečné ignorovány. Vstupní formulář použitý pro import nesmí obsahovat žádná tlačítka. Objekty podformuláře jsou ignorovány.

Poznámka: Jeden ze způsobů jak zajistit že budou data správně importována je označit objekt, do kterého má být importováno první pole a přenést jej na popředí. Pokračujte v přesunování proměnných a polí na popředí v pořadí v jakém chcete provést import, ujistěte se že máte jedno pole nebo proměnnou pro každé importované pole.

Při uložení nového záznamu z importu je poslána událost Při potvrzení vstupu. Pokud při importu používáte proměnné, můžete využít tuto událost ke zkopírování dat z proměnných do polí.

Parametr dokument může obsahovat cestu s názvy složek. Pokud předáte prázdný řetězec, otevře se standardní okno systému Otevřít soubor. Pokud uživatel okno zruší, je zrušena celá operace importu a systémová proměnná OK je nastavena na 0.

Během importu je zobrazen teploměr. Uživatel může import přerušit stisknutím tlačítka s popisem Stop. Záznamy které již byli importovány však nebudou odstraněny. Pokud je import správně dokončen, je proměnná OK nastavena na 1. Pokud se objeví chyby nebo uživatel import přeruší je OK nastavena na 0. Teploměr může být skryt pomocí příkazu MESSAGES OFF.

Operace importu je provedena s použitím výchozí ASCII mapy pro platformu, na které probíhá, pokud tuto mapu nezměníte (pomocí příkazu USE ASCII MAP). ASCII mapa může být použita k převodu dat z jiné platformy, která má jinou ASCII tabulku.

Při použití IMPORT TEXT je výchozí oddělovač polí znak tebelátor (ASCII 9) a výchozí oddělovač záznamů je return (ASCII 13). Tyto oddělovače můžete změnit přiřazením nových hodnot do systémových proměnných: *FldDelimit* a *RecDelimit*. Uživatel může změnit tyto hodnoty v prostředí uživatele v dialogovém okně Import dat. Textová pole mohou obsahovat return (nový řádek) a proto buďte opatrní při použití return jako oddělovače při importování textových polí.

Příklad

Následující příklad importuje data z textového dokumentu. Metoda nejdříve nastaví vstupní formulář, aby se data importovala do správného formuláře, změní oddělovače a pak provede import:

```
INPUT FORM([Lidé]; "Import")
FldDelimit:=27 ` Nastaví oddělovač polí na znak Escape
RecDelimit:=10 ` Nastaví oddělovač záznamů na znak Line Feed
IMPORT TEXT([Lidé];"NovíLidé") ` Import z dokumentu "NovíLidé"
```

Dále si přečtěte

EXPORT TEXT, IMPORT DIF, IMPORT SYLK, USE ASCII MAP.

Systémové proměnné a Sady

OK je nastavena na 1 pokud je import správně dokončen, jinak je nastavena na 0.

Příkazy a odkazy pro Import a Export

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

EXPORT TEXT (EXPORTOVAT TEXT)

Příkazy a odkazy pro Import a Export

Verze 3

EXPORT TEXT ({tabulka;} dokument)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ze které exportovat data nebo vychozi tabulka pokud vynecháno
dokument	Řetězec	→	Textový dokument k přijetí dat

Popis

Příkaz EXPORT TEXT zapíše data z platného výběru záznamů platného procesu. Data jsou zapsána do dokumentu, Windows nebo Macintosh dokument.

Operace exportu je provedena přes platný výstupní formulář. Export dat je prováděn v pořadí vstupu dat výstupního formuláře. Z tohoto důvodu použijte výstupní formulář který obsahuje pouze pole nebo objekty které chcete exportovat. Na exportní formulář neumísťujte žádná tlačítka nebo jiné zvláštní objekty. Objekty podformuláře budou ignorovány.

Při dokončení exportu každého záznamu je poslána událost Při zavedení. Použijte tuto událost k nastavení proměnných které máte ve formuláři exportu.

Dokument může obsahovat název nového nebo existujícího dokumentu. Pokud má dokument stejný název jako existující dokument, je obsah existujícího dokumentu přepsán. Parametr může obsahovat cestu včetně názvů složek a disků. Pokud předáte prázdný řetězec, otevře se standardní systémové okno Uložit soubor. Pokud uživatel toto okno zruší, je zrušen celý export a systémová proměnná OK je nastavena na 0.

Během exportu je zobrazen teploměr. Uživatel může kdykoli operaci přerušit stisknutím tlačítka s popisem Stop Pokud je eport správně dokončen, je proměnná OK nastavena na 1. Jestliže je export přerušen uživatelem nebo se vyskytnou nějaké chyby, je proměnná OK nastavena na 0. Teploměr můžete skrýt s použitím příkazu MESSAGES OFF.

Operace exportu je provedena s použitím výchozí ASCII mapy pro platformu na které probíhá. Pokud tuto mapu nezměníte (pomocí příkazu USE ASCII MAP). ASCII mapa může být použita k převodu dat z jiné platformy která má jinou ASCII tabulku.

Při použití EXPORT TEXT je výchozí oddělovač polí znak tabelátor (ASCII 9) a výchozí oddělovač záznamů je return (ASCII 13). Tyto oddělovače můžete změnit přiřazením nových hodnot do systémových proměnných: *FldDelimit* a *RecDelimit*. Uživatel může změnit tyto hodnoty v prostředí uživatele v dialogovém okně Export dat. Textová pole mohou obsahovat return a proto buďte opatrní při použití return jako oddělovače při exportování textových polí.

Příklad

Tento příklad exportuje data do textového dokumentu. Metoda nejdříve nastaví výstupní formulář pak změní proměnné oddělovačů a nakonec vyexportuje záznamy:

```
OUTPUT FORM([Lidé]; "Export")
FldDelimit:=27 ` Nastaví oddělovač polí na znak Escape
RecDelimit:=10 ` Nastaví oddělovač záznamů na znak Line Feed
EXPORT TEXT([Lidé];"NovíLidé") ` Import do dokumentu "NovíLidé"
```

Dále si přečtěte

EXPORT DIF, EXPORT SYLK, IMPORT TEXT, USE ASCII MAP.

Systémové proměnné a Sady

OK je nastavena na 1 pokud je export správně dokončen, jinak je nastavena na 0.

Příkazy a odkazy pro Import a Export

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

IMPORT SYLK (IMPORTOVAT FORMÁT SYLK)

Příkazy a odkazy pro Import a Export

Verze 3

IMPORT SYLK ({tabulka;} dokument)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka do které importovat data nebo výchozí tabulka pokud vynecháno
dokument	Řetězec	→	SYLK dokument ze kterého importovat data

Popis

Příkaz IMPORT SYLK přečte data z dokumentu *dokument*, Windows nebo Macintosh SYLK dokument, za vytvoří z něj nové záznamy pro tabulku *tabulka*.

Operace importu je prováděna přes platný vstupní formulář. Import čte jednotlivé pole a proměnné ve formuláři podle jejich vrstvení. Z tohoto důvodu musíte být velmi opatrní na pořadí textových objektů ve vrstvách. První objekt do kterého mají být data importována musí být na pozadí atd. Pokud je počet proměnných a polí ve formuláři nesouhlasí s počtem importovaných polí, jsou přebytečné ignorovány. Vstupní formulář použitý pro import nesmí obsahovat žádná tlačítka. Objekty podformuláře jsou ignorovány.

Poznámka: Jeden ze způsobů jak zajistit že budou data správně importována je označit objekt do kterého má být importováno první pole a přenést jej na popředí. Pokračujte v přesunování proměnných a polí na popředí v pořadí v jakém chcete importovat, ujistěte se že máte jedno pole nebo proměnnou pro každé importované pole.

Při uložení nového záznamu z importu je poslána událost Při potvrzení vstupu. Pokud při importu používáte proměnné, můžete využít tuto událost ke zkopírování dat z proměnných do polí.

Parametr *dokument* může obsahovat cestu s názvy složek. Pokud předáte prázdný řetězec, otevře se standardní okno systému Otevřít soubor. Pokud uživatel okno zruší, je zrušena celá operce importu a systémová proměnná OK je nastavena na 0.

Během importu je zobrazen teploměr. Uživatel může import přerušit stisknutím tlačítka s popisem Stop. Záznamy které již byli importovány však nebudou odstraněny. Pokud je import správně dokončen, je proměnná OK nastavena na 1. Pokud se objeví chyby nebo uživatel import přeruší je OK nastavena na 0. Teploměr může být skrytý pomocí příkazu MESSAGES OFF.

Operace importu je provedena s použitím výchozí ASCII mapy pro platformu na které probíhá, pokud tuto mapu nezměníte (pomocí příkazu USE ASCII MAP). ASCIImapa může být použita k převodu dat z jiné platformy která má jinou ASCII tabulku.

Příklad

Následující příklad importuje data ze SYLK dokumentu. Metoda nejdříve nastaví vstupní formulář, aby se data importovala do správného formuláře a pak provede import:

```
INPUT FORM([Lidé]; "Import")
IMPORT SYLK([Lidé];"NovíLidé") ` Import z dokumentu "NovíLidé"
```

Dále si přečtěte

EXPORT SYLK, IMPORT DIF, IMPORT TEXT, USE ASCII MAP.

Systémové proměnné a Sady

OK je nastavena na 1 pokud je import správně dokončen, jinak je nastavena na 0.

Příkazy a odkazy pro Import a Export

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

EXPORT SYLK (EXPORTOVAT FORMÁT SYLK)

Příkazy a odkazy pro Import a Export

Verze 3

EXPORT SYLK ({tabulka;} dokument)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ze které exportovat data nebo vychozi tabulka pokud vynecháno
dokument	Řetězec	→	SYLK dokument k přijetí dat

Popis

Příkaz EXPORT SYLK zapíše data z platného výběru záznamů platné tabulky, nebo tabulky *tabulka*, z platného procesu. Data jsou zapsána do *dokumentu*, Windows nebo Macintosh SYLK dokument.

Operace exportu je provedena přes platný výstupní formulář. Export dat je prováděn v pořadí vstupu dat výstupního formuláře. Z tohoto důvodu použijte výstupní formulář který obsahuje pouze pole nebo objekty které chcete exportovat. Na exportní formulář neumíťujte žádná tlačítka nebo jiné zvláštní objekty. Objekty podformuláře budou ignorovány.

Při dokončení exportu každého záznamu je poslána událost Při zavedení. Použijte tuto událost k nastavení proměnných které máte ve formuláři exportu.

Dokument může obsahovat název nového nebo existujícího dokumentu. Pokud má dokument stejný název jako existující dokument, je obsah existujícího dokumentu přepsán. Parametr může obsahovat cestu včetně názvů složek a disků. Pokud předáte prázdný řetězec, otevře se standardní okno Uložit soubor. Pokud uživatel toto okno zruší, je zrušen celý export a systémová proměnná OK je nastavena na 0.

Během exportu je zobrazen teploměr. Uživatel může kdykoli operaci přerušit stisknutím tlačítka s popisem Stop. Pokud je eport správně dokončen, je proměnná OK nastavena na 1. Jestliže je export přerušen uživatelem nebo se vyskytnou nějaké chyby, je proměnná OK nastavena na 0. Teploměr můžete skrýt použitím příkazu MESSAGES OFF.

Operace exportu je provedena s použitím výchozí ASCII mapy pro platformu na které probíhá. Pokud tuto mapu nezměníte (pomocí příkazu USE ASCII MAP). ASCII mapa může být použita k převodu dat z jiné platformy která má jinou ASCII tabulku.

Příklad

Tento příklad exportuje data do SYLK dokumentu. Metoda nejdříve nastaví výstupní formulář a pak vyexportuje záznamy:

```
OUTPUT FORM([Lidé]; "Export")
EXPORT SYLK([Lidé];"NovýLidé") ` Import do dokumentu "NovíLidé"
```

Dále si přečtěte

EXPORT DIF, EXPORT TEXT, IMPORT SYLK, USE ASCII MAP.

Systémové proměnné a Sady

OK je nastavena na 1 pokud je export správně dokončen, jinak je nastavena na 0.

Příkazy a odkazy pro Import a Export

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

IMPORT DIF (IMPORTOVAT FORMÁT DIF)

Příkazy a odkazy pro Import a Export

Verze 3

IMPORT DIF ({tabulka;} dokument)

Parametr	Typ	Popis
tabulka	Tabulka	Tabulka do které importovat data nebo výchozí tabulka pokud vynecháno
dokument	Řetězec	DIF dokument ze kterého importovat data

Popis

Příkaz IMPORT DIF přečte data z dokumentu, Windows nebo Macintosh DIF dokument, s vytvořením nových záznamů do tabulky *tabulka*.

Operace importu je prováděna přes platný vstupní formulář. Import čte jednotlivá pole a proměnné ve formuláři podle jejich vrstvení. Z tohoto důvodu musíte být velmi opatrní na pořadí textových objektů ve vrstvách. První objekt do kterého mají být data importována musí být na pozadí atd. Pokud je počet proměnných a polí ve formuláři nesouhlasí s počtem importovaných polí, jsou přebytečné ignorovány. Vstupní formulář použitý pro import nesmí obsahovat žádná tlačítka. Objekty podformuláře jsou ignorovány.

Poznámka: Jeden ze způsobů jak zajistit že budou data správně importována je označit objekt do kterého má být importováno první pole a přenést jej na popředí. Pokračujte v přesunování proměnných a polí na popředí v pořadí v jakém chcete importovat, ujistěte se že máte jedno pole nebo proměnnou pro každé importované pole.

Při uložení nového záznamu z importu je poslána událost Při potvrzení vstupu. Pokud při importu používáte proměnné, můžete využít tuto událost ke zkopírování dat z proměnných do polí.

Parametr *dokument* může obsahovat cestu s názvy složek. Pokud předáte prázdný řetězec, otevře se standardní okno systému Otevřít soubor. Pokud uživatel okno zruší, je zrušena celá operce importu a systémová proměnná OK je nastavena na 0.

Během importu je zobrazen teploměr. Uživatel může import přerušit stisknutím tlačítka s popisem Stop. Záznamy které již byli importovány však nebudou odstraněny. Pokud je import správně dokončen, je proměnná OK nastavena na 1. Pokud se objeví chyby nebo uživatel import přeruší je OK nastavena na 0. Teploměr může být skrytý pomocí příkazu MESSAGES OFF.

Operace importu je provedena s použitím výchozí ASCII mapy pro platformu na které probíhá, pokud tuto mapu nezměníte (pomocí příkazu USE ASCII MAP). ASCII mapa může být použita k převodu dat z jiné platformy která má jinou ASCII tabulku.

Příklad

Následující příklad importuje data z DIF dokumentu. Metoda nejdříve nastaví vstupní formulář, aby se data importovala do správného formuláře a pak provede import:

```
INPUT FORM([Lidé]; "Import")
IMPORT DIF([Lidé];"NovíLidé") ` Import z dokumentu "NovíLidé"
```

Dále si přečtěte

EXPORT DIF, IMPORT SYLK, IMPORT TEXT, USE ASCII MAP.

Systémové proměnné a Sady

OK je nastavena na 1 pokud je import správně dokončen, jinak je nastavena na 0.

Příkazy a odkazy pro Import a Export

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

EXPORT DIF (EXPORTOVAT FORMÁT DIF)

Příkazy a odkazy pro Import a Export

Verze 3

EXPORT DIF ({tabulka;} dokument)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ze které exportovat data nebo vychozi tabulka pokud vynecháno
dokument	Řetězec	→	DIF dokument k přijetí dat

Popis

Příkaz EXPORT DIF zapíše data z platného výběru záznamů tabulky *tabulka* z platného procesu. Data jsou zapsána do *dokumentu*, Windows nebo Macintosh DIF dokument.

Operace exportu je provedena přes platný výstupní formulář. Export dat je prováděn v pořadí vstupu dat výstupního formuláře. Z tohoto důvodu použijte výstupní formulář který obsahuje pouze pole nebo objekty které chcete exportovat. Na exportní formulář neumísťujte žádná tlačítka nebo jiné zvláštní objekty. Objekty podformuláře budou ignorovány.

Při dokončení exportu každého záznamu je poslána událost Při zavedení. Použijte tuto událost k nastavení proměnných které máte ve formuláři exportu.

Dokument může obsahovat název nového nebo existujícího dokumentu. Pokud má dokument stejný název jako existující dokumnet, je obsah existujícího dokumentu přepsán. Parametr může obsahovat cestu včetně názvů složek a disků. Pokud předáte prázdný řetězec, otevře se standardní okno Uložit soubor. Pokud uživatel toto okno zruší, je zrušen celý export a systémová proměnná OK je nastavena na 0.

Během exportu je zobrazen teploměr. Uživatel může kdykoli operaci přerušit stisknutím tlačítka s popisem Stop. Pokud je eport správně dokončen, je proměnná OK nastavena na 1. Jestliže je export přerušen uživatelem nebo se vyskytnou nějaké chyby, je proměnná OK nastavena na 0. Teploměr můžete skrýt s použitím příkazu MESSAGES OFF.

Operace exportu je provedena s použitím výchozí ASCII mapy pro platformu na které probíhá. Pokud tuto mapu nezměníte (pomocí příkazu USE ASCII MAP). ASCII mapa může být použita k převodu dat z jiné platformy která má jinou ASCII tabulku.

Příklad

Tento příklad exportuje data do DIF dokumentu. Metoda nejdříve nastaví výstupní formulář a pak vyexportuje záznamy:

```
OUTPUT FORM([Lidé]; "Export")
EXPORT DIF([Lidé];"NovýLidé") ` Import do dokumentu "NovíLidé"
```

Dále si přečtěte

EXPORT SYLK, EXPORT TEXT, IMPORT DIF, USE ASCII MAP.

Systémové proměnné a Sady

OK je nastavena na 1 pokud je export správně dokončen, jinak je nastavena na 0.

Příkazy a odkazy pro Import a Export

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

IMPORT DATA (IMPORTOVAT DATA)

Příkazy a odkazy pro Import a Export

Verze 6.5

IMPORT DATA (SouborNázev {; projekt{; *}})

Parametr	Typ		Popis
SouborNázev	Řetězec	→	Cesta a název souboru k importu
projekt	BLOB	→	Obsah projektu importu
		←	Nový obsah projektu importu (jestliže byla předána *)
*	*	→	Zobrazí dialogové okno pro import a obnoví projekt

Popis

Příkaz IMPORT DATA vám umožní importovat data umístěná v souboru *SouborNázev*. 4D může importovt v následujících formátech: Text, Text s pevnou délkou, SYLK, DIF, DBF (dBase) a 4th Dimension.

Pokud do parametru *SouborNázev* předáte prázdný řetězec, IMPORT DATA zobrazí standardní okno systému Otevřít soubor a umožní uživateli definovat název, typ a umístění importního souboru. Jakmile je dialogové okno potvrzeno, systémová proměnná *Document* bude obsahovat cestu k souboru. Pokud uživatel klepne na **Storno**, provádění příkazu je zastaveno a proměnná OK je nastavena na 0.

• Pokud nepředáte volitelný parametr *projekt*, zobrazí se dialogové okno importu. Uživatel může definovat parametry importu nebo načíst existující projekt.

Poznámka: *Projekt* importu obsahuje všechna nastavení pro import jako tabulku a pole do kterých importovat, oddělovače polí a záznamů, atd. Tyto parametry jsou ukázány v okně importu. Projekt může být uložen na disk pro další použití. Jestli chcete vědět více informací o okně importu, podívejte se do *Příručky uživatele 4th Dimension.*

• Pokud předáte BLOB obsahující projekt importu do parametru *projekt*, import proběhne bez dalších žádostí na potvrzení od uživatele.Projekt musí být vytvořen v okně importu a pak uložen. Máte dvě možnosti jak to udělat:

- Uložit projekt na disk, pak jej načíst s použitím DOCUMENT TO BLOB do pole nebo proměnné BLOB, a ten potom předáte jako parametr projekt.

- Použít IMPORT DATA s prázdným parametrem projekt a volitelným parametrem * a pak uložit parametr *projekt* do BLOB pole (čtěte dále). Tento způsob vám umožní uložit projekt v datovém souboru bez nutnosti načíst BLOB z disku.

Zadaný volitelný parametr * vynutí zobrazení dialogu importu s parametry uloženými v projektu. Tato možnost vám umožní použít předdefinovaný projekt se stálou možností upravit některá nastavení. Mimoto parametr *projekt* obsahuje po uzavření dialogu importu, parametry nového projektu. Pak můžete uložit nový projekt do BLOB pole, na disk, atd.

Pokud byl import správně dokončen, je proměnná OK nastavena na 1.

Dále si přečtěte

EXPORT DATA, IMPORT DIF, IMPORT SYLK, IMPORT TEXT.

Systémové proměnné a Sady

Pokud uživatel klepne na tlačítko **Storno** v okně uložit soubor nebo v okně Importu, je proměnná OK nastavena na 0. Pokud byl import správně dokončen, je proměnná OK nastavena na 1.

Příkazy a odkazy pro Import a Export

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

EXPORT DATA (EXPORTOVAT DATA)

Příkazy a odkazy pro Import a Export

Verze 6.5

EXPORT DATA (SouborNázev {; projekt{; *}})

Parametr	Typ		Popis
SouborNázev	Řetězec	→	Cesta a název souboru k exportu
projekt	BLOB	→	Obsah projektu exportu
		←	Nový obsah projektu exportu (jestliže byla předána *)
*	*	→	Zobrazí dialogové okno pro export a obnoví projekt

Popis

Příkaz EXPORT DATA vám umožní exportovat data do souboru *SouborNázev*. 4D může exportovat do následujících formátů: Text, Text s pevnou délkou, SYLK, DIF, DBF (dBase) a 4th Dimension.

Pokud do parametru *SouborNázev* předáte prázdný řetězec, EXPORT DATA zobrazí standardní okno systému Uložit soubor a umožní uživateli definovat název, typ a umístění exportního souboru. Jakmile je dialogové okno potvrzeno, systémová proměnná *Document* bude obsahovat cestu k souboru. Pokud uživatel klepne na **Storno**, provádění příkazu je zastaveno a proměnná OK je nastavena na 0.

• Pokud nepředáte volitelný parametr *projekt*, zobrazí se dialogové okno exportu. Uživatel může definovat parametry exportu nebo načíst existující projekt.

Poznámka: Projekt exportu obsahuje všechna nastavení pro export jako tabulku a pole ze kterých exportovat, oddělovače polí a záznamů, atd. Tyto parametry jsou ukázány v okně exportu. Projekt může být uložen na disk pro další použití. Jestli chcete vědět více informací o okně exportu, podívejte se do *Příručky uživatele 4th Dimension.*

• Pokud předáte BLOB obsahující projekt exportu do parametru *projekt*, export proběhne bez dalších žádostí na potvrzení od uživatele.Projekt musí být vytvořen v okně exportu a pak uložen. Máte dvě možnosti jak to udělat:

- Uložit projekt na disk, pak jej načíst s použitím DOCUMENT TO BLOB do pole nebo proměnné BLOB, který potom předáte jako parametr *projekt*.

- Použít EXPORT DATA s prázdným parametrem *projekt* a volitelným parametrem * a pak uložit parametr *projekt* do BLOB pole (čtěte dále). Tento způsob vám umožní uložit projekt v datovém souboru bez nutnosti načíst BLOB z disku.

Zadaný volitelný parametr * vynutí zobrazení dialogu exportu s parametry uloženými v projektu. Tato možnost vám umožní použít předdefinovaný projekt se stálou možností upravit některá nastavení. Mimoto parametr *projekt* obsahuje i po uzavření dialogu exportu, parametry nového projektu. Pak můžete uložit nový projekt do BLOB pole, na disk, atd.

Pokud byl export správně dokončen, je proměnná OK nastavena na 1.

Dále si přečtěte

EXPORT DIF, EXPORT SYLK, EXPORT TEXT, IMPORT DATA.

Systémové proměnné a Sady

Pokud uživatel klepne na tlačítko **Storno** v okně uložit soubor nebo v okně Exportu, je proměnná OK nastavena na 0. Pokud byl export správně dokončen, je proměnná OK nastavena na 1.

Příkazy a odkazy pro Import a Export

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ON EVENT CALL (Při události provést)

Příkazy a odkazy pro Přerušení

Verze 3

ON EVENT CALL (MetodaUdálostí{; NázevProcesu})

Parametr	Typ		Popis
MetodaUdálostí	Řetězec	→	Metoda, která bude volána při událostech, nebo prázdný řetězec "" pro zastavení odchytávání událostí
NázevProcesu	řetězec	→	Název procesu

Popis

Příkaz ON EVENT CALL instaluje metodu jejíž název umístíte do parametru *MetodaUdálostí* jako metodu pro zachytávání událostí. Tato metoda se nazývá **metoda ovládání událostí** nebo **metoda zachytávání událostí**.

Tip: Tento příkaz již vyžaduje rozšířené znalosti programování. Pro práci s událostmi většinou nebudete potřebovat příkaz ON EVENT CALL. Při používání formulářů ovládá sama 4th Dimension posílání událostí k formuláři nebo objektu.

Tip: Ve verzi 6 jsou nové příkazy jako GET MOUSE, Shift down, atd.. pro získávání informací o událostech. Tyto příkazy mohou být volány z metody objektu pro získání informací o událostech týkajících se objektu. Jejich použití vám ušetří psaní algoritmu založeného na schématu ON EVENT CALL.

Rozsah tohoto příkazu je platná pracovní sekce. Jako výchozí je metoda spuštěna v odděleném místním procesu. Najednou můžete mít pouze jednu metodu ovládající události. Pokud chcete zastavit odchytávání událostí touto metodou, proveďte znovu příkaz ON EVENT CALL a do parametru *MetodaUdálostí* vložte prázdný řetězec.

Metoda na ovládání událostí je spuštěna v odděleném procesu a je stále aktivní, i když neběží žádná jiná metoda 4th Dimension. Po instalaci této metody, 4th Dimension zavolá tuto metodu při každé události která se nastane. Událostmi jsou klepnutí na myš nebo úhoz na klávesu.

Volitelný parametr *NázevProcesu* pojmenuje proces vytvořený příkazem ON EVENT CALL. Pokud tento název začíná znakem $, je zapnut místní proces, což je většinou to co chcete. Pokud tento parametr vynecháte, 4th Dimension pojmenuje nový proces jako $Event Manager.

UPOZORNĚNÍ: Buďte opatrní na to co provádíte v metodě zachytávání událostí. NEPROVÁDĚJTE v ní příkazy které vytvářejí události, jinak může být velmi složité opustit provádění této metody. Kombinace kláves Ctrl+Shift+Backspace (Windows) nebo Command+Shift+Option+Control+Backspace (Macintosh) převede proces Event manager na normální proces. To znamená že metoda již nebude automaticky předávat všechny události které se stanou. Můžete tuto techniku použít k obnovení chybného zachytávání událostí (tam kde jste v této metodě zachytávání udělali chybu).

V metodě zachytávání událostí můžete číst následující systémové proměnné - *MouseDown, KeyCode, Modifiers, MouseX, MouseY* a *MouseProc*. Všimněte si, že proměnné jsou proměnné procesu. Jejich rozsah je proto proces zachytávání událostí. Pokud chcete jejich hodnoty použít v jiných procesech zkopírujte je do meziprocesních proměnných.

• Hodnota *MouseDown* je nastavena na 1 pokud událost byla klepnutí na tlačítko myši a na 0 pokud ne.

• *KeyCode* je nastavena na hodnotu ASCII pro klávesu která byla stisknuta. Tato proměnná vrací ASCII kód nebo kód klávesy funkce. Tyto kódy jsou zobrazeny v kapitolách ASCII kódy a Kódy kláves funkcí. 4D obsahuje konstanty pro hlavní ASCII kódy a kódy kláves funkcí. V okně Průzkumníka se můžete podívat na skupiny těchto konstant.

• Systémová proměnná *Modifiers* obsahuje hodnoty modifikátorů. Zaznamenává, jestli byly stisknuty následující klávesy:

Platforma	*Klávesa*
Windows	*Shift, Caps Lock, Alt, Ctrl, Pravé tlačítko myši*
Macintosh	*Shift, Caps Lock, Option, Command, Control*

Poznámky

- Klávesy ALT na Windows je stejná jako Option na Macintoshi
- Klávesy Ctrl na Windows je stejná jako Command na Macintoshi
- Klávesa Control na Macintoshi nemá žádný ekvivalent na Windows. Nicméně pravé tlačítko myši na Windows má stejnou funkci jako Control- klepnutí na Macintoshi.

Modifikátory nevytvářejí žádné události; musí být stisknuta další jiná klávesa nebo tlačítko myši. Proměnná *Modifiers* je 4-bytová Long Integer hodnota, na kterou může být nahlíženo jako na Array z 32 bitů. 4D obsahuje předdefinované konstanty přesně vyjadřující pozici bitů nebo masku bitů k testování každé klávesy modifikátorů. Pokud například chcete zjistit jestli byla stisknuta klávesa Shift, můžete napsat:

```
If (Modifiers ?? Shift key bit ) ` Jestliže je stisknuta klávesa Shift
```

nebo:

```
If ((Modifiers & Shift key mask)#0)` Jestliže je stisknuta klávesa Shift
```

• Systémové proměnné *MouseX* a *MouseY* obsahují vodorovnou a svislou pozici klepnutí myši. Tato pozice je vyjádřena v lokálních souřadnicích vzhledem k oknu ve kterém bylo klepnutí provedeno. Horní levý okraj okna je pozice 0,0. Jsou naplněné pouze když bylo provedeno klepnutí.

• Systémová proměnná *MouseProc* obsahuje číslo procesu ve kterém bylo klepnutí provedeno.

Důležité: Systémové proměnné *MouseDown, KeyCode, MouseX, MouseY* a *MouseProc* obsahují platné hodnoty pouze v metodě zachytávání událostí instalované pomocí ON EVENT CALL.

Příklad

Tento příklad přeruší tisk pokud uživatel stiskne klávesy Ctrl+Tečka. Nejdříve je instalována metoda pro zachytávání událostí. Pak zobrazí zprávu že uživatel může zrušit tisk stisknutím této kombinace kláves. Pokud je meziprocesní proměnná <>vbZastav nastavena na True v metodě zachytávající události, je uživatel informován o počtu záznamů které byli vytištěny. Pak je metoda zachytávání událostí odstraněna:

```
PAGE SETUP
If (OK=1)
   <>vbZastav:=False
   ON EVENT CALL("OVLADAČ ZACHYTAVANI") ` Nainstaluje zachytávací metodu
   ALL RECORDS([Lidé])
   MESSAGE("K přerušení tisku stiskněte Ctrl-Tečka.")
   $vlPocZazn:=Records in selection([Lidé])
   For ($vlZaznam;1;$vlPocZazn)
      If (<>vbZastav)
         ALERT("Tisk zrušen na záznamu "+String($vlZaznam)+" z "+String($vlPocZazn))
         $vlZaznam:=$vlPocZazn+1
      Else
         PRINT FORM([Lidé];"Special Report")
      End if
   End for
   PAGE BREAK
   ON EVENT CALL("") ` Odinstaluje metodu zachytávání
End if
```

Pokud bylo stisknuto Ctrl+Tečka, metoda zachytávání událostí nastaví <>vbZastav na True:

```
    ` Metoda projektu OVLADAČ ZACHYTAVANI
If ((Modifiers ?? Command key bit) & (KeyCode = Period))
   CONFIRM("Jste si jistý?")
   If (OK=1)
      <>vbZastav:=True
      FILTER EVENT ` NEZAPOMEŇTE toto provést, jinak rovněž 4D obdrží tuto událost
   End if
End if
```

Všiměte si že tento příklad používá ON EVENT CALL, protože provádí specielní tiskovou zprávu s použitím příkazů PAGE SETUP, PRINT FORM a PAGE BREAK se smyčkou For..end for.

Pokud tisknete zprávu pomocí PRINT SELECTION , NEPOTŘEBUJETE ovládat události pomocí kterých uživatel přeruší tisk: PRINT SELECTION to pro vás udělá.

Dále si přečtěte

FILTER EVENT, GET MOUSE, Shift down.

Příkazy a odkazy pro Přerušení

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

FILTER EVENT (ZACHYTIT UDÁLOST)

Příkazy a odkazy pro Přerušení

Verze 3

FILTER EVENT

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz FILTER EVENT voláte z metody pro zachytávání událostí která byla instalovaná pomocí ON EVENT CALL.

Pokud metoda zychytávající události provede příkaz FILTER EVENT, není událost předána stroji 4D.

Tento příkaz umožní odstranit platnou událost z fronty událostí a tak 4D neprovede žádné další reakce na tuto událost.

UPOZORNĚNÍ: Vyhněte se nainstalování metody pro zachytávání událostí která bude obsahovat pouze příkaz FILTER EVENT, protože v tomto případě budou ignorovány všechny události které provedete. V případě že budete mít již takovouto metodu pro zachytávání událostí nainstalovanou, použijte klávesovou zkratku Ctrl+Shift+Backspace (Windows) nebo Command+Shift+Option+Control+Backspace (Macintosh). Tato zkratka převede proces zachytávající události na normální proces, který nebude zachytávat události.

Příklad

Podívejte se na příklad u příkazu ON EVENT CALL.

Dále si přečtěte

ON EVENT CALL.

Příkazy a odkazy pro Přerušení

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ON ERR CALL (Při chybě provést)

Příkazy a odkazy pro Přerušení

Verze 3

ON ERR CALL (MetodaChyby)

Parametr	Typ		Popis
MetodaChyby	Řetězec	→	Název metody, která má být volána při chybách, nebo prázdný řetězec "" pro zastavení zachytávání chyb

Popis

Příkaz ON ERR CALL nainstaluje metodu jejíž název předáte do parametru *MetodaChyby* jako metodu pro zachytávání chyb. Tato metoda projektu se nazývá **metoda ovládání chyb** nebo **metoda zachytávání chyb.**

Rozsah této metody je platný proces. Pro jeden proces můžete mít pouze jednu metodu zachytávající chyby, ale můžete mít více metod pro zachytávání chyb pro více procesů.

K zastavení zachytávání chyb proveďte znovu příkaz ON ERR CALL a do parametru *MetodaChyby* vložte prázdný řetězec.

Jakmile je metoda zachytávající chyby nainstalovaná, 4th Dimension ji zavolá pokaždé když vznikne nějaká chyba.

Chyby můžete identifikovat čtením systémové proměnné *Error*, která obsahuje číslo chyby. Seznam Kódů chyb je v kapitole Kódy chyb. Jestli chcete vědět více informací, přečtěte si části Chyby syntaxe a Chyby databázového stroje. Hodnota proměnné *Error* je důležitá pouze v metodě zachytávání chyb; pokud potřebujete číslo chyby v metodě která chybu způsobila, zkopírujte proměnnou *Error* do vaší vlastní proměnné procesu.

Metoda zachytávání chyb může řídit chyby svým způsobem nebo poslat upozornění uživateli. Chyby můžou vzniknout v:

- motoru 4th Dimension, například při ukládání záznamu se pokusíte duplikovat jedinečný indexovaný klíč.
- prostředí 4th Dimension: například pokud nemáte dostatek paměti k vytvoření array.
- operačním systému na kterém databáze pracuje: například je plný disk.

Příkaz ABORT může být použit pro přerušení průběhu. Pokud neprovedete příkaz ABORT v metodě zachytávající události, 4th Dimension se vrátí k přerušené metodě a bude pokračovat v jejím provádění. Použijte příkaz ABORT pouze v případě kdy nemůže být chyba opravena.

Pokud se stane chyba v samotné zachytávací metodě, 4th Dimension skončí zachytávání. Proto je důležité aby jste se ujistili že metoda zachytávání událostí nemůže způsobit žádnou chybu. Nemůžete také použít ON ERR CALL v metodě zachytávající události.

Jakmile je metoda pro zachytávání chyb nainstalovaná, je možné tuto metodu krokovat pomocí Alt+Klepnutí (Windows) nebo Option+Klepnutí (Macintosh). Je to z toho důvodu, že Alt+Klepnutí nebo Option+Klepnutí generuje chybu (kód 1006), která aktivuje metodu pro zachytávání událostí. Tento kód chyby můžete testovat s použitím příkazu TRACE.

Příklady

1. Následující metoda projektu se pokusí vytvořit dokument jehož název je zadán jako parametr. Pokud nemůže být dokument vytvořen, metoda projektu vrátí 0 nebo kód chyby:

```
   ` Metoda projektu Vytvořit dok
   ` Vytvořit dok ( Řetězec ; Ukazatel ) → Long Integer
   ` Vytvořit dok ( NázevDok ; →DocRef ) → Výsledný kód chyby
gError:=0
```

```
ON ERR CALL("OVLADAC CHYB IO")
$2→:=Create document($1)
ON ERR CALL("")
$0:=gError
```

Metoda OVLADAC CHYB IO je zobrazena zde:

```
    ` Metoda projektu OVLADAC CHYB IO
gError:=Error    ` Pouze zkopírovat kód chyby do proměnné procesu.
```

Všiměte si že použití proměnné procesu gError k získání kódu chyby v právě prováděné metodě. Jakmile jsou tyto metody předány do databáze. můžete napsat:

```
    ` ...
C_TIME(vhDokumRef)
$vlKodChyby:= Vytvořit dok ($vsNazevDokument;→vhDokumRef)
If ($vlKodChyby=0)
        `...
    CLOSE DOCUMENT($vlKodChyby)
Else
    ALERT ("Dokument nemůže být vytvořen, I/O chyba "+String($vlKodChyby))
End if
```

2. Podívejte se na příklady u částí Array a Paměť.

3. Když zavádíte složité návazné operace, můžete pracovat s různými podprogramy, které vyžadují odlišné zachytávání chyb. Vzhledem k tomu že můžete mít pouze jednu spuštěnou metodu zachytávání chyb na proces, máte dvě možnosti:

- udržujte povědomí odkud byl příkaz volán pokaždé když provedete ON ERR CALL, nebo
- použijte procesní array proměnných (v tomto případě asErrorMethod) k naplnění metod zachytávání chyb a metod projektu (v tomto případě ON ERROR CALL) k instalování a odinstalování různých metod zachytávajících chyby.

Na úplném začátku procesu musíte nastavit array:

```
ARRAY STRING (63;asErrorMethod)
```

Zde je metoda ON ERROR CALL:

```
    ` Metoda projektu ON ERROR CALL
    ` ON ERROR CALL    {( Řetězec ) }
    ` ON ERROR CALL    {( Název metody )}
C_STRING(63;$1;$ErrorMethod)
C_LONGINT($vlElem)
If (Count parameters>0)
    $ErrorMethod:=$1
Else
    $ErrorMethod:=""
End if
If ($ErrorMethod#"")
    C_LONGINT(gError)
    gError:=0
    $vlElem:=1+Size of array(asErrorMethod)
    INSERT ELEMENT(asErrorMethod;$vlElem)
```

```
    asErrorMethod{$vlElem}:=$1
    ON ERR CALL($1)
Else
    ON ERR CALL("")
    $vlElem:=Size of array(asErrorMethod)
    If ($vlElem>0)
        DELETE ELEMENT(asErrorMethod;$vlElem)
        If ($vlElem>1)
            ON ERR CALL(asErrorMethod{$vlElem-1})
        End if
    End if
End if
```

Pak tuto metodu můžete volat následovně:

```
gError:=0
ON ERROR CALL("IO CHYBY") ` Instaluje metodu zachytávání chyb IO CHYBY
   ` ...
ON ERROR CALL("VSECHNY CHYBY") ` Instaluje metodu zachytávání chyb VSECHNY CHYBY
   ` ...
ON ERROR CALL ` odinstaluje VSECHNY CHYBY a nainstaluje IO CHYBY
   ` ...
ON ERROR CALL ` Odinstaluje metodu zachytávání chyb IO CHYBY
   ` ...
```

4. Následující metoda zachytávání chyb ignoruje uživatelská přerušení:

```
` Metoda projektu UKAZ JEN CHYBY
If (Error#1006)
   ALERT ("Vznikla chyba "+String(Error)+".")
End if
```

Dále si přečtěte

ABORT.

Příkazy a odkazy pro Přerušení

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ABORT (PŘERUŠIT)

Příkazy a odkazy pro Přerušení

Verze 3

Poznámka: Tento příkaz budete používat vyjímečně.

ABORT

Parametr	Typ	Popis
Tento příkaz nevyžaduje žádné parametry.		

Popis

Příkaz ABORT je k použití v metodě zachytávající chyby, která byla instalovaná pomocí příkazu ON ERR CALL.

Pokud vznikne chyba v době v době kdy nemáte instalovánu metodu zachytávání, zobrazí 4D své standardní dialogové okno hlášení chyby a přeruší provádění metody, kde chyba vznikla. Pokud je prováděná metoda:

- Metoda objektu, formuláře (nebo metoda projektu volaná jednou z těchto metod), navrátíte se do formuláře.
- Metoda volaná z nabídky, jste navráceni k záhlaví nabídek nebo do platného zobrazeného formuláře.
- Metoda řídící proces, je skončen proces.
- Metoda přímo nebo nepřímo volaná při exportu nebo importu, je skončena prováděná operace. To samé platí pro hledání a třídění.
- Atd.

Pokud máte instalovanou metodu pro zachytávání chyb, 4D nikdy nezobrazí své standardní dialogové okno ani nepřeruší provádění kódu, ale místo toho zavolá metodu pro zachytávání chyb a navrátí se na další řádek metody a bude pokračovat v provádění kódu.

Existují chyby, které můžete zachytit a po jejich zachycení je i opravit a pokračovat bez problémů v provádění další části metody. Nicméně existují i chyby které nemůžete zpracovat a chyby které sice rozeznáte, ale nemůžete opravit. V těchto případech je třeba přerušit provádění kódu pomocí příkazu ABORT.

Historická poznámka

Ačkoli je příkaz ABORT vytvořen pro provádění v metodě zachytávající chyby, někteří členové komunity 4D jej stále používají k přerušení provádění kódu v normálních metodách projektu. Fakt že to funguje je pouze dočasný. Nedoporučujeme používat tento příkaz jinde než v metodách zachycujících chyby.

Příkazy a odkazy pro Přerušení

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Count parameters (Počet parametrů)

Příkazy a odkazy pro Jazyk

Verze 3

Count parameters → Číslo

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz Count parameters vrací počet parametrů předávaných metodě projektu.

UPOZORNĚNÍ: Count parameters je zamýšlen pouze pro metody projektu které jsou volané jinou metodou (metodou projektu nebo jinou). Jestliže je použít Count parameters v metodě volané z nabídky, vrátí příkaz 0.

Příklady

1. Metody projektu 4th Dimension přijímají volitelné parametry začínaje zleva. Například můžete volat metodu MaMetoda(a;b;c;d) následujícími způsoby:

MaMetoda(a ; b ; c ; d) ` Všechny parametry jsou přiřazeny
MaMetoda(a ; b ; c) ` Poslední parametr není přiřazen
MaMetoda(a ; b) ` Poslední dva parametry nejsou přiřazeny
MaMetoda(a) ` Pouze první parametr je přiřazen
MaMetoda` Není přiřazen žádný parametr

Count parameters v metodě MaMetodamůže detekovat aktuální počet přiřazených parametrů a podle vráceného čísla lze provést různé akce. Následující příklad zobrazí textovou zprávu a může vložit text do oblasti 4D Write nebo poslat text do dokumentu na disku:

```
   ` Metoda projektu PRIPOJ TEXT
   ` PRIPOJ TEXT ( Text    {; Long    {; Čas}}     )
   ` PRIPOJ TEXT ( Text    {; 4D Write oblast    {; DocRef }}    )
C_TEXT ($1)
C_TIME ($2)
C_LONGINT ($3)
MESSAGE ($1)
If (Count parameters>=3)
   SEND PACKET ($3;$1)
Else
   If (Count parameters>=2)
      WR INSERT TEXT ($2;$1)
   End if
End if
```

Po uložení této metody do vaší aplikace můžete napsat:

PRIPOJ TEXT (vtNejakyText) ` Pouze zobrazí zprávu
PRIPOJ TEXT (vtNejakyText;$wrArea) ` Zobrazí zprávu a připojí text do $wrArea
PRIPOJ TEXT (vtNejakyText;0;$vhDokumRef) ` Zobrazí zprávu a zapíše jí do $vhDokumRef

2. Metody projektu 4th Dimension mohou přijímat proměnný počet parametrů stejného typu začínaje zleva. K deklaraci těchto proměnných použijte příkazy Compileru, kterým předáte ${N} jako proměnnou kde N definuje

první parametr. Pak můžete použít Count parameters, adresovat tyto parametry nepřímým odkazem ve smyčce For..End for. Tento příklad je funkce která vrací největší číslo z předaných parametrů:

```
   ` Metoda projektu Max z
   ` Max z ( Real    {; Real2... ; RealN}    ) → Real
   ` Max z ( Value    {; Value2... ; ValueN}    ) → Největší hodnota
C_REAL ($0;${1}) ` Všechny parametry jsou typu REAL stejně jako výsledek
$0:=${1}
For ($vlParam;2; Count parameters)
   If (${$vlParam}>$0)
      $0:=${$vlParam}
   End if
End for
```

Po uložení této metody do vaší aplikace můžete napsat:

vrResult:= *Max z* (**Records in set**("Akce A");**Records in set**("Akce B"))

nebo:

vrResult:= *Max z* (r1; r2; r31; r4; r5; r6;)

Dále si přečtěte

Příkazy kompilátoru, C_BLOB, C_BOOLEAN, C_DATE, C_GRAPH, C_INTEGER, C_LONGINT, C_PICTURE, C_POINTER, C_REAL, C_STRING, C_TEXT, C_TIME.

Příkazy a odkazy pro Jazyk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Type (Typ dat)

Příkazy a odkazy pro Jazyk

Verze 6.0 (Upraveno)

Type (PoleProm) → Číslo

Parametr	Typ		Popis
poleProm	Pole | Proměnná	→	Pole nebo proměnná k testování
Výsledek funkce	Číslo	←	Číslo typu dat

Popis

Příkaz Type vrací číselnou hodnotu která značí typ dat pole nebo proměnné které zadáte do parametru poleProm.

4th Dimension obsahuje následující předdefinované konstanty:

Konstanta	***Typ***	***Hodnota***
Is Alpha Field	*Long Integer*	*0*
Is StringVar	*Long Integer*	*24*
Is Text	*Long Integer*	*2*
Is Real	*Long Integer*	*1*
Is Integer	*Long Integer*	*8*
Is LongInt	*Long Integer*	*9*
Is Date	*Long Integer*	*4*
Is Time	*Long Integer*	*11*
Is Boolean	*Long Integer*	*6*
Is Picture	*Long Integer*	*3*
Is SubTabulka	*Long Integer*	*7*
Is BLOB	*Long Integer*	*30*
Is Undefined	*Long Integer*	*5*
Is Pointer	*Long Integer*	*23*
String array	*Long Integer*	*21*
Text array	*Long Integer*	*18*
Real array	*Long Integer*	*14*
Integer array	*Long Integer*	*15*
LongInt array	*Long Integer*	*16*
Date array	*Long Integer*	*17*
Boolean array	*Long Integer*	*22*
Picture array	*Long Integer*	*19*
Pointer array	*Long Integer*	*20*
Array 2D	*Long Integer*	*13*

Poznámka kompatibility: V předchozí verzi 4D, Type vracel 3 (typ Obrázek) pokud byl použit na proměnnou Diagram vytvořenou pomocí C_GRAPH. Od verze 6 vrací 9 (typ Longint) pokud je použit na proměnnou Diagram.

Type můžete použít na Pole, meziprocesní proměnnou, procesní proměnnou, místní proměnnu a dereferencované ukazatele k těmto typům objektů.

Poznámka verze 6: Od verze 6 můžete použít Type na parametry ($1, $2,... ${...}) nebo na výsledek funkce ($0).

Příklady

1. Podívejte se na příklad u příkazu APPEND TO CLIPBOARD.

2. Podívejte se na příklad u příkazu DRAG AND DROP PROPERTIES.

3. Následující metoda projektu vyprázdní některá nebo všechna pole pro platný záznam tabulky, jejíž ukazatel je předán jako parametr. Provede to bez vymazání záznamu a bez změny ukazatele na platný záznam:

```
    ` Metoda projektu PRAZDNY ZAZNAM
    ` PRAZDNY ZAZNAM ( Ukazatel {; Long}    )
    ` PRAZDNY ZAZNAM ( → [Tabulka]    {; Typ Flag}    )
C_POINTER ($1)
C_LONGINT ($2;$vlTypeFlags)
If (Count parameters>=2)
   $vlTypeFlags:=$2
Else
   $vlTypeFlags:=0xFFFFFFFF
End if
For ($vlPole;1;Count fields($1))
   $vpPole:=Field(Table($1);$vlPole)
   $vlPoleTyp:=Type($vpPole→)
   If ( $vlTypeFlags ?? $vlPoleTyp )
      Case of
         ÷ (($vlPoleTyp=Is Alpha Field)|($vlPoleTyp=Is Text))
            $vpPole→:=""
         ÷ (($vlPoleTyp=Is Real)|($vlPoleTyp=Is Integer)|($vlPoleTyp=Is LongInt))
            $vpPole→:=0
         ÷ ($vlPoleTyp=Is Date)
            $vpPole→:=!00/00/00!
         ÷ ($vlPoleTyp=Is Time)
            $vpPole→:=?00:00:00?
         ÷ ($vlPoleTyp=Is Boolean)
            $vpPole→:=False
         ÷ ($vlPoleTyp=Is Picture)
            C_PICTURE($vgEmptyPicture)
            $vpPole→:=$vgEmptyPicture
         ÷ ($vlPoleTyp=Is SubTabulka)
            Repeat
               ALL SUBRECORDS($vpPole→)
               DELETE SUBRECORD($vpPole→)
            Until(Records in subselection($vpPole→)=0)
         ÷ ($vlPoleTyp=Is BLOB)
            SET BLOB SIZE($vpPole→;0)
      End case
   End if
End for
```

Po uložení této metody do databáze, můžete napsat následující:

```
    ` Vyprázdnit celý záznam tabulky [Něco práce]
PRAZDNY ZAZNAM (→[Něco práce])
    ` Vyprázdnit Text, BLOB a Obrázek pole pro platný záznam
    ` tabulky [Něco práce]
PRAZDNY ZAZNAM (→[Něco práce]; 0 ?+ Is Text ?+ Is BLOB ?+ Is Picture )
    ` Vyprázdnit celý platný záznam tabulky [Něco práce]
    ` Mimo Alfa polí
```

PRAZDNY ZAZNAM (→[Něco práce]; -1 ?- Is Alpha Field)

Dále si přečtěte

Is a variable, Undefined.

Příkazy a odkazy pro Jazyk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Self (Samoukazatel)

Příkazy a odkazy pro Jazyk

Verze 3

Self → Ukazatel

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Ukazatel	←	Ukazatel na objekt formuláře (je-li), jehož metoda je právě prováděna. Jinak prázdný ukazatel, Nil (→>[]), je-li mimo kontext

Popis

Příkaz Self vrací ukazatel na objekt jehož metoda je právě prováděna.

Self se používá k odkazu proměnné v její vlastní metodě objektu. Vrací platný ukazatel pouze v případě že je prováděn z metody objektu. Nemůže být použit v metodě projektu, pokud není tato metoda volána metodou objektu. Pokud je Self volán mimo kontext, vrací Nulový ukazatel.

Tip: Self je použitelný, jestliže pro mnoho objektů ve formuláři potřebuje provést stejnou akci, která je prováděna na těch samých objektech.

Příklad

Podívejte se na příklad u příkazu RESOLVE POINTER.

Dále si přečtěte

RESOLVE POINTER.

Příkazy a odkazy pro Jazyk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

RESOLVE POINTER (UKAZUJE NA)

Příkazy a odkazy pro Jazyk

Verze 6.0

RESOLVE POINTER (ukazatel; PromNazev; tabČíslo; poleČíslo)

Parametr	**Typ**		**Popis**
ukazatel	Ukazatel	→	ukazatel pro nějž chceme zjistit objekt dat
PromNázev	Řetězec	←	Název odkazované proměnné nebo array nebo prázdný znak, není-li to proměnná nebo array
tabČíslo	Číslo	←	číslo referencované tabulky nebo prvku array
poleČíslo	Číslo	←	číslo referencovaného pole nebo 0 není-li pole

Popis

Příkaz RESOLVE POINTER obdrží informace o objektu na kterém je ukazatel a uloží je do proměnných *PromNázev*, *tabČíslo* a *poleČíslo*.

Podle pedaného objektu vrátí RESOLVE POINTER následující hodnoty

Odkazovaný objekt	***Parametry***		
	PromNázev	***tabČíslo***	***poleČíslo***
Žádný	*"" (prázdný řetězec)*	*0*	*0*
Proměnná	*Název proměnné*	*-1*	*0*
Array	*Název array*	*-1*	*0*
Prvek array	*Název array*	*Číslo prvku*	*0*
Tabulka	*"" (prázdný řetězec)*	*Číslo tabulky*	*0*
Pole	*"" (prázdný řetězec)*	*Číslo tabulky*	*Číslo pole*

Poznámka: Pokud hodnota kterou předáte do ukazatele nebude ukazatel, vrátí se vám chyba syntaxe.

Příklady

1. Ve formuláři vytvoříte 100 dostupných proměnných s názvy V1, V2,...V100. Můžete to udělat následovně:
a. Vytvořte jednu dostupnou proměnnou s názvem v.
b.Nastavte její vlastnosti.
c.Přiřaďte následující metodu k objektu:

```
   DelatNeco (Self) ` DelatNeco je metoda projektu ve vaší databázi
```

d. V tomto okamžiku můžete duplikovat proměnnou kolikrát chcete nebo použijte Objekty na síti v Editoru formulářů.
e. V metodě DelatNeco, pokud potřebujete znát ukazatel na objektu pro který je metoda volána, napište:

```
RESOLVE POINTER($1;$vsPromNazev;$vlTabulkaNum;$vlPoleNum)
$vlVarNum:=Num(Substring($vsPromNazev;2))
```

Všiměte si že při vytváření formuláře jste napsali metodu pro 100 proměnných najednou a nepotřebujete psát stále DelatNeco(1), DelatNeco(2), ... DelatNeco(100).

2. Z důvodů ladění, potřebujete ověřit jestli druhý parametr ($2) pro metodu je ukazatel na tabulku. Na začátku této metody napište:

```
   ` ...
If (<>DebugOn)
```

```
    RESOLVE POINTER($2;$vsPromNazev;$vlTabulkaNum;$vlPoleNum)
    If (Not(($vlTabulkaNum>0)&($vlPoleNum=0)&($vsPromNazev="")))
        ` UPOZORNĚNÍ: Ukazatel není odkaz na tabulku
      TRACE
    End if
  End if
    ` ...
```

3. Přečtěte si příklad u příkazu DRAG AND DROP PROPERTIES.

Dále si přečtěte

DRAG AND DROP PROPERTIES, Field, Get pointer, Is a variable, Nil, Table.

Příkazy a odkazy pro Jazyk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Nil (Prázdný)

Příkazy a odkazy pro Jazyk

Verze 3

Nil (Ukazatel) → Logické

Parametr	Typ		Popis
Ukazatel	Ukazatel	→	Ukazatel na testování
Výsledek funkce	Logické	←	True = Prázdný ukazatel FALSE = Ukazatel na existující objekt

Popis

Příkaz Nil vrací True pokud testovaný ukazatel je prázdný Nil (→[]). Vrací False ve všech ostatních případech (ukazatel na pole, proměnnou, atd)

Od verze 6 je výhodnější místo použití Nil použít příkaz RESOLVE POINTER, který vám vrátí informace o odkazovaném objektu, bez rozdílu jaký je to objekt (včetně prázdného ukazatele).

Dále si přečtěte

Is a variable, RESOLVE POINTER.

Příkazy a odkazy pro Jazyk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Is a variable (Je proměnnou)

Verze 3

Is a variable (Ukazatel) → Logické

Parametr	Typ		Popis
Ukazatel	Ukazatel	→	Ukazatel na testování
Výsledek funkce	Logické	←	TRUE = Ukazatel ukazuje k proměnné FALSE = Ukazatel neukazuje k proměnné

Popis

Příkaz Is a variable vrací True pokud je testovaný ukazatel na proměnné. Vrací False ve všech ostatních případech (ukazatel na poli, tabulce nebo nulový ukazatel).

Od verze 6 je výhodnější místo použití Is a variable použít příkaz RESOLVE POINTER, který vám vrátí informace o odkazovaném objektu, bez rozdílu jaký je to objekt (včetně prázdného ukazatele).

Dále si přečtěte

Nil, RESOLVE POINTER.

Příkazy a odkazy pro Jazyk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Get pointer (Získat ukazatel)

Příkazy a odkazy pro Jazyk

Verze 3

Get pointer (NázevProm) → Ukazatel

Parametr	Typ		Popis
NázevProm	Řetězec	→	Název procesní proměnné
Výsledek funkce	Ukazatel	←	Ukazatel na procesní proměnné

Popis

Příkaz Get pointer vrátí ukazatel na proměnnou jejíž název jste vložili do parametru NázevProm.

K získání ukazatele na pole, použijte Field. K získání ukazatele na tabulku, použijte Table.

Poznámka: Do příkazu Get pointer nemůžete předat výrazy jako $tTabNom+"{3}". Jsou *povolen*y pouze *názvy proměnný*ch.

Příklad

Ve formuláři vytvoříte síť 5x10 s dostupnými proměnnými s názvy v1, v2, ... v50. K nastavení všech těchto proměnných napište:

```
` ...
For ($vlVar;1;50)
   $vpVar:=Get pointer("v"+String($vlVar))
$vpVar→:=""
End for
```

Dále si přečtěte

Field, Table.

Příkazy a odkazy pro Jazyk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

EXECUTE (PROVÉST)

Verze 3

Poznámka: Tento příkaz budete používat jen vyjímečně.

EXECUTE (text))

Parametr	Typ		Popis
text	Řetězec	→	kód nebo název metody k provedení

Popis

Příkaz EXECUTE provede text jako řádek kódu. *Text* musí být jedna řádka. Pokud je předán prázdný řetězec, příkaz neprovede nic.

Pravidlo je že pokud může být tento *text* proveden jako jeden řádek libovolné metody, pak je proveden.

Důležité: Ve zkompilované databázi není tento řádek kompilován. To znamená že výraz bude při běhu kompilovné aplikace proveden, ale není kontrolován 4D Compilerem v době kompilace.

Používejte tento příkaz velmi opatrně, protože snižuje rychlost provádění.

Může být použit v následujícím:
• volání metody projektu
• volání příkazu 4D
• přiřazení

Parametr *text* může obsahovat procesní a meziprocesní proměnné. Nemůže obsahovat smyčky řízení průběhu, protože zde může být pouze jeden řádek.

Příklad

Přečtěte si příklad u příkazu Command Name.

Dále si přečtěte

Command name.

Příkazy a odkazy pro Jazyk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Command name (Název příkazu)

Příkazy a odkazy pro Jazyk

Verze 6.0

Command name (příkaz) → Řetězec

Parametr	Typ		Popis
Příkaz	Číslo	→	Číslo příkazu
Výsledek funkce	Řetězec	←	Lokalizovaný název příkazu

Popis

Příkaz Command name vrací písmenný název příkazu jehož číslo jste zadali jako parametr.

4th Dimension obsahuje dynamické překládání názvů klíčových slov, konstant a příkazů použitých ve vaší databázi. Pokud například pracujete s anglickou verzí 4D a napíšete:

DEFAULT TABLE ([MyTabulka])
ALL RECORDS ([MyTabulka])

Stejný kód ve francouzské verzi bude vypadat takto:
TABLE PAR DEFAUT ([MyTabulka])
TOUT SELECTIONNER ([MyTabulka])

Nicméně 4th Dimension také obsahuje jedinečnou vlastnost. Příkaz EXECUTE, který vám umožní vytvářet kód za běhu aplikace a pak spustit tento kód, dokonce je-li databáze kompilovaná.

Příklad kódu psaného s příkazem EXECUTE v anglické verzi vypadá takto:
EXECUTE ("DEFAULT TABLE([MyTabulka])")
EXECUTE ("ALL RECORDS([MyTabulka])")

Stejný kód ve francouzské verzi bude vypadat takto:
EXECUTER ("DEFAULT TABLE([MyTabulka])")
EXECUTER ("ALL RECORDS([MyTabulka])")

4D automaticky přeloží EXECUTE (anglicky) na EXECUTER (francouzsky), ale nemůže přeložit text který je předáný jako textový parametr.

Pokud použijete příkaz EXECUTE, můžete rovnou použít i příkaz Command name a úplně se tak vyhnout problémům s jazykovou verzí a udělat váš kód nezávislý na jazyku motoru, na kterém je prováděn. Zde je příklad takového kódu:
EXECUTE (**Command name** (46)+"([MyTabulka])")
EXECUTE (**Command name** (47)+"([MyTabulka])")

Ve francouzské verzi 4D bude kód vypadat takto:
EXECUTER (**Nom commande** (46)+"([MyTabulka])")
EXECUTER (**Nom commande** (47)+"([MyTabulka])")

Poznámka: K získání čísla příkazu si klepněte na název příkazu v okně Průzkumníka a číslo příkazu se vám zobrazí v oblasti náhledu.

Příklady

1. Ve všech tabulkách vaší databáze máte formulář s názvem "Vstupní" pro standardní zadávání záznamů. Nyní chcete vložit generickou metodu projektu která nastaví formulář jako platný vstupní formulář pro tabulku, jejíž ukazatel předáte jako parametr k metodě. Napíšete:

```
      ` Metoda projektu STANDARDNÍ VSTUPNÍ FORMULÁŘ
      ` STANDARDNÍ VSTUPNÍ FORMULÁŘ ( Ukazatel {; Řetězec} )
      ` STANDARDNÍ VSTUPNÍ FORMULÁŘ ( →Tabulka {; NázevTabulky} )
C_POINTER ($1)
C_STRING (31;$2)
If (Count parameters>=2)
   EXECUTE (Command name (55)+"(["+$2+"];"Vstupní")")
Else
   If (Count parameters>=1)
      INPUT FORM ($1→;" Vstupní ")
   End if
End if
```

Po uložení této metody do databáze můžete napsat:

```
STANDARDNÍ VSTUPNÍ FORMULÁŘ (→[Zaměstnanci])
STANDARDNÍ VSTUPNÍ FORMULÁŘ (Self;"Zaměstnanci")
```

Poznámka: Většinou je lepší při vytváření generických metod použít ukazatele. Za prvé poběží kód zkompilovaný, když je kompilovaná databáze. Za druhé 4D Insider vám tak či onak tyto ukazatele najde a dereferencuje pro patřičné místo. Za třetí váš kód bude pracovat spolehlivě i když přejmenujete tabulku. Nicméně v některých případech může použití EXECUTE vyřešit jisté problémy.

2. Ve formuláři chcete rozevírací seznam naplněný základními příkazy pro součty. V metodě objektu pro rozevírací seznam předáte:

```
Case of
   ÷ (Form event =On Before)
      ARRAY TEXT (asCommand;4)
      asCommand{1}:=Command name (1) ` Sum
      asCommand{2}:=Command name (2) ` Average
      asCommand{3}:=Command name (4) ` Min
      asCommand{4}:=Command name (3) ` Max
         ` ...
End case
```

V anglické verzi 4D bude seznam zobrazovat: Sum, Average, Min a Max a ve francouzské verzi bude zobrazovat: Somme, Moyenne, Min a Max.

Dále si přečtěte

EXECUTE.

Příkazy a odkazy pro Jazyk

4th Dimension 6.5, Seznam témat příkazů

TRACE (KROKOVAT)

Příkazy a odkazy pro Jazyk

Verze 3

TRACE

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Použijte příkaz TRACE ke krokování metody během vývoje databáze.

Příkaz TRACE zapne Ladění, pro platný proces 4th Dimension. Okno Ladění se otevře před spuštěním dalšího řádku a pokračuje ve zobrazování pro každý řádek,který má být spuštěn. Ladění můžete zapnout také stisknutím Alt (Windows) nebo Option (Macintosh) a klepnutím myši během provádění metody.

Ve zkompilované databázi je příkaz TRACE ignorován.

4D Server: Pokud zavoláte příkaz TRACE z metody projektu, otevře se okno Ladění na serveru.

Tipy

1. Nevkládejte příkaz TRACE při používání formuláře, který má aktivované události Při zavedení a Při vyvedení. Pokaždé když se objeví okno Ladění, budou tyto události opět aktivovány: uváznete v nekonečné smyčce mezi těmito událostmi a oknem ladění. Pokud se do této situace dostanete, můžete z ní odejít přepnutím do prostředí Návrháře.

Aby jste to udělali, klepněte do okna události nebo do okna ladění. Pak proveďte následující:

• Pokud je příkaz TRACE v metodě projektu nebo objektu (tyto metody jsou znova načteny při spuštění), vymažte jej. Ukončíte nekonečnou smyčku.
• Pokud je příkaz TRACE v metodě formuláře, tato metoda nebude znovu načtena dokud nebude formulář uzavřen, což je nyní nemožné protože jste uvázli ve smyčce. Musíte tedy znovu otevřít databázi nebo přerušit provádění procesu. Pokud nemůže být proces zrušen (proces uživatele/aplikace), pak musíte vypnout a znovu zapnout databázi.

2. Pokud bude příkaz TRACE volán v metodě formuláře nebo objektu prováděné při obnovování formuláře na obrazovce, také uváznete v nekonečném opakování obnovy formuláře a oknem ladění. V tomto případě stiskněte Alt+Shift (Windows) nebo Option+Shift (Macintosh). Zrušíte tak obnovovací událost pro platné okno a zastavíte volání příkazu TRACE pro metodu formuláře nebo objektu. Pak se můžete přepnout do Prostředí návrháře a odstranit příkaz TRACE.

Všiměte si, že oba tyto tipy se vztahují ke stejné situaci způsobené přítomností trvalého přerušení kódu. Co se týká prvního odseku Tipu 1, nezáleží na tom kde je přerušení, můžete jej odstranit a tím odejít z okna Ladění bez nutnosti vypínat databázi.

Příklad

Následující kód očekává, že proměnná Jazyk_motoru bude "US" nebo "FR". Pokud to tak není, zavolá metodu projektu DEBUG:

```
` ...
Case of
   ÷ (Jazyk_motoru="US")
      vsBHCmdName:=[Příkazy]CM US Name
   ÷ (Jazyk_motoru="FR")
      vsBHCmdName:=[Příkazy]CM FR Name
Else
```

```
            DEBUG ("Neznámá JAZYK_MOTORUhodnota")
        End case
```

Metoda projektu DEBUG je zobrazena zde:

```
    ` Metoda projektu DEBUG
    ` DEBUG (Text)
    ` DEBUG (Volitelná informace ladění)
    C_TEXT ($1)
    If (<>vbDebugOn) ` Meziprocesní proměnná nastavená v metodě Při spuštění
        If (Compiled Application)
            If (Count parameters>=1)
                ALERT ($1+Char(13)+"Zavolejte návrháře. ")
            End if
        Else
            TRACE
        End if
    End if
```

Dále si přečtěte

NO TRACE.

Příkazy a odkazy pro Jazyk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

NO TRACE (NEKROKOVAT)

Verze 3

NO TRACE

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Použijte příkaz NO TRACE během vývoje databáze.

NO TRACE vypne okno Ladění otevřené příkazem TRACE, při chybě nebo akcí uživate. Příkaz NO TRACE má stejný efekt jako stisknutí tlačítka Nekrokovat v okně Ladění.

Ve zkompilované databázi je příkaz NO TRACE ignorován.

Dále si přečtěte

TRACE.

Příkazy a odkazy pro Jazyk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Abs (Absolutní hodnota)

Verze 3

Abs (číslo) → Číslo

Parametr	Typ		Popis
číslo	Číslo	→	Číslo ke kterému vrátit absolutní hodnotu
Výsledek funkce	Číslo	←	Absolutní číslo parametru

Popis

Abs vrátí absolutní hodnotu předaného čísla *číslo* (kladnou, bez znaménka). Pokud je *číslo* záporné, je vráceno jako kladné. Pokud je *číslo* kladné je vráceno nezměněné.

Příklad

Následující příklad vrací absolutní hodnotu z –10,3, což je 10,3:

```
vlVector:=Abs(-10,3)
```

Příkazy a odkazy pro Matematika

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Int (Celá část)

Verze 3

Int (číslo) → Číslo

Parametr	Typ		Popis
číslo	Číslo	→	Číslo jehož celou část chceme
Výsledek funkce	Číslo	←	Celá část čísla

Popis

Int vrací celé číslo předaného čísla *číslo*. Int ořízne desetinná místa (záporná místa od nuly).

Příklad

Následující příklad ukazuje jak Int pracuje jak s kladnými tak se zápornými čísly. Všiměte si že desetinná část čísla je odstraněna:

```
vlVysledek:=Int(123,4) ` vlVysledek bude 123
vlVysledek:=Int(-123,4) ` vlVysledek bude -123
```

Dále si přečtěte

Dec.

Příkazy a odkazy pro Matematika

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Dec (Desetinná část)

Příkazy a odkazy pro Matematika

Verze 3

Dec (číslo) → Číslo

Parametr	Typ		Popis
číslo	Číslo	→	Číslo jehož desetinnou hodnotu chceme
Výsledek funkce	Číslo	←	Desetinná část čísla.

Popis

Dec vrací desetinnou (racionální) část čísla *číslo*. Vrácena hodnota je vždy kladná nebo nula.

Příkady

Následující příklad vezme peněžní částku vyjádřenou jako reálné číslo a oddělí část korun a část haléřů. Jestliže je vlHodnota 7,35, pak vlKorun bude 7 a vlHaléř bude 35:

```
vlKorun:=Int (vlHodnota) ` Vezme koruny
vlKorun:=Dec (vlHodnota) * 100 ` Vezme zlomkovou část
```

Dále si přečtěte

Int.

Příkazy a odkazy pro Matematika

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Round (Zaokrouhlit)

Příkazy a odkazy pro Matematika

Verze 3

Round (zaokr; míst) → Číslo

Parametr	Typ		Popis
zaokr	Číslo	→	Číslo které chceme zaokrouhlit
míst	Číslo	←	počet desetiných míst na zaokrouhlení
Výsledek funkce	Číslo	←	zaokrouhlené čísla

Popis

Round vrátí *číslo* zaokrouhlené na počet desetiných míst definovaných parametrem *míst*.

Pokud je *míst* kladné, číslo je zaokrouhleno na záporné řády (míst za desetinnou tečkou). Pokud je *míst* záporné je číslo zaokrouhleno na kladné řády (- míst před desetinnou tečkou).

Jestliže je číslice následovaná za řádem *míst* 5 až 9, Round zaokrouhlí číslo nahoru. Pokud číslice následovaná za řádem *míst* 0 až 4, Round zaokrouhlí dolů.

Příklady

Následující příklad ukazuje jak Round pracuje s rozdílnými argumenty. Každý řádek přiřadí hodnotu do VlResult. Komentáře popisují výsledek:

```
vlVysledek:=Round (16,857; 2) ` vlVysledek bude 16.86
vlVysledek:=Round (32345,67; -3) ` vlVysledek bude 32000
vlVysledek:=Round (29,8725; 3) ` vlVysledek bude 29.873
vlVysledek:=Round (-1,5; 0) ` vlVysledek bude -2
```

Dále si přečtěte

Trunc.

Příkazy a odkazy pro Matematika

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Trunc (Oříznout)

Verze 3

Trunc (kořízn; míst) → Číslo

Parametr	Typ		Popis
kořízn	Číslo	→	Číslo, které chceme oříznout
míst	Číslo	→	Počet desetiných míst, na která chceme oříznout
Výsledek funkce	Číslo	←	Číslo oříznuté na určený počet desetiných míst

Popis

Trunc vrací číslo *kořízn* oříznuté na tolik desetinných míst, kolik míst vzhledem k desetinné tečce zůstane je definováno v parametru *míst*. Trunc vždy ořízne směrem k zápornému nekonečnu.

Pokud je *míst* kladné, číslo je oříznuto na záporné řády (míst za desetinnou tečkou). Pokud je *míst* záporné je číslo oříznuto na kladné řády (- míst před desetinnou tečkou).

Příklady

Následující příklad ukazuje jak Trunc pracuje s rozdílnými argumenty. Každý řádek přiřadí číslo k proměnné vlVysledek. Komentáře popisují výsledek.

```
vlVysledek := Trunc (216,897; 1) ` vlVysledek bude 216.8
vlVysledek := Trunc (216,897; -1) ` vlVysledek bude 210
vlVysledek := Trunc (-216,897; 1) ` vlVysledek bude -216.9
vlVysledek := Trunc (-216,897; -1) ` vlVysledek bude -220
```

Dále si přečtěte

Round.

Příkazy a odkazy pro Matematika

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Random (Náhodné)

Příkazy a odkazy pro Matematika

Verze 3

Random → Číslo

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Číslo	←	Náhodné číslo

Popis

Random vrací náhodné integer (celé) číslo mezi 0 a 32 767 (včetně).

K definování rozsahu čísel, použijte tento výraz:

(Random%(Konec-Začátek+1))+Začátek

Hodnota Začátek je první číslo rozsahu a hodnota Konec je poslední číslo rozsahu.

Příklad

Následující příklad přiřadí náhodné celé číslo mezi 10 a 30 do proměnné vlVysledek:

vlVysledek:=(**Random**%21)+10

Příkazy a odkazy pro Matematika

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Mod (Zbytek)

Příkazy a odkazy pro Matematika

Verze 3

Mod (číslo1; číslo2) -→> Číslo

Parametr	Typ		Popis
číslo1	Číslo	→	Číslo - dělenec
číslo2	Číslo	→	Číslo - dělitel
Výsledek funkce	Číslo	←	Vrátí zbytek

Popis

Příkaz Mod vrátí zbytek dělení *číslo1*/*číslo2*.

Poznámka: Mod přijímá Integer, Long Integer a Real hodnoty. Pokud jsou obě čísla Real, pak jsou nejdříve zaokrouhlena a pak je vypočítáno modulo.

Můžete také použít operátor % k vypočítání zbytku (přečtěte si Číselné operátory).

UPOZORNĚNÍ: Operátor % může počítat s Integer a Long Integer hodnotami. Pokud potřebujete spočítat real číslice, nemůžete regulérně použít příkaz Mod.

Příklady

Následující příklad ukazuje jak funkce Mod, pracuje s rozdílnými hodnotami. Každý řádek přiřadí číslo do proměnné vlVysledek. Komentáře popisují výsledek:

```
vlVysledek:=Mod(3;2) ` vlVysledek bude 1
vlVysledek:=Mod(4;2) ` vlVysledek bude 0
vlVysledek:=Mod(3.5;2) ` vlVysledek bude 0 (4/2)
```

Dále si přečtěte

Číselné operátory.

Příkazy a odkazy pro Matematika

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Square root (Druhá odmocnina)

Verze 6.0

Square root (číslo) → Číslo

Parametr	Typ		Popis
číslo	Číslo	→	Číslo jehož druhou odmocninu dostaneme
Výsledek funkce	Číslo	←	Druhá odmocnina čísla

Popis

Square root vrátí druhou odmocninu čísla *číslo*.

Příklady

1. Řádek:

```
$vrDruháOdmocnDvou := Square root (2)
```

přiřadí hodnotu 1.414213562373 do proměnné $ vrDruháOdmocnDvou.

2. Následující metoda vrací přeponu pravoúhlého trojúhelníku jehož dvě ramena jsou přiřazena jako parametry:

```
` Metoda Přepona
` Přepona ( real ; real ) → real
` Přepona ( OdvěsnaA ; OdvěsnaB ) → Přepona
C_REAL($0;$1;$2)
$0 := Square root(($1^2)+($2^2))
```

Například, *Přepona* (4;3) vrací 5.

Dále si přečtěte

Číselné operátory.

Příkazy a odkazy pro Matematika

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Log

Příkazy a odkazy pro Matematika

Verze 3

Log(číslo) → Číslo

Parametr	Typ		Popis
číslo	Číslo	→	číslo pro které vrátit logaritmus
Výsledek funkce	Číslo	←	Log čísla

Popis

Log vrátí základní (Napieriho) log čísla *číslo*. Log je obrácená funkce Exp.

Poznámka: 4D obsahuje předdefinovanou konstantu čísla *e* e number (2,71828...)

Příklad

Následující řádek zobrazí 1:

```
ALERT (String(Log(Exp(1))))
```

Dále si přečtěte

Exp.

Příkazy a odkazy pro Matematika

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Exp

Příkazy a odkazy pro Matematika

Verze 3

Exp (číslo) → Číslo)

Parametr	Typ		Popis
číslo	Číslo	→	Číslo k výpočtu
Výsledek funkce	Číslo	←	Základ přirozených logaritmů na číslo

Popis

Exp umocňuje základ přirozených logaritmů (e = 2,71828...) o mocninu čísla *číslo*. Exp je opak funkce Log.

Poznámka: 4D obsahuje předdefinovanou konstantu čísla *e* e number (2,71828...)

Příklad

Následující příklad přiřadí exponencial 1 do vrE (Log vrE je 1):

```
vrE := Exp (1) ` vrE bude 2,7182818.....
```

Dále si přečtěte

Log.

Příkazy a odkazy pro Matematika

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Sin

Příkazy a odkazy pro Matematika

Verze 3

Sin (číslo) → Číslo

Parametr	Typ		Popis
číslo	Číslo	→	Číslo v radianech, jehož sinus chceme
Výsledek funkce	Číslo	←	Sinus čísla

Popis

Sin vrací sinus čísla *číslo*; kde *číslo* je vyjádřeno v radiánech.

Poznámka: 4D obsahuje předdefinované konstanty Pi , Degree a Radian. *Pi* vrací číslo Pí (3,14159...), *Degree* vrací jeden stupeň vyjádřený v radiánech (0,01745...) a *Radian* vrací jeden radián vyjádřený ve stupních (57,29577...).

Dále si přečtěte

Arctan, Cos, Tan.

Příkazy a odkazy pro Matematika

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Cos

Příkazy a odkazy pro Matematika

Verze 3

Cos (číslo) → Číslo

Parametr	Typ		Popis
číslo	Číslo	→	Číslo, v radiánech, jehož sinus chceme
Výsledek funkce	Číslo	←	Cosinus čísla

Popis

Cos vrací cosinus čísla *číslo*; kde *číslo* je vyjádřeno v radiánech.

Poznámka: 4D obsahuje předdefinované konstanty Pi , Degree a Radian. *Pi* vrací číslo Pí (3,14159...), *Degree* vrací jeden stupeň vyjádřený v radiánech (0,01745...) a *Radian* vrací jeden radián vyjádřený ve stupních (57,29577...).

Dále si přečtěte

Arctan, Sin, Tan.

Příkazy a odkazy pro Matematika

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Tan (Tangens)

Příkazy a odkazy pro Matematika

Verze 3

Tan (číslo) → Číslo

Parametr	Typ		Popis
číslo	Číslo	→	Číslo, v radiánech, jehož tangens chceme
Výsledek funkce	Číslo	←	Tangens čísla

Popis

Tan vrací tangens čísla *číslo*; kde *číslo* je vyjádřeno v radiánech.

Poznámka: 4D obsahuje předdefinované konstanty Pi , Degree a Radian. *Pi* vrací číslo Pí (3,14159...), *Degree* vrací jeden stupeň vyjádřený v radiánech (0,01745...) a *Radian* vrací jeden radián vyjádřený ve stupních (57,29577...).

Dále si přečtěte

Arctan, Cos, Sin.

Příkazy a odkazy pro Matematika

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Arctan (Arcustangens)

Příkazy a odkazy pro Matematika

Verze 3

Arctan (číslo) → Číslo

Parametr	Typ		Popis
číslo	Číslo	→	Tangens pro který chceme vypočítat úhel
Výsledek funkce	Číslo	←	Úhel v radianech

Popis

Arctan vrací úhel v radiánech čísla *číslo*; kde *číslo* je tangens.

Poznámka: 4D obsahuje předdefinované konstanty Pi , Degree a Radian. *Pi* vrací číslo Pí (3,14159...), *Degree* vrací jeden stupeň vyjádřený v radiánech (0,01745...) a *Radian* vrací jeden radián vyjádřený ve stupních (57,29577...).

Příklad

Následující příklad ukazuje hodnotu Pí:

```
ALERT ("Pí se rovná: "+String(Arctan(1)*4))
```

Dále si přečtěte

Tan, Cos, Sin.

Příkazy a odkazy pro Matematika

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET REAL COMPARISON LEVEL (NASTAVIT ÚROVEŇ POROVNÁVÁNÍ)

Verze 6.0

SET REAL COMPARISON LEVEL (epsilon)

Parametr	Typ		Popis
epsilon	Číslo	→	Epsilon hodnota pro porovnání reálných čísel výchozí nastavená je 6 t.j pro přesnost (10^-6)

Popis

Příkaz SET REAL COMPARISON LEVEL nastaví hodnotu epsilon použitou 4th Dimension k porovnání reálné hodnoty a vyhodnoceí rovnosti.

Počítač vždy provádí reálné výpočty přibližně: proto testování reálných čísel zda se rovnají musí vzít tuto přibližnost do úvahy. 4th Dimension toto provádí při porovnávání reálných čísel testováním jestli rozdíl mezi těmito čísly překročí jistou hodnotu. Tato hodnota se nazývá **epsilon** a pracuje následujícím způsobem:

Vezměme dvě reálná čísla a a b, jestliže Abs(a-b) je větší než epsilon, čísla nejsou uznána jako stejná; jinak jsou čísla prohlášena za stejná.

Jako výchozí 4th Dimension nastaví epsilon na hodnotu 10 na mínus 6 (10^-6). Nezapomeňte že hodnota epsilon může být změněna. Příklady:

- 0.00001=0.00002 vrátí False, protože rozdíl 0.00001 je větší než 10^-6.
- 0.000001=0.000002 vrátí True, protože rozdíl 0.000001 není větší než 10^-6.
- 0.000001=0.000003 vrátí False, protože rozdíl 0.000002 je větší než 10^-6.

S použitím příkazu SET REAL COMPARISON LEVEL můžete tuto hodnotu zvětšit nebo zmenšit podle vašich potřeb.

Poznámka: Jestliže chcete provádět hledání a třídění na reálném číselném indexovaném poli jehož hodnoty jsou menší než 10^-6, ujistěte se, že jste před vytvořením tabulky indexů použili příkaz SET REAL COMPARISON LEVEL.

UPOZORNĚNÍ: Většinou tento příkaz pro upravení hodnoty epsilon nebudete potřebovat.

DŮLEŽITÉ: Změna epsilon má význam pouze pro porovnávání. Nemá žádný efekt na jiné výpočty nebo zobrazení reálných čísel.

Dále si přečtěte

Operátory porovnání.

Příkazy a odkazy pro Matematika

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Zobrazení reálných čísel

Příkazy a odkazy pro Matematika

Verze 6.0

Předběžná poznámka

Pokud nevyvíjíte pro více platforem, můžete tuto část přeskočit.

Na počítačích jsou výpočty více technologie než matematická věda. Například ve škole se učíte že jedna třetina (1/3) může být zapsána jako nekonečný počet trojek za desetinou čárkou. Počitač toto neví a musí si výrau vypočítat. Uplně stejně víte, že třikrát jedna třetina je jedna; počítač si to musí spočítat aby dostal výsledek. Podle typu počítače který používáte, je jedna třetina nějak omezený počet trojek za desetinou čárkou. Tento počet se nazývá "přesnost" počítače.

Na 68K Macintoshi je číslo přesnosti 19; to znamená že 1/3 je vypočítána na 19 významných číslic. Na Windows nebo Power Macintosh je toto číslo 15; tak 1/3 bude zobrazena pouze na 15 významných číslic. Pokud zobrazíte výraz 1/3 v okně Ladění ve 4th Dimension, dostanete 0,3333333333333333333 na 68K Macintoshi a něco jako 0,333333333333333148 na Windows nebo Power Macintoshi. Všiměte si že poslední tři číslice jsou jiné, protože přesnost na Windows a Power Macintosh je menší než na 68K Macintoshi. Nyní, když zobrazíte výraz (1/3)*3, dostanete výsledek 1.

Pokud se vaše výpočty omezují na počet čtverečních metrů vašeho dvorku, můžete si říci "Mě se to netýká!" protože se nemusíte starat o to, kolik budete mít čísel za desetinou čárkou. Na druhou stránku pokud vyplňujete příkaz na výplatu dividend z akcií, můžete se, v některých případech, obávat přesnosti vašeho počítače. Nicméně si zapamatujte že 19 a 15 číslic za desetinou čárkou je nedostatečné pouze v případě že pracujete s biliony korun celkových dividend.

Proč vypadá hodnota 1/3 jinak na 68K Macintoshi a jinak na Windows a Power Macintoshi?

Na 68K Macintoshi ukládá operační systém čísla na 10 bytů (80 bitů) zatímco na Windows a Power Macintoshi je ukládá do 8 bytů (64 bitů). Toto je důvod proč mají reálná čísla 19 významných číslic na 68K Macintoshi a 15 významných číslic na Windows a Power Macintoshi.

Tak proč výraz (1/3)*3 vrací 1 na obou strojích?

Počítač může provést pouze přibližné výpočty. Proto při porovnávání a počítání čísel nebere počítač reálná čísla jako matematické objekty ale jako přibližné hodnoty. V našem příkladu vynásobení 0,33333... třemi bude 0,99999...; rozdíl mezi 0,99999... a 1 je tak malý že počítač provede výsledek jako 1 a vrátí 1. Pro podrobné informace o této věci si přečtěte diskuzi k příkazu SET REAL COMPARISON LEVEL.

Jsou dva způsoby chování reálných čísel a proto mezi nimi musíme dělat rozdíly:

- Jak jsou porovnávány a počítány
- Jak jsou zobrazeny na obrazovce a na tiskárně

4th Dimension jako výchozí používá pro reálná čísla standardní 10 bytový typ dat jak je používáno na 68K Macintoshi. Následně pro uložení reálných hodnot na disku používá rovněž tento formát. Pro zachvání kompatibility mezi verzemi 4th Dimension pro Macintosh, Power Macintosh a Windows datové soubory stále ukládají reálné hodnoty s použitím 10 bytového formátu. Protože na Windows a Power Macintoshi je používán 8 bytový systém, 4th Dimension převádí 10 bytové hodnoty na 8 bytové a naopak. Proto pokud načtete hodnotu která byla uložena na 68K Macintoshi z Windows nebo Power Macintoshe, může se vám stát, že ztratíte určitou přesnost (z 19 na 15 významých číslic). Když načtete záznam uložený na Windows nebo Power Macintoshi a otevřete jej na 68K Macintoshi, nedojde k žádné ztrátě přesnosti. Proto když používáte databázi současně na 68K Macintoshi a Power Macintoshi nebo Windows, používejte přesnost výpočtu v pohyblivé čárce na 15 znaků ne na 19.

S použitím Customizer Plus můžete nastavit počet znaků, které se přeskočí při zjednodušeném zobrazení na 68K a Power Macintosh nebo Windows. Výchozí nastavení je žádné číslice na 68K a pět číslic na Power Macintosh a Windows.

Příkazy a odkazy pro Matematika

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Euro converter (Převést euro měny)

Příkazy a odkazy pro Matematika

Verze 6.5

Euro Converter (hodnota; zMěny; doMěny) → Real

Parametr	Typ		Popis
hodnota	Real	→	Hodnota k převedení
zMěny	Řetězec	→	Kód měny ze které je hodnota
doMěny	Řetězec	→	Kód měny do které převést
Výsledek funkce	Real	←	Převedená hodnota

Popis

Příkaz Euro converter vám umožní převést jakoukoli *hodnotu* z a do jiných měn patřících do Evropského společenství a do měny Euro.

Můžete převést:

• Národní měnu do Euro
• Euro do národní měny
• národní měnu do jiné národní měny. V tomto případě je převod proveden podle Euro jak je definováno v Evropských pravidlech. Například k převodu z Belgických franků do Německých marek, 4D provede následující výpočet: Belgické franky → Euro → Německé marky.

Hodnotu k převedení předáte do prvního parametru *hodnota*.

Druhý parametr *zMěny* definuje kód měny kterou chcete převádět.

Třetí parametr *doMěny* definuje měnu do které chcete převádět.

K definování kódu měny, 4th Dimension obsahuje následující předdefinované konstanty umístěné v tématu Euro Currencies:

Konstanta	***Typ***	***Hodnota***
Austrian Schilling	*String*	*ATS*
Belgian Franc	*String*	*BEF*
Deutschemark	*String*	*DEM*
Euro	*String*	*EUR*
Finnish Markka	*String*	*FIM*
French Franc	*String*	*FRF*
Irish Pound	*String*	*IEP*
Italian Lire	*String*	*ITL*
Luxembourg Franc	*String*	*LUF*
Netherlands Guilder	*String*	*NLG*
Portuguese Escudo	*String*	*PTE*
Spanish Peseta	*String*	*ESP*

Pokud to bude nutné, 4th Dimension sama provede zaokrouhlení na 2 desetiná místa - mimo převodu do Italkých lir, Belgických franků, Lucemburských franků a Španělských peset, pro které 4D bere 0 desetiných míst (výsledek je celé číslo).

Převody mezi Euro a ostatními 11 členskými státy jsou pevně dané:

Měna	***Hodnota pro 1 Euro***
Austrian Schilling	*13.7603*

Belgian Franc	*40.3399*
Deutschemark	*1.95583*
Finnish Markka	*5.94573*
French Franc	*6.55957*
Irish Pound	*0.787564*
Italian Lire	*1936.27*
Luxembourg Franc	*40.3399*
Netherlands Guilder	*2.20371*
Portuguese Escudo	*200.482*
Spanish Peseta	*166.386*

Příklad

Zde jsou příklady převodů které mohou být provedeny tímto příkazem:

```
$value:=10000 `Hodnota vyjádřená ve Francouzských francích
    `Převést hodnotu do Euro
$InEuros:=Euro converter($value;French Franc; Euro)
    `Převést hodnotu do Italských lir
$InLires:=Euro converter ($value;French Franc; Italian Lire)
```

Příkazy a odkazy pro Matematika

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Řízení nabídek

Příkazy a odkazy pro Nabídky

Verze 6.0 (Upraveno)

Terminologie

Dokumentace o nabídkách používá záměnně výrazy **položka nabídky** a **příkaz nabídky** při popisování řádků (položek) nabídky.

Záhlaví nabídek

Záhlaví nabídek jsou identifikována spíše číslem než názvem. První záhlaví se nazývá Záhlaví #1. Je to výchozí záhlaví nabídek. Pokud chcete databázi otevřít s jiným záhlavím než Záhlaví #1, musíte pužít příkaz MENU BAR v metodě databáze Při spuštění.

Každý příkaz nabídky může mít přiřazenu metodu projektu. Pokud nepřiřadíte metodu k položce nabídky, vybrání této položky řekne 4th Dimension aby opustila prostředí Vlastních nabídek a vrátila se do Prostředí uživatele. Pokud uživatel pracuje pouze s verzí 4th Dimension pro prostředí vlastních nabídek, znamená to pro něj, že bude databáze vypnuta a on navrácen na plochu.

Každé záhlaví začíná vznikat se třemi nabídkami - Soubor, Úpravy a Nápověda (Windows) nebo Jablíčko, Soubor a Upravit (Macintosh).

• Nabídka *Soubor* obsahuje pouze jednu položku - Konec. Položka Konec nemá přiřazenu žádnou metodu a tím řekne 4th Dimension aby opustila prostředí Vlastních nabídek. Můžete nabídku Soubor přejmenovat, přidat do ní položky nebo ji nechat nezměněnou.Jakmile je tato nabídka přejmenovaná, již se nebude objevovat nalevo od nabídky Upravy.. Je doporučeno aby jste jako poslední položku nabídky Soubor měli položku Konec.
• Nabídka *Upravy* obsahuje standardní příkazy pro upravení. Nabídka Úpravy není zobrazena v Editoru nabídek a nemůže být upravena.
• Na Windows obsahuje nabídka Nápověda O 4th Dimension a Windows soubory nápovědy které vytvoříte pro aplikaci. Nabídka Nápověda není zobrazena v Editoru a nemůže být upravena, mimo položky O ..., která může být upravena pomocí příkazu SET ABOUT.
• Na Macintoshi obsahuje nabídka Jablíčko O 4th Dimension a aplikace umístěné v systémové složce Apple menu Items. Nabídka Jablíčko není zobrazena v Editoru a nemůže být upravena, mimo položky O ..., která může být upravena pomocí příkazu SET ABOUT.

Čísla nabídek a Čísla položek nabídek

Stejně jako záhlaví, jsou číslovány i nabídky. Nabídky Nápověda, Úpravy a Jablíčko nejsou zařazeny do tohoto součtu a nemohou být měněny. Místo toho má nabídka Soubor číslo 1 a ostatní nabídky jsou číslovány postupně zleva do prava (2, 3, 4, atd.). Číslování nabídek je důležité pokud pracujete například s funkcí Menu selected. Jakmile je nabídka přiřazena k formuláři, mění se schéma číslování. První připojená nabídka začíná číslem 2049. K odkazu na připojenou nabídku přidejte 2048 k normálnímu číslu nabídky.

Položky v nabídce jsou číslovány postupně ze shora dolů. Položka nahoře má číslo 1.

Přiřazení záhlaví nabídek

Záhlaví nabídek přiřadíte k formuláři v okně Vlastnosti formuláře na straně Obecné. Takové záhlaví nabídek se v

tomto dokumentu nazývá "Záhlaví nabídek formuláře".

Nabídky v záhlaví nabídek formuláře jsou připojeny k platné nabídce při zobrazení formuláře. Nabídky jsou pro vstupní formulář připojeny jak v prostředí Vlastních nabídek tak Uživatele. Pro výstupní formulář jsou připojeny pouze v prostředí Vlastních nabídek.

Zahlaví nabídek formuláře je definováno číslem. Pokud je číslo platného záhlaví nabídek stejné jako číslo záhlaví nabídek přiřazené k formuláři, není záhlaví připojeno.

Pokud definujete záporné číslo pro záhlaví nabídek formuláře, 4th Dimension použije absolutní hodnotu záhlaví. Pokud například předáte -3, přiřadí se Záhlaví #3. Jakmile je záhlaví nabídek definováno záporným číslem, příkazy nabídek pro všechny nabídky v záhlaví (úvodní obrazovky a formuláře) budou provádět metody které jsou k nim přiřazeny.

Pokud nedefinujete záporné číslo pro záhlaví nabídek formuláře, vybrání položky nabídky z připojeného záhlaví nabídek metodu nespustí. Místo toho je do metody formuláře poslána událost Při volbě z nabídky a vy můžete pomocí Menu selected testovat vybranou nabídku.

Programové upravení položek nabídek

4th Dimension obsahuje následující příkazy pro přidání, vymazání, vložení nebo upravení položek nabídky pro nabídky v právě zobrazeném nebo instalovaném záhlaví v procesu:

- ENABLE MENU ITEM
- DISABLE MENU ITEM
- SET MENU ITEM
- SET MENU ITEM STYLE
- SET MENU ITEM MARK
- SET MENU ITEM
- APPEND MENU ITEM
- INSERT MENU ITEM
- DELETE MENU ITEM

Rozsah každého tohoto příkazu je platné záhlaví nabídek. Po opětovném provedení příkazu MENU BAR jsou všechny změny odstraněny a záhlaví je vráceno do podoby uložené v prostředí návrháře.

Každý z těcho příkazů vyžaduje číslo nabídky a číslo položky nabídky.

Jak bylo popsáno, jsou nabídky číslovány od 1 do N zleva doprava. Například Soubor je obvykle první nabídka. Na Windows jsou z tohoto číslování vyjmuty Upravit a Nápověda a na Macintoshi jsou vyjmuty Jablíčko a Upravit. Nezapomeňte, že příkaz Count menus nepočítá tyto nabídky do svého součtu.

Pokud máte například záhlaví složené z nabídek Soubor, Faktury a Zákazníci, vrátí příkaz Count menus 3 a bude ignorovat systémové nabídky zobrazované 4th Dimension.

Položky nabídek jsou číslovány od 1 do N zezhora dolů, včetně oddělovacích čar.

Nabídky v záhlaví nabídek formuláře (záhlaví nabídek připojené k formuláři), které je připojeno k platnému záhlaví nabídek, jsou číslovány stejně s tím rozdílem, že je potřeba k nim přičíst 2048. Tím pádem jsou číslovány od 2048+1 až N.

Příkaz Menu selected vrací číslo nabídky a položky s použitím těchto konvencí.

Upozornění: Tyto příkazy nemohou zpřístupnit systémové nabídky zobrazované 4D (Upravit, Nápověda na Windfows nebo Upravit, Jablíčko na Macintoshi).

Připojené nabídky: Nabídky mohou být připojeny k záhlaví nabídek. Pokud je připojená nabídka upravena jedním z těchto příkazů, jsou upraveny všechny její výskyty. Jestli chcete vědět více informací o připojených nabídkách,přečtěte si *Příručku návrháře 4th Dimension*.

Příkazy a odkazy pro Nabídky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

MENU BAR (ZÁHLAVÍ NABÍDEK)

Příkazy a odkazy pro Nabídky

Verze 3

MENU BAR (záhlnabídky {; proces}{; *})

Parametr	Typ		Popis
záhlnabídky	Číslo	→	Číslo záhlaví nabídky
proces	Číslo	→	Číslo odkazu na proces
*		→	Návrat k předchozímu stavu položek nabídek

Popis

MENU BAR vymění platné záhlaví nabídek pro platný proces za záhlaví předané parametrm *záhlnabídky*.

Volitelný parametr *proces* vymění záhlaví pro proces jehož číslo zadáte. Volitelný parametr * vám umožní uložit předchozí stav záhlaví a vrátit se k této předchozí uložené verzi záhlaví. Jestliže je tento parametr vynechán, 4th Dimension při dalším použití MENU BAR přeinicializuje záhlaví dle stavu v uloženém v návrhu databáze.

Například bude spuštěn příkaz MENU BAR (1). Některé položky jsou deaktivovány příkazem DISABLE MENU ITEM.

Jestliže je provedeno MENU BAR(1) podruhé, ze stejného procesu nebo z jiného procesu, budou všechny položky přestaveny na jejich původní stav.

Pokud v obou případech provedete MENU BAR (1;*) budou po druhém volání položky deaktivované stejně jako jste je nastavili při prvním použití tohoto záhlaví.

Poznámka: Pokud nepoužijete volitelný parametr *proces*, parametr * může být použit jako druhý parametr. Jinými slovy MENU BAR (1;2;*) a MENU BAR (1;*) jsou obě platné příkazy.

Jakmile uživatel vstoupí do prostředí Vlastních nabídek, je otevřeno první záhlaví Záhlaví #1. Můžete jej změnit pokud v metodě databáze Při spuštění použijete příkaz MENU BAR a nastavíte jiné záhlaví pro jednotlivé přihlášené uživatele.

Příklady

1. Následující příklad mění platné záhlaví na záhlaví 3 a přestaví nastavení položek na jejich originální hodnotu:

```
MENU BAR (3)
```

2. Následující příklad mění platné záhlaví na záhlaví 3 a uloží stav položek nabídky. Položky které byly dříve deaktivovány tak zůstanou i nadále.

```
MENU BAR (3;*)
```

3. Následující příklad nastaví platné záhlaví na záhlaví 3 při upravování záznamů. Po skončení úprav záznamů je záhlaví nabídek nastaveno na 2 se zachováním stavu položek:

```
MENU BAR(3)
ALL RECORDS([Zákazníci])
MODIFY SELECTION([Zákazníci])
MENU BAR(2;*)
```

Dále si přečtěte

Řízení nabídek.

Příkazy a odkazy pro Nabídky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

HIDE MENU BAR (SKRÝT ZÁHLAVÍ NABÍDEK)

Příkazy a odkazy pro Nabídky

Verze 6.0

HIDE MENU BAR

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz HIDE MENU BAR skryje záhlaví nabídek.

Jesliže již bylo záhlaví skryto, příkaz neprovede nic.

Příklad

Následující příklad zobrazí záznam v zobrazení na celou obrazovku (Macintosh) dokud nestisknete tlačítko myši:

```
HIDE TOOL BAR
HIDE MENU BAR
Open window(-1;-1;1+ Screen width;1+ Screen height;Alternate dialog box)
INPUT FORM([Malby];"Celá obrazovka 800")
DISPLAY RECORD([Malby])
Repeat
   GET MOUSE($vlX;$vlY;$vlButton)
Until($vlButton#0)
CLOSE WINDOW
SHOW MENU BAR
SHOW TOOL BAR
```

Poznámka: Na Windows bude okno omezeno velikostí okna aplikace.

Dále si přečtěte

HIDE TOOL BAR, SHOW MENU BAR, SHOW TOOL BAR.

Příkazy a odkazy pro Nabídky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SHOW MENU BAR (UKÁZAT ZÁHLAVÍ NABÍDEK)

Verze 6.0

SHOW MENU BAR

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz SHOW MENU BAR ukáže skryté záhlaví nabídek.

Jestliže je záhlaví nabídek viditelné, příkaz neprovede nic.

Příklad

Podívejte se na příklad u příkazu HIDE MENU BAR.

Dále si přečtěte

HIDE MENU BAR, HIDE TOOL BAR, SHOW TOOL BAR.

Příkazy a odkazy pro Nabídky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET ABOUT (NASTAVIT O APLIKACI)

Příkazy a odkazy pro Nabídky

Verze 3

SET ABOUT (TextPoložky; metoda)

Parametr	Typ		Popis
TextPoložky	Řetězec	→	Nový text položky nabídky Nápověda - Co je ...
metoda	Řetězec	→	Metoda která bude volána při volbě z nabídky

Popis

Příkaz SET ABOUT změní texrt položky Co je 4th Dimension v nabídce Nápověda (Windows) nebo Jablíčko (Macintosh) na text *TextPoložka*.

Po provedení tohoto příkazu, když uživatel vybere tuto položku z nabídky, je spuštěna metoda *metoda*. Metoda může otevřít dialog pro informování uživatele o verzi vaší databáze.

Ikona 4th Dimension, číslo verze 4th Dimension, číslo verze 4D Compiler a jednořádkový copyright bude na vrcholu dialogového okna.

Poznámka: Kompilační číslo verze vaší aplikace bude také zobrazeno, pokud máte nastavenu možnost Automatic version ve 4D Compiler.

Příklady

1. Následující příklad nahradí položku nabídky Co je 4th Dimension... textem O Aplikaci.... Metoda ABOUT zobrazí informační okno:

```
SET ABOUT ("O Aplikaci..."; "ABOUT")
```

2. Následující příklad přenastaví Co je 4th Dimension zpět na původní text:

```
SET ABOUT ("Co je 4th Dimension®";"")
```

Příkazy a odkazy pro Nabídky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Menu selected (Označená nabídka)

Příkazy a odkazy pro Nabídky

Verze 3

Menu selected → Číslo

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Číslo	←	Vybraná nabídka číslo nabídky ve vrchním slovu číslo položky ve spodním slovu

Popis

Příkaz Menu selected se používá pouze když je zobrazený formulář. Zjistí, jaká položka nabídky byla vybrána.

Tip: Kdykoli je to možné, použijte spouštění akcí pomocí metody přiřazené k položce nabídky v přiřazeném záhlaví nabídek (se záporným číslem záhlaví nabídky) místo použití příkazu Menu selected. Přiřazené záhlaví nabídek je , pokud není potřeba testovat jejich výběr, snadnější k ovládání. Nicméně když použijete příkaz APPEND MENU ITEM nebo INSERT MENU ITEM, použijte příkaz Menu selected, protože nové položky nemají přiřazenou metodu.

Menu selected vrací číslo vybrané položky a nabídky v Long Integer. K nalezení čísla nabídky vydělte Menu selected číslem 65536 a převeďte výsledek na celá čísla. K nalezení čísla položky nabídky, vypočítejte modulo z Menu selected s modulus 65536. Použijte následující vzory pro vypočítání čísla nabídky a čísla položky:

Nabídka := **Menu selected** \ 65536
Položka nabídky := **Menu selected** % 65536

Od verze 6 můžete také tato čísla získat pomocí bitových - bitwise operátorů::

Nabídka := (**Menu selected** & 0xFFFF0000) >> 16
Položka nabídky := **Menu selected** & 0xFFFF

Pokud nebyla vybrána žádná položka nabídky, příkaz vrátí 0.

Příklad

Následující metoda formuláře používá Menu selected k zjištění položek nabídek pro předání do příkazu SET MENU ITEM MARK:

```
Case of
   ÷ (Form event=On Menu Selected)
      If (Menu selected # 0)
         SET MENU ITEM MARK (Menu selected\5536;Menu selected%65536 ;Char(18))
      End if
End case
```

Dále si přečtěte

Řízení nabídek.

Count menus (Počet nabídek)

Příkazy a odkazy pro Nabídky

Verze 6.0

Count menus {(proces)} → Číslo

Parametr	Typ		Popis
proces	Číslo	→	Číslo odkazu na proces
Výsledek funkce	Číslo	←	Počet nabídek v platném záhlaví nabídek

Popis

Příkaz Count menus vrátí počet nabídek v záhlaví nabídek.

Pokud vynecháte parametr proces, vrátí se vám počet nabídek v platném záhlaví nabídek. Jinak se Count menus provede na proces jehož číslo zadáte do parametru.

Dále si přečtěte

Count menu items.

Příkazy a odkazy pro Nabídky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Count menu items (Počet položek nabídky)

Příkazy a odkazy pro Nabídky

Verze 6.0

Count menu items (nabídka {; proces}) → Číslo

Parametr	Typ		Popis
nabídka	Číslo	→	Číslo nabídky
proces	Číslo	→	Číslo odkazu na proces
Výsledek funkce	Číslo	←	Počet položek v nabídce

Popis

Příkaz Count menu items vrátí počet položek pro nabídku, jejíž číslo zadáte do parametru *nabídka*.

Pokud vynecháte parametr *proces*, bude příkaz použit na nabídky platného procesu. Jinak bude příkaz použit na nabídky procesu, jehož číslo zadáte.

Dále si přečtěte

Count menus.

Příkazy a odkazy pro Nabídky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Get menu title (Získat název nabídky)

Příkazy a odkazy pro Nabídky

Verze 6.0

Get menu title (nabídka{; proces}) → Řetězec

Parametr	Typ		Popis
nabídka	Číslo	→	Číslo nabídky
proces	Číslo	→	Číslo odkazu na proces
Výsledek funkce	Řetězec	←	Titulek nabídky

Popis

Příkaz Get menu title vrací název nabídky, jejíž číslo jste zadali do parametru *nabídka*.

Pokud vynecháte parametr *proces*, bude příkaz použit na nabídky platného procesu. Jinak bude příkaz použit na nabídky procesu, jehož číslo zadáte.

Dále si přečtěte

Count menus.

Příkazy a odkazy pro Nabídky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Get menu item (Získat text položky nabídky)

Příkazy a odkazy pro Nabídky

Verze 3

Get menu item (nabídka; položkaNab{; proces}) → Řetězec

Parametr	Typ		Popis
nabídka	Číslo	→	Číslo nabídky
položkaNab	Číslo	→	Číslo položky nabídky
proces	Číslo	→	Číslo odkazu na proces
Výsledek funkce	Řetězec	←	Text položky nabídky

Popis

Příkaz Get menu item vrací text položky nabídky, jejíž nabídku a číslo položky jste zadali jako parametry.

Pokud vynecháte parametr *proces*, bude příkaz použit na nabídky platného procesu. Jinak bude příkaz použit na nabídky procesu, jehož číslo zadáte.

Dále si přečtět

SET MENU ITEM.

Příkazy a odkazy pro Nabídky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET MENU ITEM (NASTAVIT POLOŽKU NABÍDKY)

Verze 6.0

SET MENU ITEM (Nabídka; PoložkaNab; TextPoložky{; proces})

Parametr	Typ		Popis
nabídka	Číslo	→	Číslo nabídky
PoložkaNab	Číslo	→	Číslo odkazu na položku
TextPoložky	Řetězec	→	Nový text položky nabídky
proces	Číslo	→	Číslo odkazu na proces

Popis

Příkaz SET MENU ITEM změní text položky, jejíž nabídku a číslo položky jste zadali jako parametry, na text zadaný do *TextPoložky*.

Pokud vynecháte parametr *proces*, bude příkaz použit na nabídky platného procesu. Jinak bude příkaz použit na nabídky procesu, jehož číslo zadáte.

Dále si přečtěte

Get menu item.

Příkazy a odkazy pro Nabídky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Get menu item style (Získat styl položky nabídky)

Příkazy a odkazy pro Nabídky

Verze 6.0

Get menu item style (nabídka; položkaNab{; proces}) → Číslo

Parametr	Typ		Popis
nabídka	Číslo	→	Číslo nabídky
položkaNab	Číslo	→	Číslo položky nabídky
proces	Číslo	→	Číslo odkazu na proces
Výsledek funkce	Řetězec	←	Platný styl položky nabídky

Popis

Příkaz Get menu item style vrací styl písma pro položku jejíž číslo nabídky a položky zadáte jako parametry.

Pokud vynecháte parametr *proces*, bude příkaz použit na nabídky platného procesu. Jinak bude příkaz použit na nabídky procesu, jehož číslo zadáte.

Get menu item style vrací kombinaci (součet hodnot položek) následujících předdefinovaných konstant:

Konstanta	*Typ*	*Hodnota*
Plain	*Long Integer*	*0*
Bold	*Long Integer*	*1*
Italic	*Long Integer*	*2*
Underline	*Long Integer*	*4*
Outline	*Long Integer*	*8*
Shadow	*Long Integer*	*16*
Condensed	*Long Integer*	*32*
Extended	*Long Integer*	*64*

Poznámka: Na Windows jsou možné pouze kombinace Bold, Italic a Underline a samozřejmě Plain.

Příklad

K testování jestli je položka nabídky tučně, napíšete:

```
If ((Get menu item style($vlMenu;$vlItem) & Bold)#0)
   `...
End if
```

Dále si přečtěte

SET MENU ITEM STYLE.

SET MENU ITEM STYLE (NASTAVIT STYL POLOŽKY NABÍDKY)

Verze 6.0

SET MENU ITEM STYLE (nabídka; položkaNab; PoložkaStyl {; proces})

Parametr	Typ		Popis
nabídka	Číslo	→	Číslo nabídky
položkaNab	Číslo	→	Číslo položky nabídky
položkaStyl	Číslo	→	Nový styl položky
proces	Číslo	→	Číslo odkazu na proces

Popis

Příkaz SET MENU ITEM STYLE mění styl písma pro položku, jejíž číslo nabídky a položky zadáte jako parametry v *nabídka* a *položkaNab* , na styl definovaný parametrem *položkaStyl*.

Pokud vynecháte parametr *proces*, bude příkaz použit na nabídky platného procesu. Jinak bude příkaz použit na nabídky procesu, jehož číslo zadáte.

Při definování stylu pro položku musíte zadat kombinaci následujících předdefinovaných konstant (jeden nebo součet):

Konstanta	***Typ***	***Hodnota***
Plain	*Long Integer*	*0*
Bold	*Long Integer*	*1*
Italic	*Long Integer*	*2*
Underline	*Long Integer*	*4*
Outline	*Long Integer*	*8*
Shadow	*Long Integer*	*16*
Condensed	*Long Integer*	*32*
Extended	*Long Integer*	*64*

Poznámka: Na Windows jsou možné pouze kombinace Bold, Italic a Underline a samozřejmě Plain.

Dále si přečtěte

Get menu item style.

Příkazy a odkazy pro Nabídky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Get menu item mark (Získat značku položky nabídky)

Příkazy a odkazy pro Nabídky

Verze 6.0

Get menu item mark (nabídka; položkaNab{; proces}) → Řetězec

Parametr	Typ		Popis
nabídka	Číslo	→	Číslo nabídky
položkaNab	Číslo	→	Číslo položky nabídky
proces	Číslo	→	Číslo odkazu na proces
Výsledek funkce	Řetězec	←	Současné označení položky nabídky

Popis

Příkaz Get menu item mark vrací značku současně označující položku nabídky pro položku, jejíž číslo nabídky a položky zadáte jako parametry v *nabídka* a *položkaNab*.

Pokud vynecháte parametr *proces*, bude příkaz použit na nabídky platného procesu. Jinak bude příkaz použit na nabídky procesu, jehož číslo zadáte.

Pokud položka nabídky nemá označení, vrátí Get menu item mark prázdný řetězec.

Poznámka: Podívejte se na diskuzi pro popis značek na Macintoshi a Windows u příkazu SET MENU ITEM MARK.

Příklad

Následující příklad spojí značku položky nabídky:

```
SET MENU ITEM MARK($vlMenu;$vlItem;Char(18)*
        Num(Get menu item mark($vlMenu;$vlItem)=""))
```

Dále si přečtěte

SET MENU ITEM MARK.

Příkazy a odkazy pro Nabídky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET MENU ITEM MARK (NASTAVIT ZNAČKU POLOŽKY NABÍDKY)

Verze 3

SET MENU ITEM MARK (nabídka; položka; značka{; proces})

Parametr	Typ		Popis
nabídka	Číslo	→	Číslo nabídky
položkaNab	Číslo	→	Číslo položky nabídky
značka	Řetězec	→	Nová značka položky
proces	Číslo	→	Číslo odkazu na proces

Popis

Příkaz SET MENU ITEM MARK změní značku položky nabídky, jejíž číslo nabídky a položky zadáte jako parametry, na první znak řetězce předaného parametrem *značka*.

Pokud vynecháte parametr *proces*, bude příkaz použit na nabídky platného procesu. Jinak bude příkaz použit na nabídky procesu, jehož číslo zadáte.

Pokud předáte parmetrem *značka* prázdný řetězec budou odstraněny všechny značky z položky nabídky. Jinak:

- Na Macintoshi obvykle předáte Char (18), což je značka pro položky nabídek na Macintoshi.
- Na Windows je k položce přiřazena standardní značka, bez ohledu na předaný znak.

Příklad

Podívejte se na příklad u příkazu **Get menu item mark**.

Dále si přečtěte

Get menu item mark.

Příkazy a odkazy pro Nabídky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Get menu item key (Získat zkratku k položce nabídky)

Příkazy a odkazy pro Nabídky

Verze 6.0

Get menu item key (nabídka; položkaNab{; proces}) → Číslo

Parametr	Typ		Popis
nabídka	Číslo	→	Číslo nabídky
položkaNab	Číslo	→	Číslo položky nabídky
Proces	Číslo	→	Číslo odkazu na proces
Výsledek funkce	Číslo	←	ASCII kód zkratky položky

Popis

Příkaz Get menu item key vrací ASCII kód zkratky k položce nabídky, jejíž číslo nabídky a položky jste zadali do parametrů *nabídka* a *položkaNab*.

Pokud vynecháte parametr *proces*, bude příkaz použit na nabídky platného procesu. Jinak bude příkaz použit na nabídky procesu, jehož číslo zadáte.

Pokud položka nabídky nemá zkratku, příkaz vrátí 0.

Dále si přečtěte

SET MENU ITEM KEY.

Příkazy a odkazy pro Nabídky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET MENU ITEM KEY (NASTAVIT ZKRATKU K POLOŽCE NABÍDKY)

Verze 6.0

SET MENU ITEM KEY (nabídka; položkaNab; položkaKlíč{; proces})

Parametr	Typ		Popis
nabídka	Číslo	→	Číslo nabídky
položkaNab	Číslo	→	Číslo položky nabídky
PoložkaKlíč	Číslo	→	ASCII kód nové zkratky k položce
Proces	Číslo	→	Číslo odkazu na proces

Popis

Příkaz SET MENU ITEM KEY mění zkratku (vyvolávanou s Ctrl (Windows) nebo Command (Macintosh)) k položce, jejíž číslo nabídky a položky jste zadali do parametrů *nabídka* a *položkaNab*, na klávesu s ASCII kódem zadaným do parametru *položkaKlíč*.

Pokud vynecháte parametr *proces*, bude příkaz použit na nabídky platného procesu. Jinak bude příkaz použit na nabídky procesu, jehož číslo zadáte.

Pokud předáte do parametru *položkaKlíč* 0, zkratka bude odstraněna.

Poznámka: Aby bylo možné tuto zkratku používat, musíte zadat písmeno, číslici nebo jiný znak mimo Ctrl (Windows) nebo Command (Macintosh).

Dále si přečtěte

Get menu item key.

Příkazy a odkazy pro Nabídky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DISABLE MENU ITEM (ZNEPŘÍSTUPNIT POLOŽKU NABÍDKY)

Verze 3

DISABLE MENU ITEM (nabídka; položkaNab{; proces})

Parametr	Typ		Popis
nabídka	Číslo	→	Číslo nabídky
položkaNab	Číslo	→	Číslo položky nabídky
proces	Číslo	→	Číslo odkazu na proces

Popis

Příkaz DISABLE MENU ITEM znepřístupní položku, jejíž číslo nabídky a položky jste zadali do parametrů *nabídka* a *položkaNab*.

Pokud vynecháte parametr *proces*, bude příkaz použit na nabídky platného procesu. Jinak bude příkaz použit na nabídky procesu, jehož číslo zadáte.

Tip: Pokud chcete znepřístupnit všechny položky dané nabídky, vložte do parametru *položkaNab* 0 (nula).

Dále si přečtěte

ENABLE MENU ITEM.

Příkazy a odkazy pro Nabídky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ENABLE MENU ITEM (ZPŘÍSTUPNIT POLOŽKU NABÍDKY)

Verze 3

ENABLE MENU ITEM (nabídka; položkaNab{; proces})

Parametr	Typ		Popis
nabídka	Číslo	→	Číslo nabídky
položkaNab	Číslo	→	Číslo položky nabídky
proces	Číslo	→	Číslo odkazu na proces

Popis

Příkaz ENABLE MENU ITEM zpřístupní znepřístupněnou položku, jejíž číslo nabídky a položky jste zadali do parametrů *nabídka* a *položkaNab*.

Pokud vynecháte parametr *proces*, bude příkaz použit na nabídky platného procesu. Jinak bude příkaz použit na nabídky procesu, jehož číslo zadáte.

Tip: Pokud chcete zpřístupnit všechny položky dané nabídky, vložte do parametru *položkaNab* 0 (nula).

Dále si přečtěte

DISABLE MENU ITEM.

Příkazy a odkazy pro Nabídky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

APPEND MENU ITEM (PŘIPOJIT POLOŽKU NABÍDKY)

Příkazy a odkazy pro Nabídky

Verze 6.0

APPEND MENU ITEM (nabídka; položkaText{; proces})

Parametr	Typ		Popis
nabídka	Číslo	→	Číslo nabídky
položkaText	Řetězec	→	Text nové položky
proces	Číslo	→	Číslo odkazu na proces

Popis

Příkaz APPEND MENU ITEM připojí novou položku k nabídce, jejíž číslo jste zadali do parametru *nabídka*.

Pokud vynecháte parametr *proces*, bude příkaz použit na nabídky platného procesu. Jinak bude příkaz použit na nabídky procesu, jehož číslo zadáte.

APPEND MENU ITEM vám umožní připojit jednu nebo více položek jedním provedením.

Položky k připojení definujete pomocí parametru *položkaText* následujícím způsobem:

• Oddělíte každou položku středníkem (;) následovně: "položkaText1; položkaText2; položkaText3"
• Ke znepřístupnění položky: umístěte otevřené závorky "(" na počátek textu položky.
• K definování oddělovací čáry: vložte "(-" jako text položky.
• K definování stylu položky, vložte do parametru položkaText znaménko menší než (<) následované jedním ze znaků:

<B	*Tučně*
<I	*Kurzíva*
<U	*Podtržené*
<O	*Obrysové (pouze Macintosh)*
<S	*Stínované (pouze Macintosh)*

• K předání značky k položce, vložte vykřičník (!) následovaný znakem který chcete jako značku. Na Macintoshi je znak zobrazen, na Windows je zobrazena standardní značka bez rozdílu jaký znak předáte.
• K předání ikony položky: Do textu položky vložte znak (^) následovaný znakem jehož ASCII kód minus 48 je číslo ID zdroje obrázku (ikony).
• K předání klávesové zkratky na položku: do textu položky vložte lomítko (/) následované znakem pro zkratku.

Poznámka: Používejte rozumný počet položek. Pokud budete mít v položce více než 50 položek, přemýšlejte o posuvné oblasti ve formuláři místo nabídky.

Poznámka: APPEND MENU ITEM přijímá 32000 znaků, zatímco INSERT MENU ITEM pouze 255 znaků.

Důležité: Nová položka nemá přiřazenu žádnou metodu. Proto musí být tyto položky řízeny pomocí příkazu Menu selected z metody formuláře.

Příklad

Tento příkaz přiřadí názvy dostupných písem do nabídky Písmo, která v tomto příkladu je šestá nabídka platného záhlaví:

```
   ` V metodě při spuštění
   ` Seznam písem je vytvořen a texty položek nabídek jsou sestavovány
FONT LIST(<>asDostupnaPisma)
<>atPísmaPoložky:=""
For ($vlFont;1;Size of array(<>asDostupnaPisma))
```

```
    <>atPísmaPoložky:=<>atPísmaPoložky+";"+<>asDostupnaPisma{$vlFont}
End for
```

Po uložení metody do databáze můžete napsat:

```
APPEND MENU ITEM(6;<>atPísmaPoložky)
```

Dále si přečtěte

DELETE MENU ITEM, INSERT MENU ITEM.

Příkazy a odkazy pro Nabídky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

INSERT MENU ITEM (VLOŽIT POLOŽKU NABÍDKY)

Verze 6.0

INSERT MENU ITEM (nabídka; zaPoložku; PoložkaText{; proces})

Parametr	Typ		Popis
nabídka	Číslo	→	Číslo nabídky
položkaNab	Číslo	→	Číslo položky nabídky
položkaText	Řetězec	→	Text nové položky
proces	Číslo	→	Číslo odkazu na proces

Popis

Příkaz INSERT MENU ITEM přidá položku nabídky za položku, jejíž číslo nabídky a položky jste zadali do parametrů *nabídka* a *položkaNab*.

Pokud vynecháte parametr *proces*, bude příkaz použit na nabídky platného procesu. Jinak bude příkaz použit na nabídky procesu, jehož číslo zadáte.

INSERT MENU ITEM vám umožní připojit jednu nebo více položek jedním provedením.

INSERT MENU ITEM je podobná jako APPEND MENU ITEM s následujícími rozdíly:

• INSERT MENU ITEM vám umožní vložit položku kamkoli chcete, zatímco APPEND MENU ITEM vždy přidává na konec.
• S INSERT MENU ITEM je počet znaků pro položku omezen na 255, zatímco APPEND MENU ITEM má možnost až 3200 znaků.

Podívejte se na popis k příkazu APPEND MENU ITEM pro informace o přidávání položek.

Důležité: Nová položka nemá přiřazenu žádnou metodu. Proto musí být tyto položky řízeny pomocí příkazu Menu selected z metody formuláře.

Dále si přečtěte

APPEND MENU ITEM.

Příkazy a odkazy pro Nabídky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DELETE MENU ITEM (VYMAZAT POLOŽKU NABÍDKY)

Verze 6.0

DELETE MENU ITEM (nabídka; položkaNab{; proces})

Parametr	Typ		Popis
nabídka	Číslo	→	Číslo nabídky
položkaNab	Číslo	→	Číslo položky nabídky
proces	Číslo	→	Číslo odkazu na proces

Popis

Příkaz DELETE MENU ITEM vymaže položku, jejíž číslo nabídky a položky jste zadali do parametrů *nabídka* a *položkaNab*.

Pokud vynecháte parametr *proces*, bude příkaz použit na nabídky platného procesu. Jinak bude příkaz použit na nabídky procesu, jehož číslo zadáte.

Poznámka: Pro přehlednost nenechávejte žádnou nabídku bez položek.

Dále si přečtěte

APPEND MENU ITEM, INSERT MENU ITEM.

Příkazy a odkazy pro Nabídky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

MESSAGES OFF (VYPNOUT HLÁŠENÍ)

Příkazy a odkazy pro Hlášení

Verze 3

MESSAGES OFF

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz MESSAGES OFF a MESSAGES ON vypínají a zapínají ukazatele průběhu když 4th Dimension provádí časově náročné úkoly. Jako výchozí je zobrazení zpráv zapnuto.

Následující tabulka ukazuje seznam operací které zobrazují ukazatele průběhu:

Užít výraz	*Rychlé zprávy*	*Třídění*
Export dat	*Import dat*	*Diagramy*
Dotaz dle formuláře	*Dotaz dle výrazu*	*Editor dotazů*

Následující tabulka ukazuje příkazy které zobrazují ukazatele průběhu:

APPLY TO SELECTION	*IMPORT SYLK*	*QUERY*
DISTINCT VALUES	*IMPORT TEXT*	*QUERY BY FORMULA*
EXPORT DIF	*RELATE MANY*	*QUERY BY EXAMPLE*
EXPORT SYLK	*RELATE ONE*	*QUERY SELECTION*
EXPORT TEXT	*REDUCE SELECTION*	*QUERY SELECTION BY FORMULA*
GRAPH TABLE	*REPORT*	*ORDER BY FORMULA*
IMPORT DIF	*SCAN INDEX*	*ORDER BY*

Příklad

Následující příklad vypne ukazatele průběhu před provedením třídění a opět je zapne po dokončení akce:

```
MESSAGES OFF
ORDER BY ([Adresy];[Adresy]PSČ;>;[Adresy]Název2;>)
MESSAGES ON
```

Příkazy a odkazy pro Hlášení

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

MESSAGES ON (ZAPNOUT HLÁŠENÍ)

Verze 3

MESSAGES ON

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Podívejte se na popis u příkazu MESSAGES OFF.

Příkazy a odkazy pro Hlášení

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ALERT (UPOZORNIT)

Příkazy a odkazy pro Hlášení

Verze 6.0 (Upraveno)

ALERT (zpráva {; text tlačítka OK})

Parametr	Typ		Popis
zpráva	Řetězec	→	Zpráva k zobrazení ve výstražném okně
text tlačítka OK	Řetězec	→	Název tlačítka OK

Popis

Příkaz ALERT zobrazí výstražné okno složené z ikony, textu a tlačítka OK.

Zprávu předáte do parametru zpráva a může obsahovat až 255 znaků. Pokud se zpráva nevejde do okna,může být oříznuta. Záleží to na délce textu a výšce znaků.

Jako výchozí je nastaven text tlačítka OK na "OK". Pokud chcete tento text změnit, zadejte novou hodnotu do volitelného parametru text tlačítka OK. Pokud je to třeba, velikost tlačítka je zvětšena vlevo. podle velikosti textu který předáte.

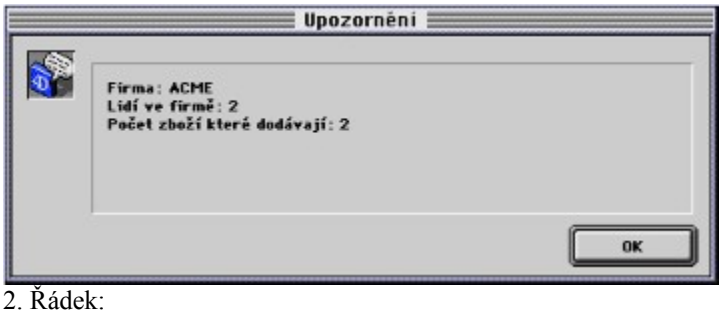
Tip: Nevkládejte příkaz ALERT do části metody formuláře nebo objektu, která pracuje s událostmi Při zavedení a Při vyvedení. Způsobili by jste tím nekonečnou smyčku.

Příklady

1. Tento příklad zobrazí výstražné okno ukazující informace o firmě. Všiměte si že řetězec k zobrazení obsahuje return, což umožní přesunout další část textu na další řádek:

```
ALERT("Firma: "+[Companies]Název+Char(13)+"Lidí ve firmě: "+
   String(Records in selection([Lidé]))+Char(13)+"Počet zboží které dodávají: "+
   String (Records in selection([Parts])))
```

Zobrazí se následující zpráva (na Macintosh)



2. Řádek:

ALERT ("Omlouvám se Davide. Neudělám to.";"Aleš")

zobrazí následující okno (na Macintosh):

3. Řádek:

ALERT ("Již nemáte právo mazat záznamy";"Bohužel jsem to nevěděl")

zobrazí následující okno (na Macintosh):

Dále si přečtěte

CONFIRM, Request.

Příkazy a odkazy pro Hlášení

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CONFIRM (POTVRDIT)

Příkazy a odkazy pro Hlášení

Verze 6.0 (Upraveno)

CONFIRM (zpráva{; titul tlačítka OK{; titul tlačítka Storno}})

Parametr	Typ		Popis
zpráva	Řetězec	→	Zpráva k zobrazení v okně pro potvrzení
titul tlačítka OK	Řetězec	→	Text tlačítka OK
titul tlačítka Storno	Řetězec	→	Text tlačítka Storno

Popis

Příkaz CONFIRM zobrazí okno pro potvrzení zprávy složené z ikony, zprávy, tlačítka OK a tlačítka Storno.

Text zprávy předáte do parametru *zprávy*. Tato zpráva může být maximálně 255 znaků dlouhá. Pokud se zpráva nevejde do okna, může být oříznuta. Záleží to na délce textu a výšce znaků.

Jako výchozí je text tlačítka OK "OK" a tlačítka Storno "Storno". K upravení těchto hodnot vložte nové texty do parametrů titul tlačítka OK a titul tlačítka Storno. Pokud to bude nutné, 4th Dimension upraví velikost těchto tlačítek.

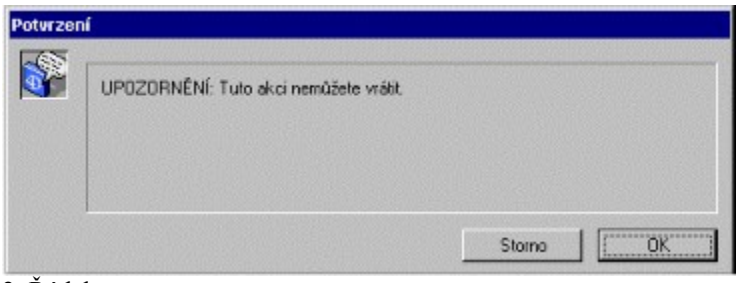
Tip: Nevkládejte příkaz CONFIRM do části metody formuláře nebo objektu, která pracuje s událostmi Při zavedení a Při vyvedení. Způsobili by jste tím nekonečnou smyčku.

Příklady

1. Následující kód:

```
CONFIRM("UPOZORNĚNÍ: Tuto akci nemůžete vrátit.")
If (OK=1)
   ALL RECORDS([Old Stuff])
   DELETE SELECTION([Old Stuff])
Else
   ALERT ("Akce zrušena.")
End if
```

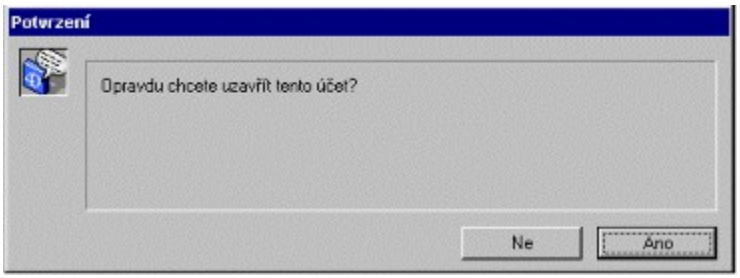
zobrazí následující okno potvrzení (na Windows):



2. Řádek:

```
CONFIRM ("Opravdu chcete uzavřít tento účet?";"Ano";"Ne")
```

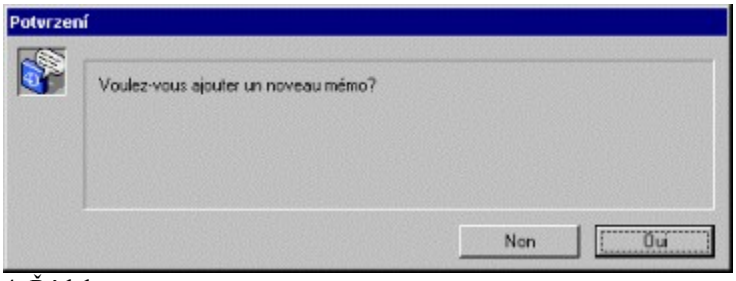
zobrazí následující okno (na Windows):



3. Vytváříte aplikaci ve 4D pro mezinárodní trh. Napíšete metodu projektu která bude vracet správně lokalizovaný text z české verze. Máte také vytvořený array s názvem <>asPřeloženéZprávy, který obsahuje nejpoužívanější slova. Pokud to uděláte, řádek:

CONFIRM(*CZ Text* ("Opravdu chcete vložit novou poznámku?");
<>asPřeloženéZprávy {kLoc_YES};<>asPřeloženéZprávy {kLoc_NO})

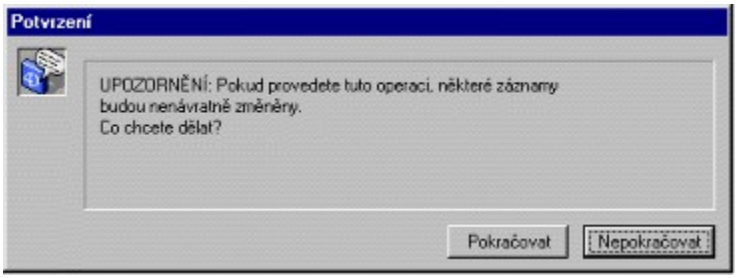
Zobrazí následující francouzské okno pro potvrzení (na Windows):



4. Řádek:

CONFIRM("UPOZORNĚNÍ: Pokud provedete tuto operaci, některé záznamy "+ **Char**(13)+
"budou nenávratně změněny."+**Char**(13)+"Co chcete dělat?";"Nepokračovat"; "Pokračovat")

zobrazí následující okno (na Windows):



Dále si přečtěte

ALERT, Request.

Příkazy a odkazy pro Hlášení

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Request (Žádost)

Příkazy a odkazy pro Hlášení

Verze 6.0 (Upraveno)

Request (zpráva{; výchozí odpověď{; titul OK{; titul Storno}}}) → Řetězec

Parametr	Typ		Popis
zpráva	Řetězec	→	zpráva zobrazená v okně požadavku
výchozí odpověď	Řetězec	→	výchozí text zobrazený ve vstupní oblasti požadavku
titul OK	Řetězec	→	Text tlačítka OK
titul Storno	Řetězec	→	Text tlačítka Storno

Popis

Příkaz Request zobrazí okno žádosti složené ze zprávy, textové oblasti, tlačítka OK a tlačítka Storno.

Text zprávy předáte do parametru zprávy. Tato zpráva může být maximálně 255 znaků dlouhá. Pokud se zpráva nevejde do okna, může být oříznuta. Záleží to na délce textu a výšce znaků.

Jako výchozí je text tlačítka OK "OK" a tlačítka Storno "Storno". K upravení těchto hodnot vložte nové texty do parametrů titul tlačítka OK a titul tlačítka Storno. Pokud to bude nutné, 4th Dimension upraví velikost těchto tlačítek.

Tlačítko OK je výchozí tlačítko. Jestliže klepnete na tlačítko OK nebo stisknete klávesu Enter pro potvrzení dialogu, systémová proměnná OK je nastavena na 1. Pokud klepnete na Zrušit, je proměnná nastavena na 0.

Uživatel může vložit text do vstupní textové oblasti. K definování výchozí hodnoty, vložte tuto hodnotu do parametru výchozí odpověď. Pokud uživatel klepne na OK, Request vrátí text. Pokud uživatel klepne na Storno, je vrácen prázdný řetězec.Pokud má být výsledek číslo nebo datum, převeďte tuto hodnotu pomocí patřičných příkazů, Num nebo Date do správného formátu.

Tip: Nevkládejte příkaz Request do části metody formuláře nebo objektu, která pracuje s událostmi Při zavedení a Při vyvedení. Způsobili by jste tím nekonečnou smyčku.

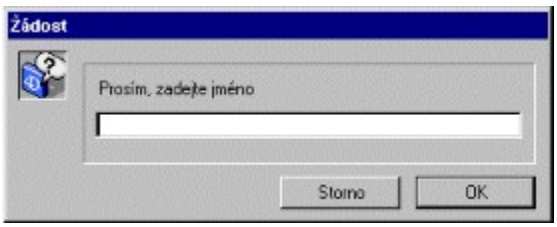
Tip: Pokud potřebujete od uživatele získat více informací, navrhněte formulář a zobrazte jej pomocí příkazu DIALOG. Je to jednodušší než několikrát volat Request.

Příklady

1. Řádek:

```
$vsJméno:= Request ("Prosím, zadejte jméno:")
```

zobrazí následující okno (na Windows):

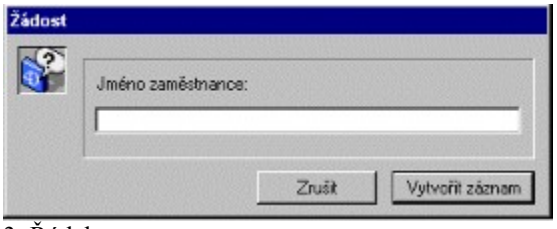


2. Následující kód:

```
vsOdezva := Request ("Jméno zaměstnance:";"";"Vytvořit záznam";"Zrušit")
If (OK=1)
   ADD RECORD ([Zaměstnanci])
```

```
      ` Poznámka: vsOdezva je kopírován do pole [Zaměstnanci]Příjmení
   End if
```

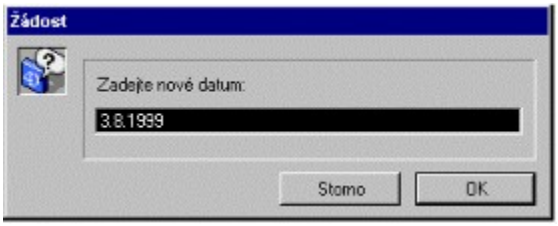
zobrazí následující okno (na Windows):



3. Řádek:

```
   $vdOdezva := Date (Request ("Zadejte nové datum:";String (Current date)))
```

zobrazí následující okno (na Windows):



Dále si přečtěte

ALERT, CONFIRM.

Příkazy a odkazy pro Hlášení

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

MESSAGE (HLÁŠENÍ)

Příkazy a odkazy pro Hlášení

Verze 3

MESSAGE (zpáva)

Parametr	Typ		Popis
zpráva	Řetězec	→	Zpráva k zobrazení

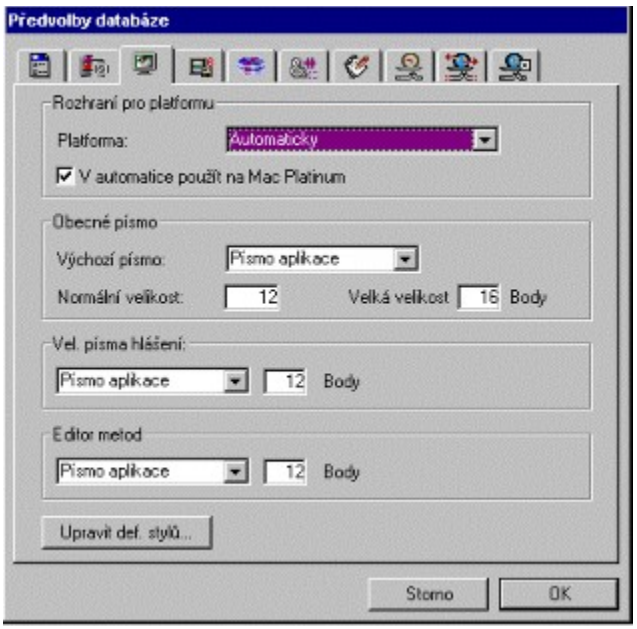
Popis

Příkaz MESSAGE se obvykle používá k informování uživatele o nějaké aktivitě. Zobrazí zprávu ve specielním okně které se otevře a zavře pokaždé, když voláte příkaz MESSAGE, dokud pracujete s oknem které bylo předtím otevřeno pomocí Open window (čtěte dále). Zpráva je dočasná a je vymazána ihned jak je zobrazen formulář nebo se dokončí metoda. Pokud je volán jiný příkaz MESSAGE, je stará zpráva přemazána.

Pokud je okno otevřeno pomocí Open window, všechny následující příkazy MESSAGE zobrazí svou zprávu v tomto pkně. Okno se chová jako terminál:

- Předchozí zprávy nejsou přemazávány pokud je zobrazeno okno. Místo toho jsou připojeny k existující zprávě.
- Pokud je text větší než zobrazené okno, 4th Dimension automaticky provede rozdělení textu.
- Pokud má text více řádků než okno, 4th Dimension automaticky posune okno zprávy.
- Ke kontrole zlomů textu, vložte return - char (13) - do vaší zprávy
- Pokud chcete text zobrazit na určitém místě v okně, provedte příkaz GOTO XY.
- K vymazání obsahu okna, proveďte příkaz ERASE WINDOW.
- Okno je pouze výstupní a nepřekreslí se pokud jej překryjí jiná okna.

4th Dimension obsahuje předvolby Velikost písma hlášení pro definování zobrazení zpráv. Toto nastavení můžete změnit v Předvolbách databáze.



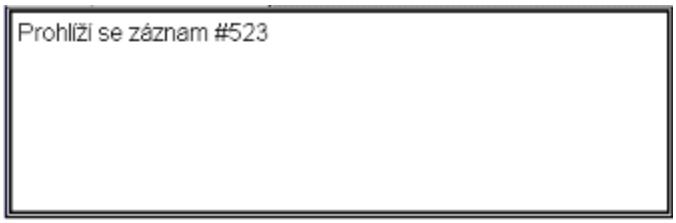
Můžete zvolit písmo a jeho velikost (v limitech) podle vašich potřeb. Pokud použijete MESSAGE a GOTO XY, je dobré použít pevnou výšku písma jako Terminal na Windows a Monaco na Macintosh.

Příklady

1. Následující příkad projde výběr záznamů a použije MESSAGE k informování uživatele o průběhu:

```
For($vlZaznam;1;Records in selection([maTabulka]))
   MESSAGE ("Prohlíží se záznam #"+String($vlZaznam))
      ` Udělat něco se záznamem
   NEXT RECORD([maTabulka])
End for
```

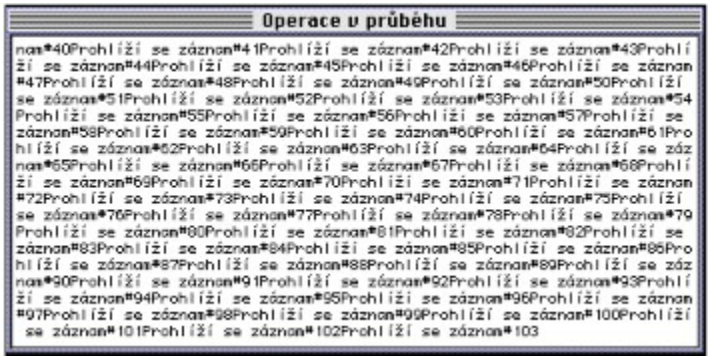
Zobrazí se a zavře následující okno při každém volání MESSAGE:



2. Aby jste se vyhnuli "blikajícímu" oknu, můžete zprávu zobrazit do okna, které se otevřete pomocí příkazu Open window:

```
Open window(50;50;500;250;5;"Operace v průběhu")
For($vlZaznam;1;Records in selection([maTabulka]))
   MESSAGE ("Prohlíží se záznam #"+String($vlZaznam))
      ` Udělat něco se záznamem
   NEXT RECORD([maTabulka])
End for
CLOSE WINDOW
```

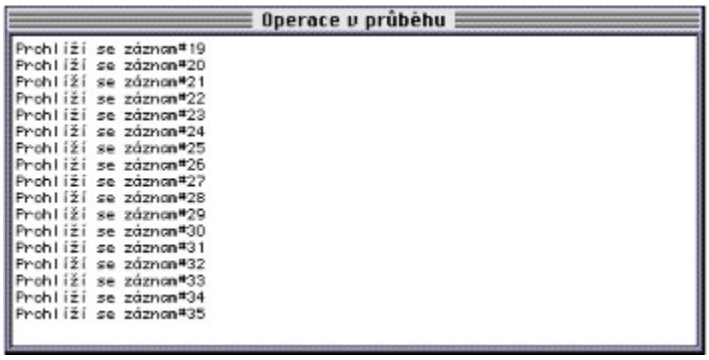
Výsledek bude vypadat následovně (na Macintoshi):



3. Po předání return bude vše přehlednější:

```
Open window(50;50;500;250;5;"Operace v průběhu")
For($vlZaznam;1;Records in selection([maTabulka]))
   MESSAGE ("Prohlíží se záznam #"+String($vlZaznam)+Char(13))
      ` Udělat něco se záznamem
   NEXT RECORD([maTabulka])
End for
CLOSE WINDOW
```

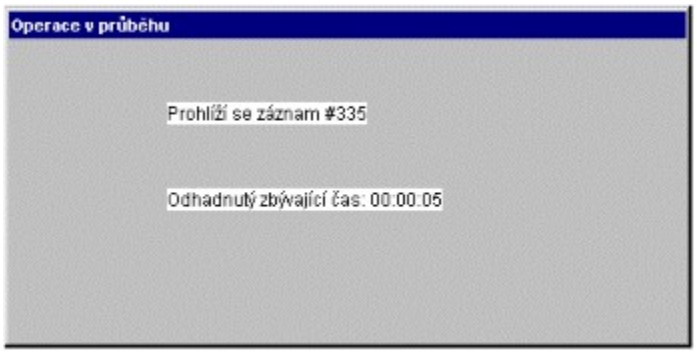
Výsledek bude vypadat následovně (na Macintoshi):



4. Použití GOTO XY a předání dalších řádků:

```
Open window(50;50;500;250;5;"Operace v běhu")
$vlPocZazn:=Records in selection([maTabulka])
$vhStartTime:=Current time
For($vlZaznam;1;$vlPocZazn)
   GOTO XY(5;2)
   MESSAGE ("Prohlíží se záznam #"+String($vlZaznam)+Char(13))
      ` Udělat něco se záznamem
   NEXT RECORD([maTabulka])
   GOTO XY(5;5)
   $vlRemaining:=(($vlPocZazn/$vlZaznam)-1)*( Current time-$vhStartTime)
   MESSAGE ("Odhadnutý zbývající čas: "+Time string($vlRemaining))
End for
CLOSE WINDOW
```

Výsledek bude vypadat následovně (na Windows):



Dále si přečtěte

CLOSE WINDOW, ERASE WINDOW, GOTO XY, Open window.

Příkazy a odkazy pro Hlášení

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GOTO XY (JÍT NA XY)

Verze 3

GOTO XY (x;y)

Parametr	Typ		Popis
x	Číslo	→	x (vodorovná) pozice kurzoru
y	Číslo	→	y (svislá) pozice kurzoru

Popis

Příkaz GOTO XY je používán společně s příkazem MESSAGE když zobrazujete zprávu do okna otevřeného pomocí Open window.

GOTO XY umístí znakový kurzor (neviditelný kurzor) k nastavení umístění další zprávy v okně.

Horní levý roh je pozice 0,0. Při otevření okna a při provedení příkazu ERASE WINDOW je kurzor automaticky umístěn na pozici 0,0.

Jakmile GOTO XY umístí kurzor, můžete použít MESSAGE k zobrazení textu hlášení.

Tip: S použitím pevné výšky písma, jako Terminal na Windows a Monaco na Macintoshi je zobrazení hlášení s pomocí GOTO XY a MESSAGE nejlepší. Podívejte se na popis u příkazu MESSAGE pro více informací.

Příklady

1. Podívejte se na příklad u příkazu MESSAGE.

2. Podívejte se na příklad u příkazu Milliseconds.

3. Následující příklad:

```
Open window(50;50;300;300;5;"Toto je pouze test")
For ($vlRow;0;9)
   GOTO XY($vlRow;0)
   MESSAGE(String($vlRow))
End for
For ($vlLine;0;9)
   GOTO XY(0;$vlLine)
   MESSAGE(String($vlLine))
End for
$vhStartTime:=Current time
Repeat
Until ((Current time-$vhStartTime)>†00:00:30†)
```

Dále si přečtěte

MESSAGE.

Pojmenovaný výběr

Příkazy a odkazy pro Pojmenované výběry

Verze 3

Pojmenovaný vyběr je jednoduchý způsob pro práci s více výběry najednou. Pojmenovaný výběr je tříděný seznam záznamů pro nějakou tabulku v určitém procesu. Tomuto seznamu může být dán název a lze jej uložit do paměti. Je to snadný způsob jak uložit do paměti informaci o konkrétních vybraných záznamech. pořadí záznamů ve výběru a platném záznamu výběru.

Následující příklady umožňují práci s pojmenovaným výběrem:

- COPY NAMED SELECTION
- CUT NAMED SELECTION
- USE NAMED SELECTION
- CLEAR NAMED SELECTION
- CREATE SELECTION FROM ARRAY

Pojmenované výběry jsou vytvářeny pomocí příkazů COPY NAMED SELECTION, CUT NAMED SELECTION a CREATE SELECTION FROM ARRAY. Pojmenované výběry jsou použity hlavně k práci s jedním a více výběry a k uložení a pozdějšímu načtení tříděných výběrů. Pro každou tabulku a každý proces může existovat mnoho pojmenovaných výběrů. K ořevedení pojmenovaného výběru do platného výběru, použijte příkaz USE NAMED SELECTION. Pokud jste skončili práci s tímto výběrem, vymažte jej pomocí příkazu CLEAR NAMED SELECTION.

Pojmenované výběry mohou být procesní nebo meziprocesní.

Pojmenovaný výběr je meziprocesní, pokud začíná znaky <> - "menší než" následované "větší než".

Poznámka: Tento typ zápisu může být použit jak na Windows tak na Macintoshi. Na Macintoshi máte ještě možnost použít diamant (Option-Shift-V na US klávesnici).

Rozsah platnosti meziprocesního pojmenovaného výběru je stejný jako rozsah platnosti meziprocesní proměnné. Tento výběr může být použit ve všech procesech dané stanice.

Pojmenovaný výběr jehož název nezačíná znaky (<>) je procesní a může být použit pouze v procesu ve kterém byl vytvořen.

Ve struktuře klint/server je meziprocesní výběr použitelný pouze na počítači, kde byl vytvořen. Nemůže jej používat jiný klient.

Upozornění: Vytvoření pojmenovanáho výběru si vyžaduje přístup k výběrům tabulky. Vzhledem k tomu, že výběry jsou uložené na serveru a místní proces nemá na server přístup, nepoužívejte pojmenované výběry v místních procesech.

Pojmenované výběry a Sady

Rozdíly mezi pojmenovaným výběrem a sadou (set) jsou:

- Pojmenovaný výběr je tříděný seznam záznamů; sada ne.
- Sady velmi šetří paměť, neboť potřebují pouze jeden bit pro každý záznam v souboru. Pojmenovaný výběr potřebuje 4 byty pro každý záznam ve výběru.
- Na rozdíl od sady, nemůže být pojmenovaný výběr uložen na disk.
- Na sady mohou provedeny množinové operace, sjednocení , průnik bitový součet….; pojmenovaný výběr nemůže být kombinován s jiným pojmenovaným výběrem.

Pojmenovaný výběr a Sada mají společné následující:

• Stejně jako sada je i pojmenovaný výběr uložený v paměti.
• Pojmenovaný výběr a sada obsahují odkazy k záznamům. Pokud byl záznam upraven nebo vymazán, pojmenovaný výběr nebo sada mohou být neplatné.
• Stejně jako sada si pojmenovaný výběr "pamatuje, který záznam byl platný ve chvíli uložení.

O sadách se dozvíte v kapitole Příkazy a odkazy pro Sady

Příkazy a odkazy pro Pojmenované výběry

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

COPY NAMED SELECTION (KOPÍROVAT POJMENOVANÝ VÝBĚR)

Příkazy a odkazy pro Pojmenované výběry

Verze 3

COPY NAMED SELECTION ({tabulka;} název)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ze které kopírovat výběr, nebo Výchozí tabulka pokud vynecháno
název	Řetězec	→	Název pojmenovaného výběru

Popis

COPY NAMED SELECTION kopíruje platný výběr do pojmenovaného výběru název. Je použita výchozí tabulka procesu, pokud není předán parametr tabulka, kde se definuje tabulka. Parametr název obsahuje kopii výběru. Platný výběr a platný záznam pro tabulku nejsou změněny.

Pojmenovaný výběr neobsahuje záznamy ale pouze odkazy k záznamům. Každý odkaz na záznam zabere 4 byty paměti. To znamená že pokud je pojmenovaný výběr vytvořený příkazem COPY NAMED SELECTION, je velikost paměti potřebná k jeho uložení 4 byty krát počet obsahovaných záznamů. Protože pojmenovaný výběr je uložen v paměti, nemusíte míst dost paměti pro uložení pojmenovaného výběru a platného výběru procesu pro tabulku.

Použijte příkaz CLEAR NAMED SELECTION k uvolnění paměti zabrané výběrem název.

Příklad

Tento příklad umožní zjistit, jestli existují nějaké platné faktury v tabulce [Lidé]. Výběr je setříděn a pak uložen. Hledáme všechny záznamy, kde faktury jsou splatné. Pak znovu použijeme výběr a vymažeme pojmenovaný výběr z paměti. Vymazání pojmenovaného výběru je volitelné, v některých případech může chtít návrhář databáze ponechat tříděný výběr pro další použití:

```
ALL RECORDS([Lidé])
   `Umožnit uživateli třídit záznamy
ORDER BY([Lidé])
` Uložit tříděný výběr jako pojmenovaný výběr
COPY NAMED SELECTION([Lidé];"UserSort")
   ` Hledat faktury které jsou splatné
QUERY([Lidé];[Lidé]InvoiceDue=True)
   ` Pokud jsou nalezeny záznamy
If (Records in selection([Lidé])>0)
      ` Upozornit uživatele
   ALERT("Ano existují faktury po splatnosti.")
End if
   ` Znovu použít tříděný pojmenovaný výběr
USE NAMED SELECTION("UserSort")
   ` Odstranit výběr z paměti
CLEAR NAMED SELECTION("UserSort")
```

Dále si přečtěte

CLEAR NAMED SELECTION, CUT NAMED SELECTION, USE NAMED SELECTION.

Příkazy a odkazy pro Pojmenované výběry

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CUT NAMED SELECTION (VYJMOUT POJMENOVANÝ VÝBĚR)

Verze 3

CUT NAMED SELECTION ({tabulka;} název)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ze které kopírovat výběr, nebo Výchozí tabulka pokud vynecháno
název	Řetězec	→	Název pojmenovaného výběru

Popis

CUT NAMED SELECTION vytvoří pojmenovaný výběr a přesune do něj platný výběr tabulky. Rozdíl mezi tímto příkazem a COPY NAMED SELECTION je vtom, že tento příkaz nekopíruje výběr ale přesune samotný výběr.

Po provedení příkazu je platný výběr prázdný. Proto nemůže být použit CUT NAMED SELECTION když se upravuje záznam.

CUT NAMED SELECTION šetří více paměti než COPY NAMED SELECTION. S COPY NAMED SELECTION je potřeba 4 byty na záznam duplikovaný v paměti. Pomocí CUT NAMED SELECTION je přesunut pouze odkaz na výběr.

Příklad

Následující příklad vyprázdní platný výběr pro tabulku [Zákazníci]:

```
CUT NAMED SELECTION([Zákazníci]; "ToBeCleared")
CLEAR NAMED SELECTION("ToBeCleared")
```

Dále si přečtěte

CLEAR NAMED SELECTION, COPY NAMED SELECTION, USE NAMED SELECTION.

Příkazy a odkazy pro Pojmenované výběry

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

USE NAMED SELECTION (POUŽÍT POJMENOVANÝ VÝBĚR)

Verze 3

USE NAMED SELECTION (název)

Parametr	Typ		Popis
název	Řetězec	→	Název pojmenovaného výběru

Popis

Příkaz USE NAMED SELECTION použije pojmenovaný výběr jako platný výběr pro tabulku ke které patří.

Při vytvoření pojmenovaného výběru je zapamatován platný záznam. USE NAMED SELECTION najde a zjistí pozici tohoto záznamu a udělá z něj znovu platný záznam; tento příkaz načte platný záznam. Pokud byl tento záznam upraven po vytvoření pojmenovaného výběru, měl by být uložen před použitím USE NAMED SELECTION aby se zabránilo případné ztrátě upravených informací.

• Pokud byl pojmenovaný výběr vytvořen pomocí COPY NAMED SELECTION, je výběr znovu zkopírován do platného výběru. Pojmenovaný výběr název bude existovat v paměti dokud jej nevymažete. Použijte příkaz CLEAR NAMED SELECTION k vymazání pojmenovaného výběru a uvolnění paměti.
• Pokud byl k vytvoření pojmenovaného výběru použít příkaz CUT NAMED SELECTION, je platný výběr nastaven na název a název přestane existovat.

Zapamatujte si, že pojmenovaný výběr je zástupce výběru který byl v okamžiku vytvoření pojmenovaného výběru. Pokud byly některé záznamy změněny, nemusí již být pojmenovaný výběr přesný. Proto používejte pojmenovaný výběr raději na záznamy, které se nemění nijak často. Existuje mnoho věcí které mohou zrušit platnost pojmenovaného výběru: upravení záznamů pojmenovaného výběru, vymazání záznamů pojmenovaného výběru nebo změna kritérií které určují pojmenovaný výběr.

Poznamenejte si, že během transakce jsou použity dočasné adresy k záznamům. Pokud je pojmenovaný výběr vytvořen během transakce, může obsahovat adresy, které nebudou platit jakmile bude transakce potvrzena nebo zrušena, protože záznamy obdrží zpět svoje originální adresy.

Dále si přečtěte

COPY NAMED SELECTION, CUT NAMED SELECTION, USE NAMED SELECTION.

Příkazy a odkazy pro Pojmenované výběry

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CLEAR NAMED SELECTION (VYMAZAT POJMENOVANÝ VÝBĚR)

Verze 3

CLEAR NAMED SELECTION (název)

Parametr	Typ		Popis
název	Řetězec	→	Název pojmenovaného výběru

Popis

CLEAR NAMED SELECTION vymaže pojmenovaný výběr z paměti a uvolní ji. Nemá žádný vliv na tabulky, záznamy nebo výběry. Vzhledem k tomu že pojmenovaný výběr je umístěn v paměti, je důležité vymazat pojmenovaný výběr pokud jej již nepotřebujete.

Pokud byl pojmenovaný výběr vytvořen pomocí příkazu CUT NAMED SELECTION a pak používán pomocí USE NAMED SELECTION, již v paměti neexistuje. V tomto případě příkaz CLEAR NAMED SELECTION neprovede nic.

Dále si přečtěte

COPY NAMED SELECTION, CUT NAMED SELECTION, USE NAMED SELECTION.

Příkazy a odkazy pro Pojmenované výběry

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CREATE SELECTION FROM ARRAY (VYTVOŘIT VÝBĚR Z ARRAY)

Příkazy a odkazy pro Pojmenované výběry

Verze 6.5

CREATE SELECTION FROM ARRAY ({tabulka;} záznArray{; název})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ze které kopírovat výběr, nebo Výchozí tabulka pokud vynecháno
záznArray	LongInt | Logický array	→	Array čísel záznamů v databázi, nebo Array logické (True = záznam je ve výběru, False = záznam není ve výběru)
název	Řetězec	→	Název pojmenovaného výběru

Popis

Příkaz CREATE SELECTION FROM ARRAY vytvoří pojmenovaný výběr název z:

- z array čísel záznamů
- z logického array. V tomto případě definuje hodnota prvků jestli je (True) nebo neni (False) záznam ve výběru.

Pokud nepředáte parametr název nebo pokud předáte prázdný řetězec, příkaz bude použit na platný výběr, který bude upraven.

Pokud použijete array LongInt hodnot všechna čísla v tomto array udávají záznamy. Pokud je toto číslo špatně (záznam nebyl vytvořen), je generována chyba -10503.

Pokud použijete array logických hodnot, N-tý prvek v array značí jestli záznam N je (True) nebo není (False) ve výběru. Počet prvků v array musí být stejný jako počet záznamů v tabulce. Pokud je array menší než počet záznamů, jenom záznamy definované v array budou ve výběru.

Poznámka: S array logických hodnot použije příkaz prvky od 0 do N-1.

Upozornění: Pojmenovaný výběr je vytvořen a načten do paměti. Proto se ujistěte, že máte dostatek paměti.

Dále si přečtěte

CLEAR NAMED SELECTION, COPY NAMED SELECTION, CREATE SET FROM ARRAY, USE NAMED SELECTION.

Příkazy a odkazy pro Pojmenované výběry

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Vlastnosti objektů

Příkazy a odkazy pro Vlastnosti objektů

Verze 6.0

Příkazy vlastností objektu

Příkazy Vlastností objektu jsou:

- FONT
- FONT SIZE
- FONT STYLE
- ENABLE BUTTON
- DISABLE BUTTON
- BUTTON TEXT
- SET CHOICE LIST
- SET ENTERABLE
- SET VISIBLE
- SET FORMAT
- SET FILTER
- SET COLOR
- SET RGB COLORS

Příkazy vlastností objektu jsou určeny pro upravování vlastností objektů ve formuláři. Umožní vám změnit vzhled a chování objektu při používání formuláře v prostředí Uživatele nebo Vlastních nabídek.

Důležité: Rozsah těchto příkazů je formulář který je právě používán: změny budou zrušeny při uzavření formuláře.

Přístup k objektům pomocí jejich názvu nebo názvu datového zdroje

Příkazy Vlastností objektu používají stejný generický zápis jak je ukázáno zde:

COMMAND NAME({*; }objekt { ; dodatečné parametry pro každý příkaz})

Pokud definujete parametr *, definujete název objektu (řetězec).

Můžete použít znak @ v názvu objektu, jestliže chcete měnit více objektů jedním příkazem. Následující tabulka ukazuje příklady názvů objektu které můžete použít s tímto příkazem.

Název objektu	**Objekty které budou ovlivněny příkazem**
HlavníSkupinaObj	Pouze objekt HlavníSkupinaObj
hlavní@	Objekty jejichž název začíná na "hlavní"
@SkupinaObj	Objekty jejichž název končí na "SkupinaObj"
@Skupina@	Objekty jejichž název obsahuje "Skupina"
hlavní@Obj	Objekty jejichž název začíná na "hlavní" a končí na "Obj"
@	Všechny objekty ve formuláři

Pokud vynecháte volitelný parametr *, označíte tím pole nebo proměnnou v objektu. V tomto případě definujete odkaz na pole nebo proměnné (pouze objekty polí nebo proměnných) místo řetězce.

Poznámka: Druhý zápis je kompatibilní s předchozí verzí 4th Dimension.

Dále si přečtěte

BUTTON TEXT, DISABLE BUTTON, ENABLE BUTTON, FONT, FONT SIZE, FONT STYLE, SET CHOICE LIST, SET ENTERABLE, SET FILTER, SET FORMAT, SET RGB COLORS, SET VISIBLE.

Příkazy a odkazy pro Vlastnosti objektů

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

FONT (PÍSMO)

Příkazy a odkazy pro Vlastnosti objektů

Verze 6.0 (Upraveno)

FONT ({*; }objekt; písmo)

Parametr	Typ		Popis
*		→	Pokud definováno je objekt název objektu (řetězec) pokud vynecháno objekt je pole nebo proměnná
objekt	Objekt formuláře	→	Název objektu (pokud * definována) Pole nebo proměnná (pokud * vynechána)
písmo	Řetězec | Číslo	→	Název nebo číslo písma

Popis

Příkaz FONT nastaví pro objekt definovaný v parametru objekt písmo, jehož číslo nebo název předáte do písma.

Pokud definujete parametr * je objekt název objektu. Pokud parametr * vynecháte, objekt bude značit pole nebo proměnnou. V tomto případě definujete odkaz na pole nebo proměnnou (pouze objekty pole nebo proměnné) místo řetězce. Jestli chcete vědět více informací o názvech objektů, podívejte se na část Vlastnosti objektů.

Příklady

1. Následující příklad nastaví písmo pro tlačítko s názvem bOK:

```
FONT (bOK; "Arial")
```

2. Následující příklad nastaví písmo pro všechny objekty jejichž název obsahuje "info":

```
FONT (*; "@info@"; "Times")
```

Dále si přečtěte

FONT SIZE, FONT STYLE.

Příkazy a odkazy pro Vlastnosti objektů

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

FONT SIZE (VELIKOST PÍSMA)

Příkazy a odkazy pro Vlastnosti objektů

Verze 6.0 (Upraveno)

FONT SIZE ({*; }objekt; velikost)

Parametr	Typ		Popis
*		→	Pokud definováno je objekt název objektu (řetězec) pokud vynecháno objekt je pole nebo proměnná
objekt	Objekt formuláře	→	Název objektu (pokud * definována) Pole nebo proměnná (pokud * vynechána)
velikost	Číslo	→	Velikost písma v bodech

Popis

Příkaz FONT SIZE nastaví velikost písma pro objekt definovaný parametrem objekt.

Pokud definujete parametr * je objekt název objektu. Pokud parametr * vynecháte, objekt bude značit pole nebo proměnnou. V tomto případě definujete odkaz na pole nebo proměnnou (pouze objekty pole nebo proměnné) místo řetězce. Jestli chcete vědět více informací o názvech objektů, podívejte se na část Vlastnosti objektů.

Size je celé číslo mezi 1 a 255. Pokud patřičná velikost písma neexistuje, je použita nejbližší hodnota.

Oblast objektu, jak je definovaná ve formuláři, musí být dostatečně velká aby zobrazila data v nové velikosti písma. Jinak může být text oříznut nebo nebude zobrazen celý.

Příklady

1. Následující příklad nastaví velikost písma pro proměnnou vtInfo:

```
FONT SIZE (vtInfo; 14)
```

2. Následující příklad nastaví velikost písma pro všechny objekty ve formuláři jejichž název začíná na "hl":

```
FONT SIZE (*; "hl@"; 14)
```

Dále si přečtěte

FONT, FONT STYLE.

Příkazy a odkazy pro Vlastnosti objektů

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

FONT STYLE (STYL PÍSMA)

Verze 6.0 (Upraveno)

FONT STYLE ({*; }objekt; styl)

Parametr	Typ		Popis
*		→	Pokud definováno je objekt název objektu (řetězec) pokud vynecháno objekt je pole nebo proměnná
objekt	Objekt formuláře	→	Název objektu (pokud * definována) Pole nebo proměnná (pokud * vynechána)
styl	Číslo	→	Styl písma

Popis

Příkaz FONT STYLE nastaví styl objektu který definujete v parametru objekt.

Pokud definujete parametr * je objekt název objektu. Pokud parametr * vynecháte, objekt bude značit pole nebo proměnnou. V tomto případě definujete odkaz na pole nebo proměnnou (pouze objekty pole nebo proměnné) místo řetězce. Jestli chcete vědět více informací o názvech objektů, podívejte se na část Vlastnosti objektů.

Do parametru styl vkládáte součty předdefinovaných konstant. Následující tabulka ukazuje tyto předdefinované konstanty:

Konstanta	***Typ***	***Hodnota***
Plain	*Long Integer*	*0*
Bold	*Long Integer*	*1*
Italic	*Long Integer*	*2*
Underline	*Long Integer*	*4*
Outline	*Long Integer*	*8*
Shadow	*Long Integer*	*16*
Condensed	*Long Integer*	*32*
Extended	*Long Integer*	*64*

Poznámka: Na Windows jsou možné pouze kombinace Bold, Italic a Underline a samozřejmě Plain.

Příklady

1. Tento příklad nastaví styl pro tlačítko jehož název je bPřidat. Styl písma je nastaven na tučnou kurzívu:

```
FONT STYLE (bPřidat; Bold + Italic)
```

2. Tento příklad nastaví styl Normální pro všechny objekty ve formuláři, jejichž název začíná na "vt":

```
FONT STYLE (*; "vt@"; Plain)
```

Dále si přečtěte

FONT, FONT SIZE.

ENABLE BUTTON (ZPŘÍSTUPNIT TLAČÍTKO)

Verze 6.0 (Upraveno)

ENABLE BUTTON({*;} objekt)

Parametr	Typ		Popis
*		→	Pokud definováno je objekt název objektu (řetězec) pokud vynecháno objekt je pole nebo proměnná
objekt	Objekt formuláře	→	Název objektu (pokud * definována) Pole nebo proměnná (pokud * vynechána)

Popis

Příkaz ENABLE BUTTON ukáže objekty formuláře definované parametrem objekt.

Ukázaný objekt nebo tlačítko reaguje na klepnutí myši a zkratky.

Pokud definujete parametr * je objekt název objektu. Pokud parametr * vynecháte, objekt bude značit pole nebo proměnnou. V tomto případě definujete na pole nebo proměnnou (pouze objekty pole nebo proměnné) místo řetězce. Jestli chcete vědět více informací o názvech objektů, podívejte se na část Vlastnosti objektů.

Tento příkaz může být použit na následující typy objektů:

- Tlačítko, Výchozí tlačítko, 3D tlačítko, Neviditelné tlačítko, Zvýrazněné tlačítko
- Přepínač, 3D přepínač, Obrázkový přepínač
- Zaškrtávací tlačítko, 3D zaškrtávací tlačítko
- Rozevírací nabídky, Rozevírací seznam, Text se seznamem, Nabídka/Seznam
- Teploměr, Pravítko

Poznámka: Není praktické používat tento příkaz na tlačítka s automatickou akcí, protože 4th Dimension sama změní status tohoto tlačítka, když je to třeba.

Příklady

1. Následující příklad ukáže tlačítko bPotvrdit:

```
ENABLE BUTTON (bPotvrdit)
```

2. Tento příklad ukáže všechny objekty formuláře, jejichž název obsahuje "tla":

```
ENABLE BUTTON (*; "@tla@")
```

3. podívejte se na příklad u příkazu BUTTON TEXT.

Dále si přečtěte

BUTTON TEXT, DISABLE BUTTON.

DISABLE BUTTON (ZNEPŘÍSTUPNIT TLAČÍTKO)

Verze 6.0 (Upraveno)

DISABLE BUTTON({*;} objekt)

Parametr	Typ		Popis
*		→	Pokud definováno je objekt název objektu (řetězec) pokud vynecháno objekt je pole nebo proměnná
objekt	Objekt formuláře	→	Název objektu (pokud * definována) Pole nebo proměnná (pokud * vynechána)

Popis

Příkaz DISABLE BUTTON skryje objekty formuláře definované parametrem objekt.

Skrytý objekt nebo tlačítko nereaguje na klepnutí myši a zkratky.

Pokud definujete parametr * je objekt název objektu. Pokud parametr * vynecháte, objekt bude značit pole nebo proměnnou. V tomto případě definujete na pole nebo proměnnou (pouze objekty pole nebo proměnné) místo řetězce. Jestli chcete vědět více informací o názvech objektů, podívejte se na část Vlastnosti objektů.

Tento příkaz může být použit na následující typy objektů

- Tlačítko, Výchozí tlačítko, 3D tlačítko, Neviditelné tlačítko, Zvýrazněné tlačítko
- Přepínač, 3D přepínač, Obrázkový přepínač
- Zaškrtávací tlačítko, 3D zaškrtávací tlačítko
- Rozevírací nabídky, Rozevírací seznam, Text se seznamem, Nabídka/Seznam
- Teploměr, Pravítko

Poznámka: Není praktické používat tento příkaz na tlačítka s automatickou akcí, protože 4th Dimension sama změní status tohoto tlačítka, když je to třeba

Příklady

1. Následující příklad skryje tlačítko bPotvrdit:

   DISABLE BUTTON (bPotvrdit)

2. Tento příklad skryje všechny objekty formuláře, jejichž název obsahuje "tla":

   DISABLE BUTTON (*; "@tla@")

3. podívete se na příklad u příkazu BUTTON TEXT.

Dále si přečtěte

BUTTON TEXT, ENABLE BUTTON.

BUTTON TEXT (TEXT TLAČÍTKA)

Příkazy a odkazy pro Vlastnosti objektů

Verze 6.0 (Upraveno)

BUTTON TEXT ({*; }objekt; TextTlačítka)

Parametr	Typ		Popis
*		→	Pokud definováno je objekt název objektu (řetězec) pokud vynecháno objekt je pole nebo proměnná
objekt	Objekt formuláře	→	Název objektu (pokud * definována) Pole nebo proměnná (pokud * vynechána)
TextTlačítka	Řetězec	→	Nový text tlačítka

Popis

Příkaz BUTTON TEXT mění titulek tlačítka definovaného parametrem objekt na text zadaný do textTlačítka.

Pokud definujete parametr * je objekt název objektu. Pokud parametr * vynecháte, objekt bude značit pole nebo proměnnou. V tomto případě definujete na pole nebo proměnnou (pouze objekty pole nebo proměnné) místo řetězce. Jestli chcete vědět více informací o názvech objektů, podívejte se na část Vlastnosti objektů.

Tento příkaz má vliv pouze na tlačítka, která mají zobrazený text: tlačítka, zaškrtávací tlačítka a přepínače.

Většinou budete používat tento příkaz zvlášť pro každé tlačítko. Oblast tlačítka musí být dostatečně veliká pro přijetí nového textu. Pokud není, text bude oříznut. V textu tlačítek nepoužívejte Return.

Příklad

Následující příklad je metoda objektu pro tlačítko vyhledávání, které je umístěno v zápatí formuláře zobrazeného pomocí MODIFY SELECTION. Metoda hledá v tabulce; podle výsledku se skryje nebo ukáže tlačítko bDelete a změní jeho titul:

```
QUERY ([Lidé]; [Lidé]Název = vName)
Case of
   ÷ (Records in selection ([Lidé]) = 0) ` Nikdo nenalezen
      BUTTON TEXT (bDelete;" Vymazat")
      DISABLE BUTTON (bDelete)
   ÷ (Records in selection ([Lidé]) = 1) ` jeden člověk nalezen
      BUTTON TEXT (bDelete;"Vymazat člověka")
      ENABLE BUTTON (bDelete)
   ÷ (Records in selection([Lidé]) > 1) ` Mnoho lidí nalezeno
      BUTTON TEXT (bDelete;"Vymazat lidi")
      ENABLE BUTTON (bDelete)
End case
```

Dále si přečtěte

DISABLE BUTTON, ENABLE BUTTON.

SET FORMAT (NASTAVIT FORMÁT)

Příkazy a odkazy pro Vlastnosti objektů

Verze 6.0 (Upraveno)

SET FORMAT ({*;} objekt; formátzobr)

Parametr	Typ		Popis
*		→	Pokud definováno je objekt název objektu (řetězec) pokud vynecháno objekt je pole nebo proměnná
objekt	Objekt formuláře	→	Název objektu (pokud * definována) Pole nebo proměnná (pokud * vynechána)
formátzobr	Řetězec	→	Nový formát zobrazení pro objekt

Popis

Příkaz SET FORMAT nastaví formát zobrazení pro objekt který definujete v parametrech na formát zadaný do parametru formátzobr.

Pokud definujete parametr * je objekt název objektu. Pokud parametr * vynecháte, objekt bude značit pole nebo proměnnou. V tomto případě definujete odkaz na pole nebo proměnnou (pouze objekty pole nebo proměnné) místo řetězce. Jestli chcete vědět více informací o názvech objektů, podívejte se na část Vlastnosti objektů.

SET FORMAT může být použit jak pro vstupní tak pro výstupní formulář. Může být použit na pole, dostupné/nedostupné proměnné a číselné posuvné oblasti.

Používáte formát dat podle typu dat objektu.

• Formát logického objektu budou dvě hodnoty oddělené středníkem (;)

• K formátování datumového pole nebo proměnné vložte Char(n) do parametru formátzobr, kde n je jedna z následujících předefinovaných konstant:

Konstanta	***Typ***	***Hodnota***
Short	*Long Integer*	*1*
Abbreviated	*Long Integer*	*2*
Long	*Long Integer*	*3*
MM DD YYYY	*Long Integer*	*4*
Month Date Year	*Long Integer*	*5*
Abbr Month Date	*Long Integer*	*6*
MM DD YYYY Forced	*Long Integer*	*7*

• K formátování polí nebo proměnných typu čas, vložte Char (n) do parametru formátzobr, kde je jedna z následujících konstant:

Konstanta	***Typ***	***Hodnota***
HH MM SS	*Long Integer*	*1*
HH MM	*Long Integer*	*2*
Hour Min Sec	*Long Integer*	*3*
Hour Min	*Long Integer*	*4*
HH MM AM PM	*Long Integer*	*5*

• K formátování pole nebo proměnné typu obrázek, vložte Char (n) do parametru formátzobr, kde n je jedna z následujících konstant :

Konstanta	***Typ***	***Hodnota***
Truncated Centered	*Long Integer*	*1*

Scaled to Fit	*Long Integer*	*2*
On Background	*Long Integer*	*3*
Truncated non Centered	*Long Integer*	*4*
Scaled to fit proportional	*Long Integer*	*5*
Scaled to fit prop centered	*Long Integer*	*6*

Více informací o alfanumerických a číselných formátech zobrazení najdete v *Příručce návrháře 4th Dimension.*

Poznámka: Pro použití v parametru formátzobr, můžete mít předem připravené formáty zobrazení. Vytváříte je v Předvolbách databáze. Pokud je chcete použít, musí být před názvem formátu znak svislého lomítka (|).

Příklady

1. Následující řádek kódu formátuje pole [Zaměstnanci]Datum:

```
SET FORMAT ([Zaměstnanci]Datum; Char (Month Date Year))
```

2. Následující příklad mění formát pro pole [Firmy]PSČ podle délky zadaných dat:

```
If(Length ([Firmy]PSČ) = 9)
   SET FORMAT ([Firmy]PSČ; "#####-####")
Else
   SET FORMAT ([Firmy]PSČ; "### ##")
End if
```

3. Následující příklad změní formát logické pole na Ženatý a Svobodný, místo výchozího Ano a Ne:

```
SET FORMAT ([Employee]Marital Status;"Ženatý;Svobodný")
```

Dále si přečtěte

SET FILTER.

Příkazy a odkazy pro Vlastnosti objektů

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET FILTER (NASTAVIT FILTR)

Příkazy a odkazy pro Vlastnosti objektů

Verze 6.0 (Upraveno)

SET FILTER ({*;} objekt; vstupFiltr)

Parametr	Typ		Popis
*		→	Pokud definováno je objekt název objektu (řetězec) pokud vynecháno objekt je pole nebo proměnná
objekt	Objekt formuláře	→	Název objektu (pokud * definována) Pole nebo proměnná (pokud * vynechána)
vstupFiltr	Řetězec	→	Nový vstupní filtr pro objekt

Popis

Příkaz SET FILTER nastaví vstupní filtr pro objekt definovaný v objekt na filtr definovaný v vstupFiltr.

Pokud definujete parametr * je objekt název objektu. Pokud parametr * vynecháte, objekt bude značit pole nebo proměnnou. V tomto případě definujete odkaz na pole nebo proměnnou (pouze objekty pole nebo proměnné) místo řetězce. Jestli chcete vědět více informací o názvech objektů, podívejte se na část Vlastnosti objektů.

SET FILTER může být použit na vstupní i výstupní formuláře dialogů a na pole a dostupné objekty které přijímají vstupní filtr v prostředí Návrháře.

Poznámka: Tento příkaz nemůže být použit na pole umístěná v podformuláři.

Poznámka: Vstupní filtr můžete být předdefinovaný v Předvolbách databáze. Pokud jej chcete použít, musí být před názvem filtru znak svislého lomítka (|).

Příklady

1. Následující příklad nastaví vstupní filtr pro PSČ. Pokud je adresa v US, filtr je nastaven na poštovní kód. Jinak je nastavena na přijetí jakéhokoli znaku.

```
If ([Companies]Země = "US") ` Nastavit filtr pro PSČ
   SET FILTER ([Companies]ZIP Code; "&9#####")
Else ` Nastavit filtr na písmena a číslice a zvětšit písmena
   SET FILTER ([Companies]ZIP Code; "~@")
End if
```

2. Následující příklad umožní pouze písmena "a", "b", "c" nebo "g" na dvě místa pro pole:

```
SET FILTER([Tabulka]Field ;"&"+Char(Double quote)+ "a;b;c;g"+Char(Double quote)+"##")
```

Poznámka: Tento příklad nastaví vstupní filtr na &"a;b;c;g"##.

Dále si přečtěte

SET FORMAT.

SET CHOICE LIST (NASTAVIT VÝBĚROVÝ SEZNAM)

Verze 6.0 (Upraveno)

SET CHOICE LIST ({*;} objekt; seznam)

Parametr	Typ		Popis
*		→	Pokud definováno je objekt název objektu (řetězec) pokud vynecháno objekt je pole nebo proměnná
objekt	Objekt formuláře	→	Název objektu (pokud * definována) Pole nebo proměnná (pokud * vynechána)
seznam	Řetězec	→	Název seznamu k použití jako Výběrový seznam (jak je definováno v prostředí Návrháře)

Popis

Příkaz SET CHOICE LIST nastaví výběrový seznam pro objekt definovaný parametrem objekt na seznam definovaný v seznamu.

Tento příkaz může být použit na vstupní formulář nebo formulář dialogu na pole a dostupné proměnné jejichž hodnota může být předána jako text. Seznam je zobrazen během vstupu dat když uživatel označí textovou oblast.

Pokud definujete parametr * je objekt název objektu. Pokud parametr * vynecháte, objekt bude značit pole nebo proměnnou. V tomto případě definujete odkaz na pole nebo proměnnou (pouze objekty pole nebo proměnné) místo řetězce. Jestli chcete vědět více informací o názvech objektů, podívejte se na část Vlastnosti objektů.

Poznámka: Tento příkaz nemůže být použit na pole v podformuláři.

Příklad

Následující příklad nastaví výběrový seznam pro pole lodní dopravy.Pokud je doprava přes noc, jsou vybrány lodě, které mohou cestovat přes noc, jinak jsou zobrazeny standardní lodě:

```
If ([Doprava]Noční)
   SET CHOICE LIST([Doprava]Lodě; "Rychlé lodě")
Else
   SET CHOICE LIST([Doprava] Lodě; "Normální lodě")
End if
```

Příkazy a odkazy pro Vlastnosti objektů

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET ENTERABLE (NASTAVIT DOSTUPNOST)

Příkazy a odkazy pro Vlastnosti objektů

Verze 6.0 (Upraveno)

SET ENTERABLE ({*;} vstupníOblast; dostup)

Parametr	Typ		Popis
*		→	Pokud definováno je objekt název objektu (řetězec) pokud vynecháno objekt je pole nebo proměnná
vstupníOblast	Objekt formuláře	→	Název objektu (pokud * definována) Pole nebo proměnná (pokud * vynechána)
dostup	Logické	→	True pro dostupné, Flase pro nedostupné

Popis

Příkaz SET ENTERABLE nastaví objekt definovaný pomocí objekt na dostupné nebo nedostupné.

Pokud definujete parametr * je objekt název objektu. Pokud parametr * vynecháte, objekt bude značit pole nebo proměnnou. V tomto případě definujete odkaz na pole nebo proměnnou (pouze objekty pole nebo proměnné) místo řetězce. Jestli chcete vědět více informací o názvech objektů, podívejte se na část Vlastnosti objektů.

Použití tohoto příkazu je stejné jako zaškrtnutí políčka Dostupné v okně Vlastnosti objektu v Editoru formulářů. Tento příkaz pracuje na podformulář pouze pokud je umístěn v metodě formuláře pro tento podformulář.

Pokud je vstupníOblast dostupná (TRUE) uživatel může přesunout kurzor do tohoto objektu a zadat data. Pokud je vstupníOblast nedostupná (FALSE) uživatel nemůže přesunout kurzor do objektu a nemůže zadávat data.. Pokud je objekt nedostupný, tak stále můžete měnit jeho hodnotu programově.

Příklad

Následující příklad nastaví podle dopravce a podle váhy dopravovaného zboží. Pokud je méně jak 1 kg, je dopravce nastaven na Pošta a pole je nedostupné. Jinak je pole dostupné:

```
If ([Doprava]Váha<=1)
   [Doprava]Dopravce:="Pošta"
   SET ENTERABLE([Doprava]Dopravce;False)
Else
   SET ENTERABLE([Doprava]Dopravce;True)
End if
```

Dále si přečtěte

DISABLE BUTTON, ENABLE BUTTON, SET VISIBLE.

Příkazy a odkazy pro Vlastnosti objektů

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET VISIBLE (NASTAVIT VIDITELNOST)

Příkazy a odkazy pro Vlastnosti objektů

Verze 6.0

SET VISIBLE ({*;} objekt; viditelné)

Parametr	Typ		Popis
*		→	Pokud definováno je objekt název objektu (řetězec) pokud vynecháno objekt je pole nebo proměnná
objekt	Objekt formuláře	→	Název objektu (pokud * definována) Pole nebo proměnná (pokud * vynechána)
viditelné	Logické	→	True pro zviditelnění, False pro uschován

Popis

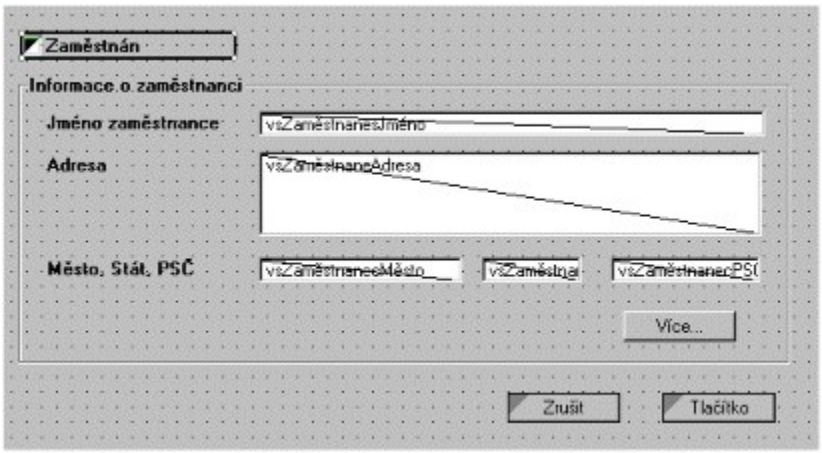
Příkaz SET VISIBLE ukáže nebo zneviditelní objekt definovaný v objekt.

Pokud definujete parametr * je objekt název objektu. Pokud parametr * vynecháte, objekt bude značit pole nebo proměnnou. V tomto případě definujete odkaz na pole nebo proměnnou (pouze objekty pole nebo proměnné) místo řetězce. Jestli chcete vědět více informací o názvech objektů, podívejte se na část Vlastnosti objektů.

Pokud předáte do viditelný TRUE, objekt je ukázán. Pokud předáte viditelný FALSE, objekt je skrytý.

Příklad

Zde je normální formulář v prostředí Návrháře:



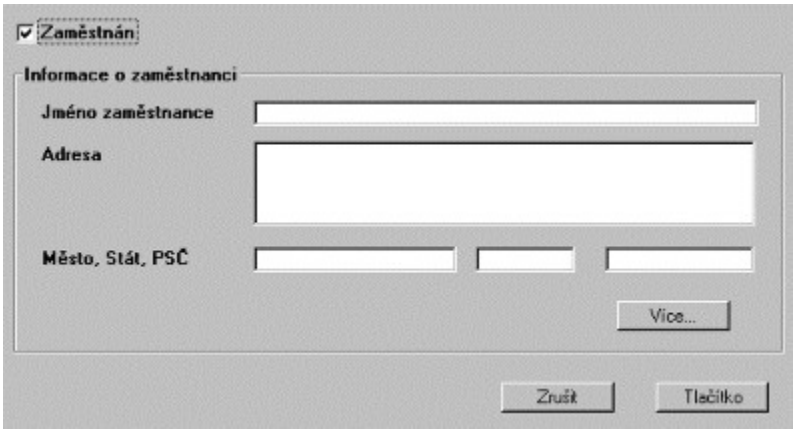
Názvy všech objektů ve skupině **Informace o zaměstnanci** obsahují text zaměsnanec. Pokud je tlačítko **Zaměstnán** zaškrtnuté, objekty musí být viditelné. Pokud toto tlačítko není viditelné, objekty budou neviditelné.

Zde je metoda objektu pro zaškrtávací tlačítko:

```
   ` Metoda objektu cbZaměstnán
Case of
   ÷ (Form event=On Load)
      cbZaměstnán:=1
   ÷ (Form event=On Clicked)
         ` Skrýt nebo ukázat objekty jejichž název obsahuje "zam"
      SET VISIBLE(*;"@zam@"; cbZaměstnán # 0)
         ` Ale vždy umožnit tlačítko cbZaměstnán viditelné
```

```
        SET VISIBLE(cbZaměstnán;True)
    End case
```

Proto v prostředí Uživatele a Vlastních nabídek bude formulář vypadat následovně:



nebo:

Dále si přečtěte

DISABLE BUTTON, ENABLE BUTTON, SET ENTERABLE.

Příkazy a odkazy pro Vlastnosti objektů

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET COLOR (NASTAVIT BARVU)

Příkazy a odkazy pro Vlastnosti objektů

Verze 6.0 (Upraveno)

SET COLOR VISIBLE ({*;}objekt; barva)

Parametr	Typ		Popis
*		→	Pokud definováno je objekt název objektu (řetězec) pokud vynecháno objekt je pole nebo proměnná
objekt	Pole n. proměnná	→	Název objektu (pokud * definována) Pole nebo proměnná (pokud * vynechána)
barva	Číslo	→	Nová barva objektu

Popis

Příkaz SET COLOR nastaví barvy popředí a pozadí pro objekt definovaný v objekt.

Pokud definujete parametr * je objekt název objektu. Pokud parametr * vynecháte, objekt bude značit pole nebo proměnnou. V tomto případě definujete odkaz na pole nebo proměnnou (pouze objekty pole nebo proměnné) místo řetězce. Jestli chcete vědět více informací o názvech objektů, podívejte se na část Vlastnosti objektů.

Parametr barva nastaví barvu popředí i pozadí. Barvy jsou počítány jako:

Barva:= - (Popředí + (256*Pozadí))

kde popředí a pozadí jsou čísla barev (od 0 do 255) v paletě barev.

Barva je vždy záporné číslo. Například když je barva popředí 20 a barva pozadí je 10, pak barva bude -(20+(256*10)) nebo -2580.

Poznámka: Paletu barev najdete v okně Vlastnosti objektu.

Čísla nejpoužívanějších barev můžete zadat jako předdefinované konstanty:

Konstanta	***Typ***	***Hodnota***
White	*Long Integer*	*0*
Yellow	*Long Integer*	*1*
Orange	*Long Integer*	*2*
Red	*Long Integer*	*3*
Purple	*Long Integer*	*4*
Dark Blue	*Long Integer*	*5*
Blue	*Long Integer*	*6*
Light Blue	*Long Integer*	*7*
Green	*Long Integer*	*8*
Dark Green	*Long Integer*	*9*
Dark Brown	*Long Integer*	*10*
Dark Grey	*Long Integer*	*11*
Light Grey	*Long Integer*	*12*
Brown	*Long Integer*	*13*
Grey	*Long Integer*	*14*
Black	*Long Integer*	*15*

Poznámka: Zatímco SET COLOR pracuje s indexovanými barvami ve výchozí paletě 4D, verze 6 obsahuje příkaz SET RGB COLORS, který vám umožní pracovat s jakoukoli RGB barvou.

Příklad

Následující příklad nastaví barvu pro tlačítko s názvem bInfo. Barva je nastavena do dvou proměnných s názvy vPopředí a vPozadí:

```
SET COLOR (vInfo; -(vPopředí + (256 * vPozadí)))
```

Dále si přečtěte

SET RGB COLORS.

Příkazy a odkazy pro Vlastnosti objektů

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET RGB COLORS (NASTAVIT RGB BARVU)

Příkazy a odkazy pro Vlastnosti objektů

Verze 6.0

SET RGB COLORS ({*;} objekt; popředíBarva; pozadíBarva)

Parametr	Typ		Popis
*		→	Pokud definováno je objekt název objektu (řetězec) pokud vynecháno objekt je pole nebo proměnná
objekt	Pole n. proměnná	→	Název objektu (pokud * definována) Pole nebo proměnná (pokud * vynechána)
popředíBarva	Číslo	→	Hodnota RGB barvy pro Popředí
pozadíBarva	Číslo	→	Hodnota RGB barvy pro Pozadí

Popis

Příkaz SET RGB COLORS změní barvy popředí a pozadí objektu na barvy definované v popředíBarva a pozadíBarva.

Pokud definujete parametr * je objekt název objektu. Pokud parametr * vynecháte, objekt bude značit pole nebo proměnnou. V tomto případě definujete odkaz na pole nebo proměnnou (pouze objekty pole nebo proměnné) místo řetězce. Jestli chcete vědět více informací o názvech objektů, podívejte se na část Vlastnosti objektů.

Barvu RGB definujete v popředí a pozadí. Hodnota RGB je 4 bytový Long Integer jehož formát (0x00RRGGBB) je popsán v následující tabulce (byty jsou číslovány od 0 do 3 zprava do leva):

Byt	***Popis***
3	*Musí být nula pokud absolutní RGB barva*
2	*Červená část barvy (0..255)*
1	*Zelená část barvy (0..255)*
-1	*Modrá část barvy (0..255)*

Následující tabulka ukazuje některé příklady barev RGB:

Hodnota	***Popis***
0x00000000	*Černá*
0x00FF0000	*Jasná červená*
0x0000FF00	*Jasná zelená*
0x000000FF	*Jasná modrá*
0x007F7F7F	*Šedá*
0x00FFFF00	*Jasná žlutá*
0x00FF7F7F	*Pastelová červená*
0x00FFFFFF	*Bílá*

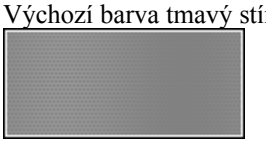
Jako alternativu můžete nastavit jednu ze čtyř automatických barev použitých 4th Dimension při kreslení objektů s automatickým nastavením barev. 4th Dimension obsahuje následující předdefinované konstanty:

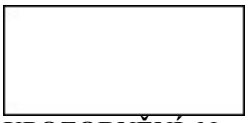
Konstanta	***Typ***	***Hodnota***
Default foreground color	*Long Integer*	*-1*
Default background color	*Long Integer*	*-2*
Default dark shadow color	*Long Integer*	*-3*
Default light shadow color	*Long Integer*	*-4*

Tyto barvy (na standardním systému) jsou ukázány zde:

Výchozí barva popředí Výchozí barva pozadí

Výchozí barva tmavý stín Výchozí barva světlý stín

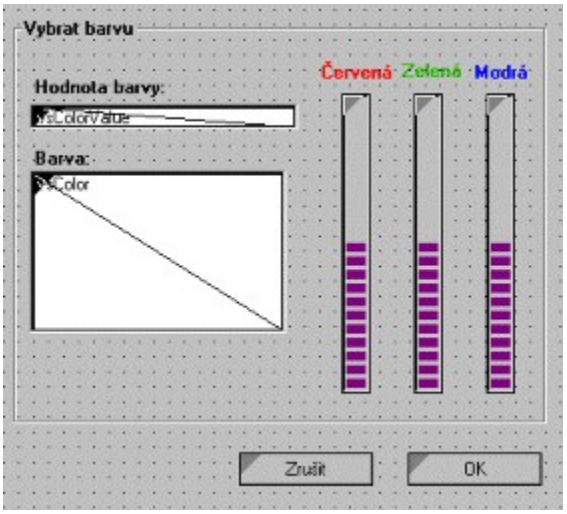




UPOZORNĚNÍ: Na vindows jsou tyto automatické barvy závislé na systému. Pokud změníte výše systémové barvy panelu barev, 4th Dimension se přizpůsobí podle tohoto nastavení. Použijte toto nastavení pro nastavení systémových barev, ne pro nastavení barev ukázaných zde.

Příklad

Tento formulář obsahuje dvě nedostupné proměnné vsColorValue a vsColor a tři teploměry: thRed, thGreen a thBlue:



Zde jsou metody pro tyto objekty:

```
   ` Metoda objektu vsColorValue nedostupné
Case of
   ÷ (Form event=On Load)
      vsColorValue:="0x00000000"
End case
   ` Metoda objektu vsColor nedostupná proměnná
Case of
   ÷ (Form event=On Load)
      vsColor:=""
      SET RGB COLORS(vsColor;0x00FFFFFF;0x0000)
End case

   ` Metoda objektu thRed teploměr
```

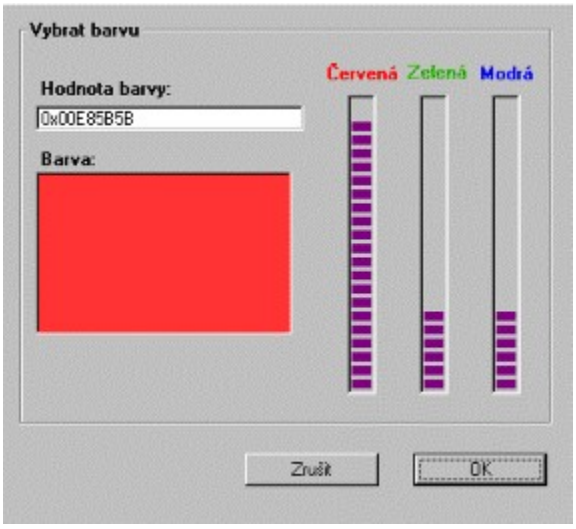
CLICK IN COLOR THERMOMETER
` Metoda objektu thGreen teploměr
CLICK IN COLOR THERMOMETER
` Metoda objektu thBlue teploměr
CLICK IN COLOR THERMOMETER

Metoda volaná teploměry je:

```
` Metoda projektu CLICK IN COLOR THERMOMETER
SET RGB COLORS(vsColor;0x00FFFFFF;(thRed << 16) +(thGreen << 8)+thBlue)
vsColorValue:=String((thRed << 16)+(thGreen << 8)+thBlue;"&x")
If (thRed=0)
   vsColorValue:=Substring(vsColorValue;1;2)+"0000"+ Substring(vsColorValue;3)
End if
```

Všimněte si použití Bitwise operátorů pro výpočty barev z teploměrů.

V prostředí Uživatele nebo Vlastních nabídek, bude formulář vypadat následovně:



Dále si přečtěte

Bitwise operátory, SET COLOR.

Příkazy a odkazy pro Vlastnosti objektů

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GET OBJECT RECT (ZÍSKAT OBDÉLNÍK OBJEKTU)

Verze 6.5

GET OBJECT RECT ({*;}objekt; levá; vrch; pravá; spodní)

Parametr	Typ		Popis
*		→	Pokud definováno je objekt název objektu (řetězec) pokud vynecháno objekt je pole nebo proměnná
objekt	Pole n. proměnná	→	Název objektu (pokud * definována) Pole nebo proměnná (pokud * vynechána)
levá	LongInt	←	Levá souřadnice objektu v okně aplikace
vrch	LongInt	←	Horní souřadnice objektu v okně aplikace
pravá	LongInt	←	Pravá souřadnice objektu v okně aplikace
spodní	LongInt	←	Spodní souřadnice objektu v okně aplikace

Popis

Příkaz GET OBJECT RECT vrací levé,horní, pravé a spodní souřadnice (v bodech) objektu pole nebo proměnné definované parametry objekt a *.

Pokud definujete parametr * je objekt název objektu. Pokud parametr * vynecháte, objekt bude značit pole nebo proměnnou. V tomto případě definujete odkaz na pole nebo proměnnou (pouze objekty pole nebo proměnné) místo řetězce. Jestli chcete vědět více informací o názvech objektů, podívejte se na část Vlastnosti objektů.

Pokud předáte název objektu do parametru objekt a použijete znak výběru náhranou (@) k označení více než jednoho objektu, budou vráceny souřednice obdélníku tvořeného všemi objekty.

Poznámka: Od 4th Dimension verze 6.5 je možné použít v řetězci znaků znak výběru náhradou ("@"). Tato možnost má účinek na příkazy Vlastností objektů. Podrobnější popis najdete v *Přiručce návrháře 4th Dimension.*

Pokud objekt neexistuje nebo není příkaz prováděn ve formuláři, jsou vráceny souřednice (0;0;0;0).

Příklad

Řekněme že chcete získat souřadnice všech objektů začínajících na "Tlačítko":

```
GET OBJECT RECT (*;"Tlačítko@";levý;vrchní;pravý;spodní)
```

Dále si přečtěte

MOVE OBJECT.

Příkazy a odkazy pro Vlastnosti objektů

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

MOVE OBJECT (POSUNOUT OBJEKT)

Příkazy a odkazy pro Vlastnosti objektů

Verze 6.5

MOVE OBJECT ({*;}objekt; posunH; posunV{; změnaH{; změnaV{; *}}})

Parametr	Typ		Popis
*	*	→	Je-li uveden, objekt je odkazován názvem Je-li vynechán, je odkazováno názvem proměnné
objekt	Řetězec	→	Objekt formuláře, název objektu (je-li uvedena *) nebo název proměnné (není-li uvedena *)
posunH	Číslo	→	Hodnota horizontálního posunu objektu (>0 = vpravo, <0 = vlevo)
posunV	Číslo	→	Hodnota vertikálního posunu (>0 = dolů, <0 = nahoru)
změnaH	Číslo	→	počet bodů změny hotizontální velikosti
změnaV	Číslo	→	Počet bodů změny vertikální velikosti
*	*	→	je-li uveden, absolutní koordinály (v okně aplikace) není-li uveden, relativní koordinály (změna o)

Popis

Příkaz MOVE OBJECT vám umožní posunout objekt(y) v platném formuláři, definované pomocí parametrů * a objekt o počet bodů definovaných posunH vodorovně a posunV svisle.

Je také možné (volitelné) změnit velikost objektu o počet bodů definovaných v změnaH vodorovně a změnaV svisle.

Směr posunu závisí na hodnotách předáných do parametů posunH a posunV:

- Pokud je hodnota kladná, je objekt posunut doprava a dolu.
- Pokud je hodnota záporná, je objekt posunut doleva a nahotu.

Pokud předáte první volitelný parametr *, označíte tím že do objektu se zadá název objektu (řetězec znaků). Pokud tento parametr nepředáte, je objekt pole nebo proměnná. V tomto případě nepředáte řetězec, ale odkaz na pole nebo proměnnou (pouze u objektů typu pole nebo proměnná).

Pokud do objektu předáte název objektu a za něj znak výběru náhradou (@) pro označení více než jednoho objektu, budou posunuty nebo změněny všechny objekty vyhovující podmínce.

Poznámka: Od 4th Dimension verze 6.5 je možné použít v řetězci znaků znak výběru náhradou ("@"). Tato možnost má účinek na příkazy Vlastností objektů. Podrobnější popis najdete v *Přiručce návrháře 4th Dimension.*

Jako výchozí mění parametry posunH, posunV, změnaH a změnaV souřadnice v závislosti na jeho dřívější poloze. Pokud chcete tyto parametry definovat jako absolutní parametry, vložte poslední parametr *.

Příklady

1. Následující řádek posune objekt "button_1" 10 bodů vpravo, 2O bodů nahoru, změní jeho šířku o 30 a výšku o 40:

```
MOVE OBJECT (*;"button_1";10;-20;30;40)
```

2. Následující příklad posune objekt "button_1" na následující souřadnice (10;20) (30;40):

```
MOVE OBJECT (*;"button_1";10;20;30;40;*)
```

Dále si přečtěte

GET OBJECT RECT.

Příkazy a odkazy pro Vlastnosti objektů

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SEARCH BY INDEX (HLEDAT DLE INDEXU)

Verze 3

SEARCH BY INDEX

Tento příkaz je ve 4th Dimension stále obsažen pro kampatibilitu s verzí 1. Pro novější programování použijte příkaz QUERY.

UPOZORNĚNÍ: Tento příkaz bude odstraněn z dalších verzí. Prosíme aby jste jej nepoužívali.

Příkazy a odkazy pro Staré příkazy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SORT BY INDEX (TŘÍDIT DLE INDEXU)

Verze 3

SORT BY INDEX

Tento příkaz je ve 4th Dimension stále obsažen pro kampatibilitu s verzí 1. Pro novější programování použijte příkaz ORDER BY.

UPOZORNĚNÍ: Tento příkaz bude odstraněn z dalších verzí. Prosíme aby jste jej nepoužívali.

Příkazy a odkazy pro Staré příkazy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ON SERIAL PORT CALL (Při volání sériového portu)

Příkazy a odkazy pro Staré příkazy

Verze 3

ON SERIAL PORT CALL (serialMetoda{; proces})

UPOZORNĚNÍ: Tento příkaz byl již odstraněn z 4th Diomension v6. Jestli jej používáte ve vašich aplikacích, je potřeba jej nahradit novými příkazy dodanými do 4th Dimesion

Příkazy a odkazy pro Staré příkazy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant.

Matematické řady

Příkazy a odkazy pro Matematické řady

Verze 3

Předmětem tohoto tématu je provádění výpočtů na řadách hodnot.

Funkce Average, Max, Min, Sum, Sum squares, Std deviation a Variance mohou být použity na pole nebo podpole. V případě pole jsou použity na výběr záznamů a v případě podpole jsou použity na výběr podzáznamů pro platný záznam. Pamatujte si, že Sum squares, Std deviation a Variance mohou být použity na pole pouze během tisku.

Tyto funkce pracují pouze na číselná data. Každá z nich vrací číselnou hodnotu.

Použití na pole

Při použití funkcí Average, Max, Min nebo Sum jsou použity na pole mimo operaci tisku, budou si načítat každý záznam pro výpočet hodnoty. Pokud je použijete na velký výběr záznamů, může tato akce zabrat mnoho času. Aby jste to omezili, indexujte pole.

Pokud jsou tyto funkce použity při zprávě, tak mají jiný styl chování. Je to proto, že zpráva musí načíst každý záznam. Používejte tyto funkce ve formuláři nebo v metodě objektu při tisku pomocí příkazu PRINT SELECTION nebo při tisku vybráním položky Tisk z nabídky Soubor v prostředí uživatele.

Při použití ve zprávě jsou tyto příkazy použitelné pouze ve zlomu úrovně 0. Znamená to, že jsou použitelné až na konci zprávy po prohlédnutí celého výběru.

Můžete je použít pouze pro metody objektu v nedostupných objektech umístěných v oblasti zlomu B0.

Dále si přečtěte

Average, Max, Min, Std deviation, Sum, Sum squares, Variance.

Příkazy a odkazy pro Matematické řady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Sum (Součet)

Příkazy a odkazy pro Matematické řady

Verze 3

Sum (čís.řada) → Číslo

Parametr	Typ		Popis
čís.Řada	pole nebo podpole	→	Data pro která chceme součet
Výsledek funkce	Číslo	←	Součet řady

Popis

Příkaz Sum vrací součet hodnot pro řadu. Pokud je řada indexovaná, jsou použity indexy pro vytvoření součtu.

```
vSoučet := Sum ([Zaměstnanci]Plat)
```

Následující metoda je volána pro tisk výběru záznamů a pro aktivování zlomů:

```
ALL RECORDS ([Zaměstnanci])
ORDER BY ([Zaměstnanci];[ Zaměstnanci]LastNm;>)
BREAK LEVEL (1)
ACCUMULATE ([Zaměstnanci]Plat)
OUTPUT FORM ([Zaměstnanci];"TiskForm")
PRINT SELECTION ([Zaměstnanci])
```

Poznámka: Parametr u příkazu BREAK LEVEL bude stejný jako počet zlomů které chcete provést. Pro více informací si přečtěte příkazy tisku.

Dále si přečtěte

ACCUMULATE, Average, BREAK LEVEL, Max, Min, ORDER BY, PRINT SELECTION, Subtotal.

Příkazy a odkazy pro Matematické řady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Average (Průměr)

Příkazy a odkazy pro Matematické řady

Verze 3

Average (čís.řada) → Číslo

Parametr	Typ		Popis
čís.řada	Pole nebo podpole	→	Data ze kterých vytvořit průměr
Výsledek funkce	Číslo	←	Průměr vypočítaný z řady

Popis

Příkaz Average vrací průměr ze zadané řady. Pokud je řada indexované pole, jsou použity indexy k výpočtu průměru.

Příklad

Následující příklad nastaví proměnnou vAverage, která je umístěna v oblasti zlomu B0 ve výstupním formuláři. Tento řádek kódu je metoda objektu pro vAverage. Metoda objektu není spuštěna pokud není prováděn zlom úrovně 0:

```
vAverage := Average ([Zaměstnanci]Plat)
```

Následující metoda je volána pro tisknutí výběru záznamů a pro aktivování zlomů:

```
ALL RECORDS ([Zaměstnanci])
ORDER BY ([Zaměstnanci];[ Zaměstnanci]Příjmení;>)
BREAK LEVEL (1)
ACCUMULATE ([Zaměstnanci]Plat)
OUTPUT FORM ([Zaměstnanci];"TiskForm")
PRINT SELECTION ([Zaměstnanci])
```

Poznámka: Parametr u příkazu BREAK LEVEL bude stejný jako počet zlomů které chcete provést. Pro více informací si přečtěte příkazy tisku.

Dále si přečtěte

ACCUMULATE, BREAK LEVEL, Max, Min, ORDER BY, PRINT SELECTION, Subtotal, Sum.

Příkazy a odkazy pro Matematické řady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Min (Minimum)

Příkazy a odkazy pro Matematické řady

Verze 3

Min (čís.řada) → Číslo

Parametr	Typ		Popis
čís.Řada	pole nebo podpole	→	Data pro která chceme minimum
Výsledek funkce	Číslo	←	Minimální hodnota v řadě

Popis

Příkaz Min vrací nejmenší hodnotu v číselné řadě. Pokud je řada indexované pole,je tato hodnota brána z indexu.

Příklad

1. Následující příklad je metoda objektu vMin umístěného v úrovni 0 ve formuláři. Proměnná je tištěna na konci zprávy. Metoda objektu přiřadí k proměnné minimální hodnotu z přiřazeného pole a vytiskne se na konci zprávy.

```
vMin:=Min ([Zaměstnanci]Plat)
```

Následující metoda je volána pro tisk výběru záznamů a pro aktivování zlomů:

```
ALL RECORDS ([Zaměstnanci])
ORDER BY ([Zaměstnanci];[ Zaměstnanci]LastNm;>)
BREAK LEVEL (1)
ACCUMULATE ([Zaměstnanci]Plat)
OUTPUT FORM ([Zaměstnanci];"PrintForm")
PRINT SELECTION ([Zaměstnanci])
```

Poznámka: Parametr u příkazu BREAK LEVEL bude stejný jako počet zlomů které chcete provést. Pro více informací si přečtěte příkazy tisku.

2. Následující příklad najde minimální prodej zaměstnance podle pole [Zaměstnanci]SalesDollars a zobrazí jej v okně upozornění:

```
ALERT ("Minimání prodej = " + String(Min([Zaměstnanci]SalesDollars)))
```

Dále si přečtěte

Execute on server, Execute on server, GET PROCESS VARIABLE, Max, Procesy, SET PROCESS VARIABLE.

Příkazy a odkazy pro Matematické řady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Max (Maximum)

Příkazy a odkazy pro Matematické řady

Verze 3

Max (čís.řada) → Číslo

Parametr	Typ		Popis
čís.Řada	pole nebo podpole	→	Data pro která chceme maximum
Výsledek funkce	Číslo	←	Maximální hodnota v řadě

Popis

Příkaz Max vrací maximální hodnotu nalezenou v zadané řadě. Pokud je řada indexované pole, jsou použity indexy pro získání maximální hodnoty.

Příklad

Následující příklad je metoda objektu vMax umístěného v úrovni 0 ve formuláři. Proměnná je tištěna na konci zprávy. Metoda objektu přiřadí k proměnné maximální hodnotu z přiřazeného pole a vytiskne se na konci zprávy.

```
vMin:=Max ([Zaměstnanci]Plat)
```

Následující metoda je volána pro tisk výběru záznamů a pro aktivování zlomů:

```
ALL RECORDS ([Zaměstnanci])
ORDER BY ([Zaměstnanci];[ Zaměstnanci]LastNm;>)
BREAK LEVEL (1)
ACCUMULATE ([Zaměstnanci]Plat)
OUTPUT FORM ([Zaměstnanci];"PrintForm")
PRINT SELECTION ([Zaměstnanci])
```

Poznámka: Parametr u příkazu BREAK LEVEL bude stejný jako počet zlomů které chcete provést. Pro více informací si přečtěte příkazy tisku.

Dále si přečtěte

Min.

Příkazy a odkazy pro Matematické řady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Std deviation (Standardní odchylka)

Příkazy a odkazy pro Matematické řady

Verze 3

Std deviation (čís.řada) → Číslo

Parametr	Typ		Popis
čís.řada	Pole nebo podpole	→	Data ze kterých vrátit standardní odchylku
Výsledek funkce	Číslo	←	Standardní odchylka řady

Popis

Příkaz Std deviation vrací standardní odchylku v řadě. Pokud je řada indexované pole, jsou pro výpočet odchylky použity indexy. Pole s touto funkcí můžete použít pouze během tisku zprávy.

Příklad

Následující příklad je metoda objektu vDeviate. Tato metoda přiřadí standardní odchylku řady k objektu vDeviate:

```
vDeviate := Std deviation ([Tabulka1]ŘadaDat)
```

Následující metoda je volána pro tisk výběru záznamů a pro aktivování zlomů:

```
ALL RECORDS ([Zaměstnanci])
ORDER BY ([Zaměstnanci];[ Zaměstnanci]LastNm;>)
BREAK LEVEL (1)
ACCUMULATE ([Zaměstnanci]Plat)
OUTPUT FORM ([Zaměstnanci];"PrintForm")
PRINT SELECTION ([Zaměstnanci])
```

Poznámka: Parametr u příkazu BREAK LEVEL bude stejný jako počet zlomů které chcete provést. Pro více informací si přečtěte příkazy tisku.

Dále si přečtěte

Average, Sum, Sum squares, Variance.

Příkazy a odkazy pro Matematické řady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Variance

Příkazy a odkazy pro Matematické řady

Verze 3

Variance (čís.řada) → Číslo

Parametr	Typ		Popis
čís.řada	Pole nebo podpole	→	Data ze kterých vrátit varianci
Výsledek funkce	Číslo	←	Variance z matematické řady

Popis

Variance vrátí varianci z matematické řady. Pokud je řada indexované pole, jsou hodnoty brány z indexu. Tuto funkci můžete na pole použít pouze v případě že je tištěno.

Příklad

Následující příklad je metoda objektu pro proměnnou var. Metoda objektu přiřadí součet čtverců pro řadu dat do objektu var:

```
var:=Variance ([Studenti]Ročník)
```

Následující metoda je volána pro tisk výběru záznamů a pro aktivování zlomů:

```
ALL RECORDS ([Zaměstnanci])
ORDER BY ([Zaměstnanci];[ Zaměstnanci]LastNm;>)
BREAK LEVEL (1)
ACCUMULATE ([Zaměstnanci]Plat)
OUTPUT FORM ([Zaměstnanci];"PrintForm")
PRINT SELECTION ([Zaměstnanci])
```

Poznámka: Parametr u příkazu BREAK LEVEL bude stejný jako počet zlomů které chcete provést. Pro více informací si přečtěte příkazy tisku.

Dále si přečtěte

Average, Std deviation, Sum, Sum squares.

Příkazy a odkazy pro Matematické řady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Sum squares (Součet čtverců)

Příkazy a odkazy pro Matematické řady

Verze 3

Sum squares (čís.řada) → Číslo

Parametr	Typ		Popis
čís.řada	Pole nebo podpole	→	Číselá řada pro kterou navrátit součet čtverců
Výsledek funkce	Číslo	←	Součet čtverců pro číselnou řadu

Popis

Sum squares vrátí součet čtverců z matematické řady. Pokud je řada indexované pole, jsou hodnoty brány z indexu. Tuto funkci můžete na pole použít pouze v případě že je tištěno.

Příklad

Následující příklad je metoda objektu pro proměnnou vSquares. Tato metoda přiřadí součet čtverců do proměnné vSquares. Tato proměnná je tištěna v posledním zlomu zprávy:

```
vSquares:=Sum squares ([Tabulka1]ŘadaDat)
```

Následující metoda je volána pro tisk výběru záznamů a pro aktivování zlomů:

```
ALL RECORDS ([Zaměstnanci])
ORDER BY ([Zaměstnanci];[ Zaměstnanci]LastNm;>)
BREAK LEVEL (1)
ACCUMULATE ([Zaměstnanci]Plat)
OUTPUT FORM ([Zaměstnanci];"PrintForm")
PRINT SELECTION ([Zaměstnanci])
```

Poznámka: Parametr u příkazu BREAK LEVEL bude stejný jako počet zlomů které chcete provést. Pro více informací si přečtěte příkazy tisku.

Dále si přečtěte

Average, Std deviation, Sum, Variance.

Příkazy a odkazy pro Matematické řady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Operátory

Příkazy a odkazy pro Operátory

Verze 3

Operátory jsou znaky které určují nějaké akce mezi výrazy. Provádějí:

• Výpočty čísel, datumů a času

• Operace na řetězcích, Logické operace na logických výrazech a specializované operace na obrázcích.

• Kombinují jednoduché výrazy pro vytvoření dalšího výrazu.

Priorita

Pořadí ve kterém jsou výrazy vyvolánány se nazývá *priorita*. 4th Dimension má přesně danou prioritu zleva do prava, ve které jsou vyvolávány matematické výrazy. Například:

```
3 + 4 * 5
```

vrátí 35, protože výrazy jsou prováděny v pořadí 3 + 4, je 7 a to je potom vynásobeno 5. Proto je celkový výsledek 35.

Pokud chcete přerušit pořadí z leva do prava, MUSÍTE použít závorky. Například:

```
3 + (4 * 5)
```

vrátí 23, protože (4 * 5) je provedeno jako první, protože je obsaženo v závorkách. Výsledek je 20 a k němu je pak přičteno 3.

Závorky mohou být použity i uvnitř jiných závorek. Pro správné provedení funkce si musíte být jisti, že každá levá závorka má i svoji pravou závorku. Chybějící závorky mohou způsobit neplatnost operace. Pak když budte kompilovat databázi, kompilátor najde chybějící závorky jako chybu syntaxe.

Operátor přiřazení

Operátor přiřazení MUSÍTE odlišovat od všech ostatních operátorů. Na rozdíl od všech jiných operátorů které provádějí různé akce, operátor přiřazení překopíruje hodnotu výrazu na pravé straně do výrazu na levé straně. Například následující příklad přiřadí hodnotu 4 (počet znaků slova Ahoj) do proměnné MojeProm. MojeProm je zapsána jako číselná proměnná:

```
MojeProm := Length ("Ahoj")
```

Důležité: NEPLEŤTE si operátor přiřazení ":=" se znaménkem rovná se "=".

Podívejte se na další operátory obsažené v jazyku 4th Dimension, popsané v následujících částech.

Operátory řetězce

Přečtěte si část Operátory řetězce.

Číselné operátory

Přečtěte si část Číselné operátory.

Datumové operátory

Přečtěte si část Datumové operátory.

Časové operátory

Přečtěte si část Časové operátory.

Operátory porovnání

Přečtěte si část Operátory porovnání.

Logické operátory

Přečtěte si část Logické operátory.

Obrázkové operátory

Přečtěte si část Obrázkové operátory.

Bitwise operátory

Přečtěte si část Bitwise operátory.

Dále si přečtěte

Konstanty, Typy dat, Identifikátory.

Příkazy a odkazy pro Operátory

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Operátory řetězce

Příkazy a odkazy pro Operátory

Verze 3

Výrazy které používají operátory řetězce vrací řetězec. Následující tabulka ukazuje operátory řetězce:

Operace	Zápis	Vrací	Výraz	Hodnota
Součet	Řetězec + Řetězec	Řetězec	"abc" + "abc"	"abcabc"
Opakování	Řetězec + Řetězec	Řetězec	"ab" * 3	"ababab"

Dále si přečtěte

Bitwise operátory, Operátory porovnání, Datumové operátory, Logické operátory, Číselné operátory, operátory, Obrázkové operátory, Časové operátory.

Příkazy a odkazy pro Operátory

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Číselné operátory

Příkazy a odkazy pro Operátory

Verze 3

Výraz který používá číselné operátory vrací číslo. Následující tabulka ukazuje číselné operátory:

Operace	Zápis	Vrací	Výraz	Hodnota
Součet	Číslo + Číslo	Číslo	2 + 3	5
Odečet	Číslo - Číslo	Číslo	3 - 2	1
Násobení	Číslo * Číslo	Číslo	5 * 2	10
Dělení	Číslo / Číslo	Číslo	5 / 2	2,5
Celé dělení	Číslo Číslo	Číslo	5 2	2
Modulo	Číslo % Číslo	Číslo	5 % 2	1
Exponenciál	Číslo ^ Číslo	Číslo	2 ^ 3	8

Operátor Modulo

Operátor modulo % dělí první číslo druhým a vrací zbytek. Zde jsou příklady:

• 10 % 2 vrací 0, protože 10 je dělitelné 2.

• 10 % 3 vrací 1, protože zbytek je 1.

• 10,5 % 2 vrací 0, protože zbytek není celé číslo.

UPOZORNĚNÍ: Modulo operátor vrací významnou hodnotu z čísel které jsou v rozsahu Long integer (od minus 2 ^ 31 do 2 ^ 31 mínus jedna). K výpočtu modulo mimo tento rozsah, použijte příkaz Mod.

Dále si přečtěte

Bitwise operátory, Operátory porovnání, Datumové operátory, Logické operátory, Operátory řetězce, operátory, Obrázkové operátory, Časové operátory.

Příkazy a odkazy pro Operátory

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Datumové operátory

Příkazy a odkazy pro Operátory

Verze 3

Výraz který používá datumové operátory vrací datum nebo číslo, podle operace. Všechny datumové operátory vrací přesné datum, a to včetně rozdílu mezi normálním a přestupným rokem. Následující tabulka ukazuje datumové operátory:

Operace	Zápis	Vrací	Výraz	Hodnota
Rozdíl datumů	Datum - Datum	Číslo	!20.1.97! - !1.1.97!	19
Přidání dní	Datum + Číslo	Datum	!20.1.97! + 9	!29.1.97!
Odečtení dní	Datum - Číslo	Datum	!20.1.97! - 9	!11.1.97!

Dále si přečtěte

Bitwise operátory, Operátory porovnání, Číselné operátory, Logické operátory, Operátory řetězce, operátory, Obrázkové operátory, Časové operátory.

Příkazy a odkazy pro Operátory

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Časové operátory

Příkazy a odkazy pro Operátory

Verze 3

Výraz který používá časové operátory vrací čas nebo číslo, podle operace. Následující tabulka ukazuje časové operátory:

Operace	Zápis	Vrací	Výraz	Hodnota
Přidání	Čas + Čas	Čas	?02:03:04? + ?01:02:03?	?03:05:07?
Odečtení	Čas - Čas	Čas	?02:03:04? - ?01:02:03?	?01:01:01?
Přidání	Čas + Číslo	Číslo	?02:03:04? + 65	7449
Odečtení	Čas - Číslo	Číslo	?02:03:04? - 65	7319
Násobení	Čas * Číslo	Číslo	?02:03:04? * 2	14768
Dělení	Čas / Číslo	Číslo	?02:03:04? / 2	3692
Celé dělení	Čas Číslo	Číslo	?02:03:04? 2	3692
Modulo	Čas % Číslo	Číslo	?02:03:04? % 2	0

Tipy

1. K získání času z výrazu který kombinuje časový výraz s číslem, použijte příkazy Time a Time string.

Příklad:

```
    ` Následující řádek přiřadí do proměnné $vlSekundy číslo, ktreré je mezi půlnocí
    ` a času za jednu hodinu
$vlSekundy := Current time + 3600
    ` Následujcí řádek přiřadí do $vhPozději čas, který bude za hodinu
$vhPozději := Time(Time string(Current time + 3600))
```

Druhý řádek může být zapsán i jednodušším způsobem:

```
    ` Následující řádek přiřadí do $vhPozději čas, který bude za hodinu
$vhPozději := Current time + ?01:00:00?
```

Nicméně při vývoji aplikace můžete očekávat situace, kde zpoždění, vyjádřeno v sekundách a předáno do časové hodnoty je dostupné pouze jako číselná hodnota.

V tomto případě použijte následující radu:

2. Některé situace mohou vyžadovat převedení časového výrazu do číselné hodnoty.

Například otevřete dokument pomocí Open document, který vrací odkaz na dokument (DocRef), který je časový výraz. Později budete chtít vložit DocRef do 4D Extension rutiny, která vyžaduje číselnou hodnotu jako odkaz na dokument. V tomto případě použijte 0 (nula) k získání číselné hodnoty z času, ale bez změnění hodnoty.

Příklad:

```
    ` Označit a otevřít dokument
$vhDokumRef:=Open document("")
If (OK=1)
        ` Vložit časový výraz DocRef
        ` jako číselný výraz do 4D Extension rutiny
    UDĚLAT NĚCO SPECIALNÍHO  (0+$vhDokumRef)
```

End if

Dále si přečtěte

Bitwise operátory, Operátory porovnání, Číselné operátory, Logické operátory, Operátory řetězce, operátory, Obrázkové operátory, Datumové operátory.

Příkazy a odkazy pro Operátory

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Operátory porovnání

Příkazy a odkazy pro Operátory

Verze 3

Tabulky v této části ukazují operátory porovnání jak jsou použity na řetězce, čísla, datumy, čas a ukazatele. Výraz který používá operátor porovnání vrací logickou hodnotu True nebo False.

Řetězce

Operace porovnání	Zápis	Vrací	Výraz	Hodnota
Rovná se	Řetězec = Řetězec	Logické	"abc" = "abc"	True
			"abc" = "abd"	False
Nerovná se	Řetězec # Řetězec	Logické	"abc" # "abd"	True
			"abc" # "abc"	False
Větší než	Řetězec > Řetězec	Logické	"abd" > "abc"	True
			"abc" > "abc"	False
Menší než	Řetězec < Řetězec	Logické	"abc" < "abd"	True
			"abc" < "abc"	False
Větší nebo rovno	Řetězec >= Řetězec	Logicnké	"abd" >= "abc"	True
			"abc" >= "abd"	False
Menší nebo rovno	Řetězec <= Řetězec	Logické	"abc" <= "abd"	True
			"abd" <= "abc"	False

Čísla

Operace porovnání	Zápis	Vrací	Výraz	Hodnota
Rovná se	Číslo = Číslo	Logické	10 = 10	True
			10 = 11	False
Nerovná se	Číslo # Číslo	Logické	10 # 11	True
			10 # 10	False
Větší než	Číslo > Číslo	Logické	11 > 10	True
			10 > 11	False
Menší než	Číslo < Číslo	Logické	10 < 11	True
			11 < 10	False
Větší nebo rovno	Číslo >= Číslo	Logické	11 >= 10	True
			10 >= 11	False
Menší nebo rovno	Číslo <= Číslo	Logické	10 <= 11	True
			11 <= 10	False

Datumy

Operace porovnání	Zápis	Vrací	Výraz	Hodnota
Rovná se	Datum = Datum	Logické	!1.1.97! = !1.1.97!	True
			!1.1.97! = !20.1.97!	False
Nerovná se	Datum # Datum	Logické	!1.1.97! # !20.1.97!	True
			!1.1.97! # !1.1.97!	False
Větší než	Datum > Datum	Logické	!20.1.97! > !1.1.97!	True
			!1.1.97! > !20.1.97!	False
Menší než	Datum < Datum	Logické	!1.1.97! < !20.1.97!	True
			!20.1.97! < !1.1.97!	False
Větší nebo rovno	Datum >= Datum	Logické	!20.1.97! >= !1.1.97!	True

			!1.1.97! >= !20.1.97!	False
Menší nebo rovno	Datum <= Datum	Logické	!1.1.97! <= !20.1.97!	True
			!20.1.97! <= !1.1.97!	False

Čas

Operace porovnání	**Zápis**	**Vrací**	**Výraz**	**Hodnota**
Rovná se	Čas = Čas	Logické	?01:02:03? = ?01:02:03?	True
			?01:02:03? = ?01:02:04?	False
Nerovná se	Čas # Čas	Logické	?01:02:03? # ?01:02:04?	True
			?01:02:03? # ?01:02:03?	False
Větší než	Čas > Čas	Logické	?01:02:04? > ?01:02:03?	True
			?01:02:03? > ?01:02:04?	False
Menší než	Čas < Čas	Logické	?01:02:03? < ?01:02:04?	True
			?01:02:04? < ?01:02:03?	False
Větší nebo rovno	Čas >= Čas	Logické	?01:02:04? >= ?01:02:03?	True
			?01:02:03? >= ?01:02:04?	False
Menší nebo rovno	Čas <= Čas	Logické	?01:02:03? <= ?01:02:04?	True
			?01:02:04? <= ?01:02:03?	False

Porovnání ukazatelů

V tabulce se porovnávají:

` vUkazA a vUkazB jsou ukazatele ke stejnému objektu
vUkazA:= →ObjektA
vUkazB:= →ObjektA
` vUkazC je ukazatel na jinému objektu
vUkazC:= →ObjektB

Operace	**Zápis**	**Vrací**	**Výraz**	**Hodnota**
Rovná se	Ukazatel = Ukazatel	Logické	vUkazA = vUkazB	True
			vUkazA = vUkazC	False
Nerovná se	Ukazatel # Ukazatel	Logické	vUkazA # vUkazC	True
			vUkazA # vUkazB	False

Více o porovnávání řetězců

• Řetězce jsou porovnávány znak po znaku.

• Při porovnávání znaků je ignorována jejich velikost: to znamená že "A" = "a" vrátí True. Aby jste zjistili jestli je velikost dvou písmen odlišná, musíte použít jejich ASCII číslo. Například následující výraz vrátí FALSE:

Ascii ("A") = **Ascii** ("a") ` protože 65 není rovno 97

• Při porovnávání řetězců jsou diakritické znaky porovnávány podle tabulky porovnávání systému. Například následující výrazy vrací True:

"a" = "á"
"n" = "Ň"
"A" = "á"
` atd.

• Znak výběru náhradou (@) může být použit v řetězci znaků místo jakéhokoli počtu znaků. Například následující výraz vrací True:

"abcdefg" = "abc@"

Znak výběru náhradou musí být použit v druhém řetězci (řetězec na pravé straně) pro nahrazení jakéhokoli počtu znaků. Následující výraz vrací False, protože byl znak výběru náhradou použit v prvním řetězci:

"abc@" = "abcdefgh"

Výběr náhradou znamená "jeden nebo více znaků nebo nic". Následující výrazy vrací True:

"abcdefghij" = "abcdefghij@"
"abcdefghij" = "@abcdefghij"
"abcdefghij" = "abcd@efghij"
"abcdefghij" = "@abcdefghij@"
"abcdefghij" = "@abcde@fghij@"

Na druhou stránku vrátí jakýkoli výraz False, pokud v něm budou použity dva znaky výběru náhradou za sebou. Například následující výraz vrátí False

"abcdef" = "abc@@"

Tip

Pokud vyžadujete řetězec ze vstupu dat, může obsahovat znak @ - nemůžete s tímto znakem zacházet jako s ostaními. Uvažte následující příklad:

```
$vsValue:=Request("Zadejte hodnotu kterou hledáte:")
If (OK=1)
   QUERY ([Zákazníci];[Zákazníci]Jméno=$vsValue+"@")
End if
```

Hodnota je vyžádána příkazem Request. Tato hodnota je pak použita jako začáteční řetězec pro hledání. Použití dvou znaků @ bylo popsáno výše a vrací při porovnávání False, proto můžete přiřadit tento znak pouze v případě, že není na konci zadaného řetězce.

Můžete udělat například následovně:

```
$vsValue:=Request("Zadejte hodnotu kterou hledáte:")
If (OK=1)
   If (Ascii($vsValue[[Length($vsValue)]])#64)
      $vsValue:=$vsValue+"@"
   End if
   QUERY ([Zákazníci];[Zákazníci]Jméno=$vsValue+"@")
End if
```

Musíte použít příkaz Ascii, protože následující výraz (jestliže není $vsValue prázdný) vrací vždy True:

$vsValue[[Length ($vsValue)]]="@"

Když budeme pokračovat v tomto příkladu, může zadaný řetězec obsahovat více znaků @ a může vypadat třeba takto "@@D@OE@@@". Následující kód odstraní všechny znaky "@" z řetězce:

```
   ` Metoda projektu No at signs
   ` No at signs ( Řetězec ) → Řetězec
   ` No at signs ( Nějaký řetězec ) → Řetězec bez @
$0:=""
For ($vlChar;1;Length($1))
```

```
    If (Ascii($1[[$vlChar]])#64)
        $0:=$0+$1[[$vlChar]]
    End if
End for
```

Jinými slovy udělá tato krátká metoda stejnou věc jako Replace string. Nicméně je potřeba (a jsou použity ASCII kódy), protože následující Replace string výraz vždy vrátí prázdný řetězec:

```
Replace string ($vsValue;"@";"") ` Všechny znaky jsou odstraněny
```

Konečná verze příkladu bude:

```
$vsValue:=Request("Zadejte hodnotu kterou hledáte:")
If (OK=1)
    QUERY ([Zákazníci];[Zákazníci]Jméno= Not at sign    ($vsValue)+"@")
End if
```

S tímto kódem bude hledání vždy podle počátku, bez rozdílu toho, jaký řetězec zadáte do dialogového okna.

Dále si přečtěte

Bitwise operátory, Datumové operátory, Logické operátory, Číselné operátory, operátory, Obrázkové operátory, Časové operátory.

Příkazy a odkazy pro Operátory

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Logické operátory

Příkazy a odkazy pro Operátory

Verze 3

4th Dimension obsahuje dva operátory, které pracují s logickými hodnotami: spojení (A) a rozdělení (NEBO). Operátor A vrací True pokud jsou oba výroky Pravda a operátor NEBO vrací True v případě, že je alespoň jeden výraz Pravda.

4th Dimension také obsahuje funkce True, False a Not. Jestli chcete vědět více informací, přečtěte si o nich v tomto manuálu.

Nsáledující tabulka ukazuje logické operátory:

Operace	Zápis	Vrací	Výraz	Hodnota
A	Logické & Logické	Logické	("A" = "A") & (15 # 3)	True
			("A" = "B") & (15 # 3)	False
			("A" = "B") & (15 = 3)	False
NEBO	Logické | Logické	Logické	("A" = "A") | (15 # 3)	True
			("A" = "B") | (15 # 3)	True
			("A" = "B") | (15 = 3)	False

Následující tatbulka ukazuje výsledky pro operátor A:

Výraz1	Výraz2	Výraz1 & Výraz2
True	True	True
True	False	False
False	True	False
False	False	False

Následující tabulka ukazuje výsledky pro operátor NEBO:

Výraz1	Výraz2	Výraz1 | Výraz2
True	True	True
True	False	True
False	True	True
False	False	False

Tip

Pokud potřebujete vypočítat exclizivní rozdělení mezi Výraz1 a Výraz2, napište:

(Výraz1 | Výraz2) & **Not**(Výraz1 & Výraz2)

Dále si přečtěte

Bitwise operátory, Operátory porovnání, Číselné operátory, Časové operátory, Operátory řetězce, operátory, Obrázkové operátory, Datumové operátory.

Příkazy a odkazy pro Operátory

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Obrázkové operátory

Příkazy a odkazy pro Operátory

Verze 3

Výrazy které používají obrázkové operátory vrací obrázek. Následující tabulka ukazuje obrázkové operátory.

Operátor	Zápis	Akce
Vodorovný součet	Obr1 + Obr2	Vloží Obr2 na pravou stránku Obr1
Svislý součet	Obr1 / Obr2	Vloží Obr2 na spodek Obr1
Výhradní superpoložení	Obr1 & Obr2	Provede XOR na Obr1a Obr2
Celkové superpoložení	Obr1 | Obr2	Provede OR na Obr1 a Obr2
Vodorovný posun	Obr + Číslo	Posunout obrázek vodorovně o Číslo bodů
Svislý posun	Obr1 / Obr2	Posunout obrázek svisle o Číslo bodů
Změna velikosti	Obr * Číslo	Zvětší obrázek v poměru Číslo
Vodorovná změna	Obr *+ Číslo	Změní vodorovné měřítko v poměru Číslo
Svislá změna	Obr */ Číslo	Změní svislé měřítko v poměru Číslo

Dva operátory & a | vždy vrací bitmapový obrázek bez rozdílu jaký obrázek bude vstupní. Je to z důvodu, že 4th Dimension jej nejdříve zkopíruje do paměti jako bitmapu, provede operaci a pak vrátí obrázek.

Ostatní operátory vrací vektorové obrázky, pokud jsou oba vstupní obrázky vektorové. Nezapomeňte že pokud jsou obrázky zobrazeny ve formuláři jako Na pozadí, jsou zobrazeny jako bitmapa.

Příklady

V následujících příkladech jsou všechny obrázky zobrazeny ve formátu zobrazení Na pozadí.

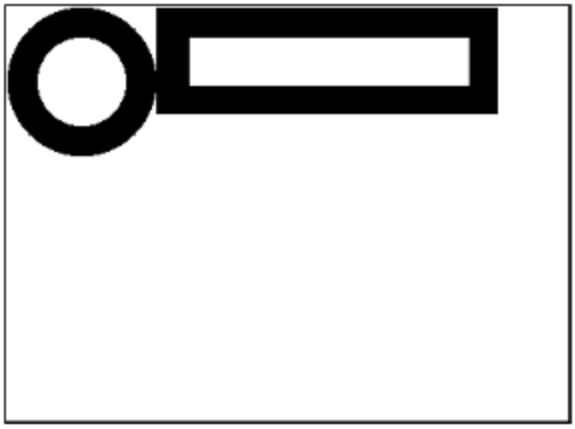
Zde je obrázek kruh:

Zde je obrázek obdélník:

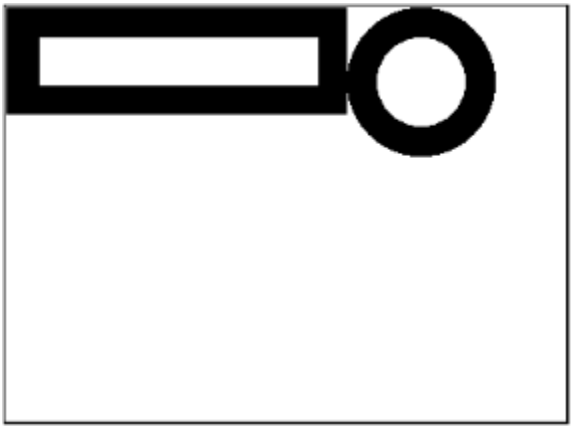
U všech následujících příkladů je za výrazem zobrazen jejich grafický výsledek.

• Vodorovný součet

kruh + obdélník ` Umístí obdélník za kruh

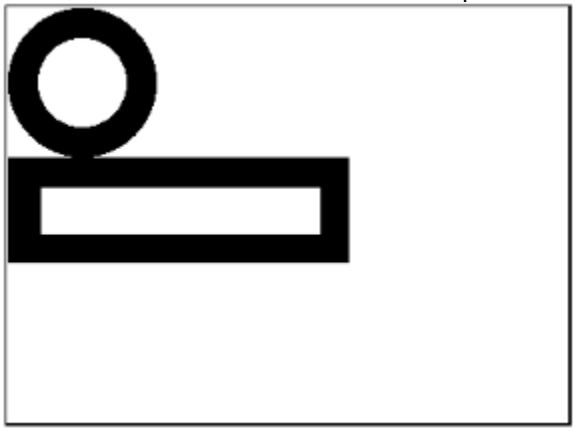


obdélník + kruh ` Umístí kruh za obdélník

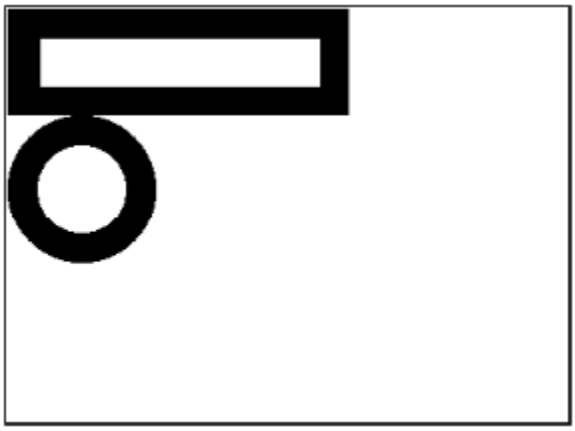


• Svislý součet

kruh / obdélník ` Umístí obdélník pod kruh

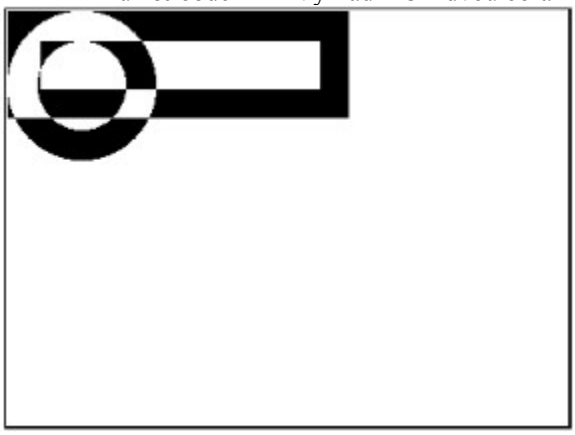


obdélník / kruh ` umístí kruh pod obdélník



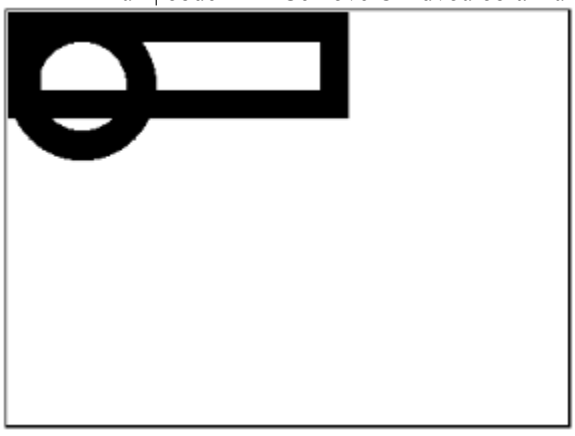
• Výhradní superpoložení (XOR)

kruh & obdélník ` Výhradní OR dvou obrázků



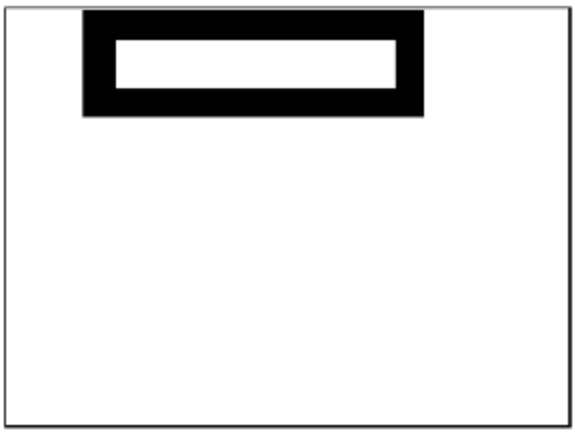
• Celkové superpoložení

kruh | obdélník ` Celkové OR dvou obrázků

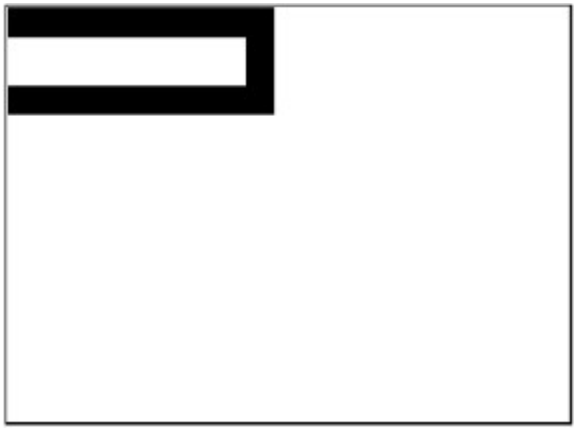


• Vodorovný posun

obdélník + 50 ` Posune obdélník o 50 bodů napravo

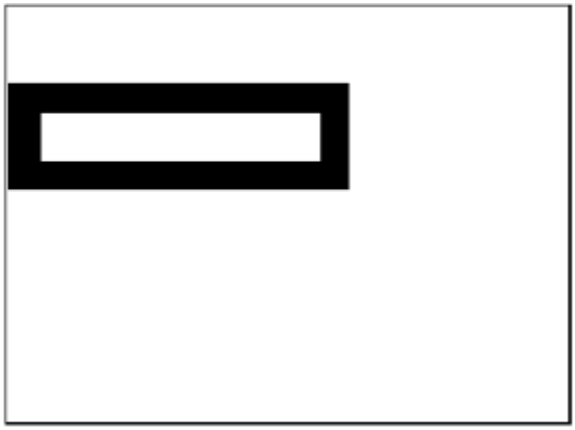


obdélník - 50 ` Posune obdélník o 50 bodů nalevo



• Svislý posun

obdélník / 50 ` Posune obdélník o 50 bodů dolů

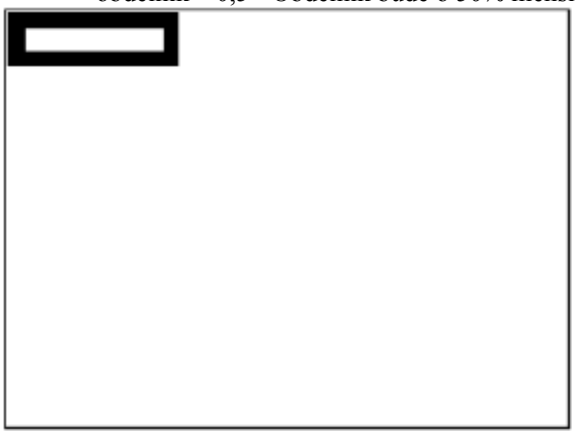


obdélník /-20 ` Posune obdélník o 20 bodů nahoru

• Změna velikosti

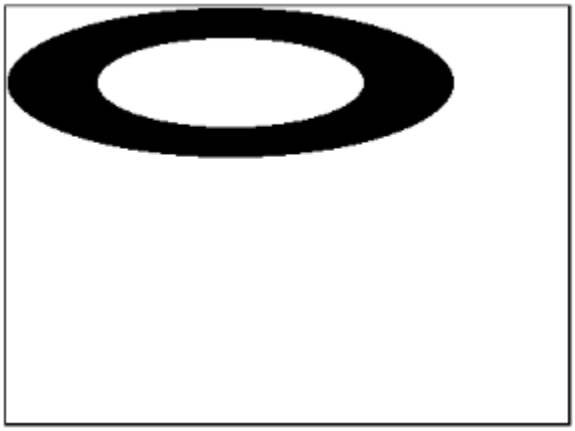
obdélník * 1,5 ` Obdélník bude o 50% větší

obdélník * 0,5 ` Obdélník bude o 50% menší



• Vodorovná změna

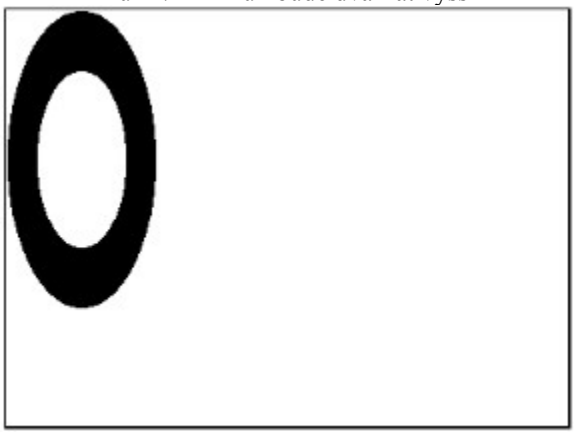
kruh *+ 3 ` Kruh bude 3x širší



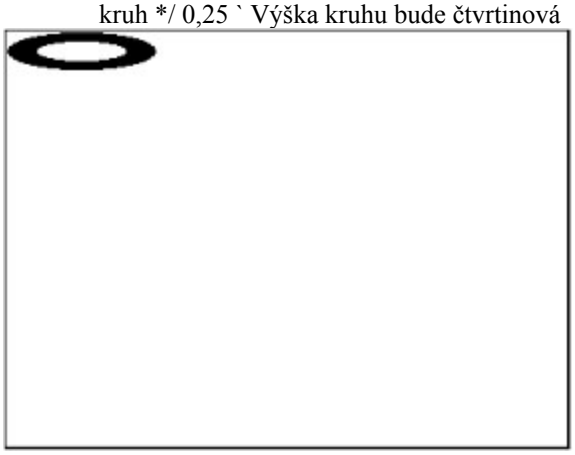
kruh *+ 0,25 ` Kruh bude mít čtvrtinovou šířku

• Svislá změna

kruh */ 2 ` Kruh bude dvakrát vyšší



kruh */ 0,25 ` Výška kruhu bude čtvrtinová



Dále si přečtěte

Bitwise operátory, Operátory porovnání, Číselné operátory, Časové operátory, Operátory řetězce, operátory, Logické operátory, Datumové operátory.

Příkazy a odkazy pro Operátory

Bitwise operátory

Příkazy a odkazy pro Operátory

Verze 3

Tyto operátory pracují s Long Integer výrazem nebo hodnotou.

Poznámka: Pokud předáte Integer nebo Real hodnotu, 4th Dimension si jí převede na Long Integer před provedením výpočtu.

Před tím, než použijete tyto operátory, musíte přemýšlet o Long Integer hodnotě jako o array 32 bitů. Bity jsou číslovány od 0 do 31 zleva do prava.

Protože každý bit může mít hodnotu 0 nebo 1, můžete uvažovat o Long Integer jako o array 32 logických hodnot. Bit obsahující 1 znamená True a 0 znamená False.

Výrazy které používají Bitwise operátory, vždy vrací hodnotu Long Integer, mimo operátoru Bit test, který vrací logickou hodnotu. Následující tabulka ukazuje seznam bitwise operátorů a jejich zápis:

Operace	Operátor	Zápis	Vrací	
A	&	Long & Long	Long	
NEBO (inclusive)	|	Long | Long	Long	
NEBO (exclusive)	^|	Long ^| Long	Long	
Levé posunutí Bitů	<<	Long << Long	Long	(Poznámka 1)
Pravé posunutí Bitů	>>	Long >> Long	Long	(Poznámka 1)
Nastavení Bitu	?+	Long ?+ Long	Long	(Poznámka 2)
Vymazání Bitu	?-	Long ?- Long	Long	(Poznámka 2)
Testování Bitu	??	Long ?? Long	Logické	(Poznámka 2)

Poznámky

1. Pro Levé a Pravé posunutí bitu určuje druhý člen pozici o kterou bude posunuta první zadaná hodnota.

Proto může být druhá hodnota mezi 0 a 32. Nezapomeňte že zatím co zadání 0 nechá číslo nezměněné, číslo větší než 31 vrátí 0x00000000, protože bity budou ztraceny. Pokud předáte jinou hodnotu do druhé části, výsledek nebude významný.

2. Pro operace Nastavení bitu, Vymazání Bitu a Testování bitu, znamená druhá část číslo bitu, na který chcete akci provést. Proto musí být druhý člen mezi 0 a 31. Jinak dostanete hodnotu nezměněnou pro Nastavení a Vymazání bitu a Testování bitu vrátí False.

Následující tabulka bitwise operátory a jejich výsledky:

Operace	Popis	
Operátor A	Každý výsledný bit je spojením hodnot které jsou předány. Zde je tabulka výsledků: 1 & 1 → 1 0 & 1 → 0 1 & 0 → 0 0 & 0 → 0 Jinými slovy výsledek je 1 pouze v případě že vstupní bity jsou oba jedna: jinak vrací 0	
Operátor NEBO (inclusive)	Každý výsledný bit je logickým NEBO obou předáných bitů.	Zde je tabulka výsledků:

	1 | 1 → 1 0 | 1 → 1 1 | 0 → 1 0 | 0 → 0 Jinými slovy je výsledný bit 1 v případě že alespoň jeden ze vstupních je jedna: jinak vrací 0.
Operátor NEBO (Exclusive)	Každý výsledný bit je logickým XNEBO dvou předaných bitů. Zde je tabulka výsledků: 1 ^| 1 → 0 0 ^| 1 → 1 1 ^| 0 → 1 0 ^| 0 → 0 Jinými slovy výsledný bit je 1 pouze v případě, že je pouze jeden ze vstupních jedna: jinak vrací 0.
Levé posunutí bitu	Výsledek je nastaven na první hodnotu a pak je posunut o počet bitů zadaných v druhé hodnotě. Bity nalevo jsou ztraceny a nové napravo jsou nastaveny na 0. Poznámka: Do druhého čísla vkládejte vždy kladnou hodnotu. Posunutí nalevo o N je to samé jako dělení 2^N.
Pravé posunutí bitu	Výsledná hodnota je nastavena na první hodnotu a pak je posunuta doprava o počet bitů zadaných v druhé hodnotě. Bity napravo jsou ztraceny a nové nalevo jsou nastaveny na 0. Poznámka: Do druhého čísla vkládejte vždy kladnou hodnotu. Posunutí napravo o N je to samé jako dělení 2^N.
Nastavení Bitu	Výsledná hodnota je nastavena na první předánou hodnotu a pak výsledný bit, jehož číslo je zadáno v druhé hodnotě je nastaven na 1. Ostatní zůstane nezměněno.
Vymazání Bitu	Výsledná hodnota je nastavena na první zadanou a pak výsledný bit, který je určen druhou hodnotou je nastaven na 0. Ostatní zůstane nezměněno.
Test Bitu	Vrací True pokud je bit v první hodnotě, určený číslem v druhé hodnotě, 1. Vrací Flase, pokud je tento bit 0.

Příklady

1. Následující tabulka ukazuje příklad ke každému Bitwise oprátoru:

Operace	***Příklad***	***Výsledek***
Operátor A	*0x0000FFFF & 0xFF00FF00*	*0x0000FF00*
NEBO (inclusive)	*0x0000FFFF | 0xFF00FF00*	*0xFF00FFFF*
NEBO (exclusive)	*0x0000FFFF & 0xFF00FF00*	*0xFF0000FF*
Levé posunutí	*0x0000FFFF << 8*	*0x00FFFF00*
Pravé posunutí	*0x0000FFFF >> 8*	*0x000000FF*
Nastavení bitu	*0x00000000 ?+ 16*	*0x00010000*
Vymazání bitu	*0x00010000 ?- 16*	*0x00000000*
Test bitu	*0x00010000 ?? 16*	*True*

2. 4th Dimension obsahuje mnoho předdefinovaných konstant. Názvy některých těchto konstant končí na "bit" nebo "mask". Například zde jsou konstanty obsažené v tématu Resources properties:

Konstanta	***Typ***	***Hodnota***
System heap resource mask	*Long Integer*	*64*
System heap resource bit	*Long Integer*	*6*
Purgeable resource mask	*Long Integer*	*32*

Purgeable resource bit	*Long Integer*	*5*
Locked resource mask	*Long Integer*	*16*
Locked resource bit	*Long Integer*	*4*
Protected resource mask	*Long Integer*	*8*
Protected resource bit	*Long Integer*	*3*
Preloaded resource mask	*Long Integer*	*4*
Preloaded resource bit	*Long Integer*	*2*
Changed resource mask	*Long Integer*	*2*
Changed resource bit	*Long Integer*	*1*

Tyto konstanty vám umožní testovat hodnotu vrácenou příkazem Get resource properties nebo je můžete použít při vytváření hodnoty předáné do SET RESOURCE PROPERTIES. Konstanty jejichž název končí na "bit" dávají pozici bitu který chcete testovat, upravit nebo vymazat. Konstanty jejich název končí na "mask" dávají Long Integer hodnotu kde bit (který chcete testovat, vymazat nebo nastavit) je roven 1.

Například k testování jestli je resource prázdný nebo není, můžete napsat následující:

If ($vlResAttr ?? Purgeable resource bit) ` Je resource prázdný?

nebo:

If (($vlResAttr & Purgeable resource mask) # 0) ` Je resource prázdný?

N druhou stránku je můžete použít k nastavení stejného bitu. Můžete napsat:

$clResAttr := $vlResAttr ?+ Purgeable resource bit

Nebo:

$vlResAttr:=$vlResAttr | Purgeable resource bit

3. Tento příklad uloží dvě Integer hodnoty do jedné hodnoty Long Integer. Můžete napsat:

$vlLong := ($viIntA << 16) | $viIntB ` Uložit dva Integer do Long Integer

$viIntA := $vlLong>>16 ` Zpět vyjmout integer uložený ve "vyšší" části

$viIntB := $vlLong & 0xFFFF ` Zpět vyjmout Integer uložený v "nižší" části

Tip: Buďte opatrní při práci s Long Integer a Integer hodnotami při kombinování číselných a bitwise operátorů. Nejvyšší bit (31 pro Long Integer a 15 pro Integer) nastavuje značku hodnoty - kladná pokud je to vymazáno, záporná jestliže je to nastaveno. Číselné operátory používají tento bit pro určení znaménka u čísla, ale bitwise operátory se o něj nestarají.

Dále si přečtěte

Obrázkové operátory, Operátory porovnání, Číselné operátory, Časové operátory, Operátory řetězce, operátory, Logické operátory, Datumové operátory.

Příkazy a odkazy pro Operátory

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Report (Sestava)

Příkazy a odkazy pro Tisk

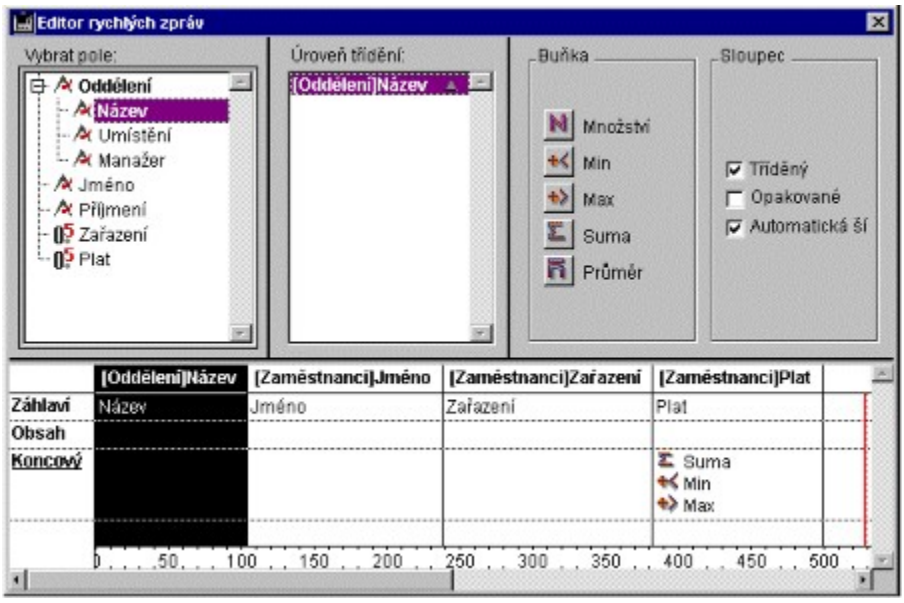
Verze 3

REPORT ({tabulka; }dokument{; *})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka k tisku nebo výchozí tabulka pokud vynecháno
dokument	Řetězec	→	Dokument rychlé zprávy
*		→	Zakázat okna tisku

Popis

Příkaz REPORT vytiskne zprávu pro tabulku vytvořenou v Editoru rychlých zpráv, který je ukázán zde:



Parametr dokument určuje název rychlé zprávy vytvořené a uložené na disku. Dokument zprávy uložíte vybráním položek Uložit nebo Uložit jako z nabídky Soubor v Editoru Rychlých zpráv. Dokument obsahuje nstavení zprávy nikoli záznamy k tištění.

Pokud do parametru dokument zadáte prázdný řetězec, je otevřeno standardní dialogové okno Otevřít soubor, kde můžete zprávu vybrat ručně. Po vybrání zprávy a potvrzení okna se zobrazí dialogy tisku, pokud není zadán parametr *. Pokud je tento parametr definován, nejsou otevřena okna pro nastavení tiskárny a zpráva je hned tištěna.

Pokud parametr dokument určuje zprávu která neexistuje, je otevřen Editor Rychlých zpráv.

Editor rychlých zpráv umožní uživateli vytvořit svojí vlastní zprávu. Jakmile je editor otevřen, jsou zobrazeny čtyři stejné nabídky které slouží k řízení zpráv v prostředí uživatele: Soubor, Upravit, Písmo a Styl. Jestli chcete vědět více informací o Editoru rychlých zpráv, přečtěte si *Přiručku uživatele 4th Dimension.*

Pokud Editor rychlých zpráv není zrušen, je proměnná OK nastavena na 1pokud je zpráva tištěna: jinak je nastavena na 0.

Příklady

1. Následující příklad umožní uživateli vyhledat záznamy v tabulce [Lidé] a pak je automaticky vytiskne ve zprávě

"Podrobný tisk":

```
QUERY ([Lidé])
If (OK = 1)
   REPORT ([Lidé]; "Podrobný tisk"; *)
End if
```

2. Následující příklad umožní uživateli hledat v tabulce [Lidé] a pak vybrat návrh zprávy k tisku:

```
QUERY ([Lidé])
If (OK = 1)
   REPORT ([Lidé]; "")
End if
```

3. Následující příklad umožní uživateli hledat v tabulce [Lidé] a pak zobrazí Editor zpráv, kde bude uživatel moci navrhnout, uložit, načíst a tisknout zprávy:

```
QUERY ([Lidé])
If (OK = 1)
   REPORT ([Lidé]; Char(1))
End if
```

Dále si přečtěte

PRINT LABEL, PRINT SELECTION.

Příkazy a odkazy pro Tisk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

PRINT LABEL (TISKNOUT ŠTÍTEK)

Příkazy a odkazy pro Tisk

Verze 3

PRINT LABEL ({tabulka; }dokument{; *})

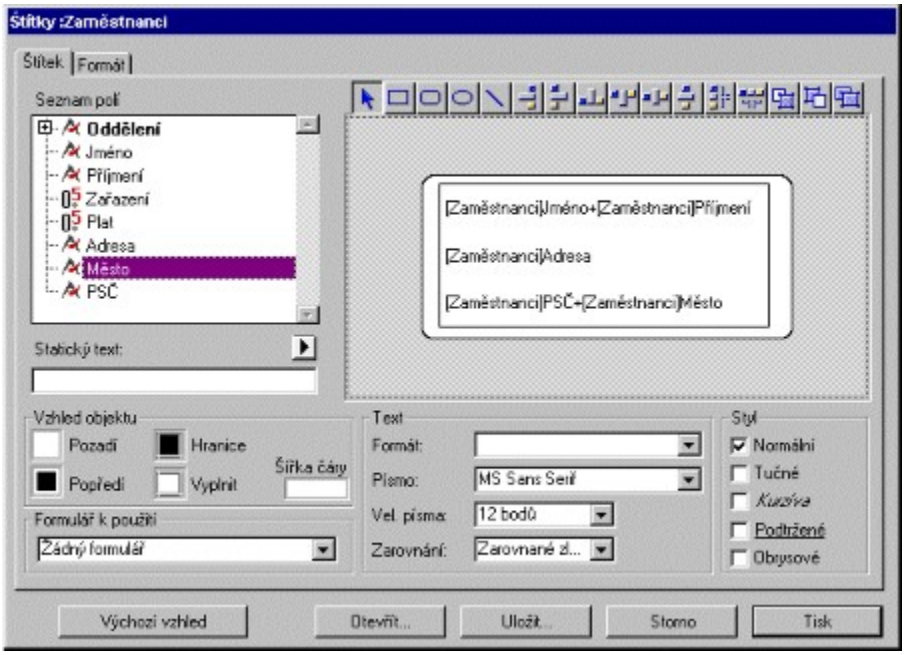
Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka k tisku nebo výchozí tabulka pokud vynechámo
dokument	Řetězec	→	Dokument štítků
*			Zakázat okna tisku

Popis

Příkaz PRINT LABEL vám umožní tisknout štítky z výběru pro tabulku.

Pokud nedefinujete parametr dokument, PRINT LABEL vytiskne aktuální výběr jako štítky pomocí platného výstupního formuláře. Tento příkaz nemůžete použít k tisku podformuláře. Jestli chcete vědět více informací o vytváření formulářů pro štítky přečtěte si *Příručku návrháře 4th Dimension.*

Pokud zadáte parametr dokument, umožní vám 4th Dimension otevřít Editor štítků nebo otevřít uložený vzor štítků. Přečtěte si následující diskuzi:



V obou případech, k vynechání okna tisku, vložte parametr *. Nezapomeňte že tento parametr nemá žádný vliv na zobrazení Editoru štítků.

Pokud Editor štítků neni zrušen, je proměnná OK nastavena na 1pokud je zpráva tištěna: jinak je nastavena na 0.

Pokud zadáte parametr dokument, jsou tištěny štítky,jejichž návrh je uložen v tomto dokumentu. Pokud je dokument prázdný řetězec (""), je otevřeno okno Otevřít soubor, kde může uživatel vybrat ručně dokument s návrhem štítků. Pokud zadaný dokument neexistuje (například vložte Char(1)), je zobrazen Editor štítků.

Příklady

1. Následující příklad vytiskne štítky pomocí výstupního formuláře pro tabulku. Příklad ukazuje dvě metody. První nastaví správný výstupní formulář a pak vytiskne štítky:

```
ALL RECORDS([Adresy]) ` Vybere všechny záznamy
OUTPUT FORM ([Adresy]; "Štítky") ` Nastaví výstupní formulář
PRINT LABEL([Adresy]) ` Vytiskne štítky
OUTPUT FORM ([Adresy];"Seznam") ` Nastaví výchozí výstupní formulář
```

Druhá metoda je metoda formuláře pro formuláře "Štítky". Formulář obsahuje jednu proměnnou vlLabel, která je použita pro spojená pole. Pokud druhé pole adresy (Adre2) je prázdné, není připojeno k této proměnné. Nezapomeňte, že tato akce je automaticky prováděna Editorem štítků. Metoda formuláře vytvoří štítky pro každý záznam:

```
   ` Metoda formuláře [Adresy]; "Štítky"
Case of
   ÷ (Form event=On load)
      vLabel:=[Adresy]Jméno+" "+[Adresy]Příjmení+Char(13)+[Adresy]Adre1+Char(13)
      If ([Adresy]Adre2 # "")
         vLabel:=vLabel +[Adresy]Adre2+Char(13)
      End if
      vLabel:=vLabel+[Adresy]Město+", "+[Adresy]Stát+" "+[Adresy]PSČ
End case
```

2. Následující příklad umožní uživateli vyhledat záznamy v tabulce [Lidé] a pak je automaticky vytiskne jako štítky z návrhu dokumentu"Štítky":

```
QUERY ([Lidé])
If (OK = 1)
   PRINT LABEL ([Lidé]; "Štítky"; *)
End if
```

3. Následující příklad umožní uživateli hledat v tabulce [Lidé] a pak vybrat návrh štítků k tisku:

```
QUERY ([Lidé])
If (OK = 1)
   PRINT LABEL ([Lidé]; "")
End if
```

3. Následující příklad umožní uživateli hledat v tabulce [Lidé] a pak zobrazí Editor štítků, kde bude uživatel moci navrhnout, uložit, načíst a tisknout štítky:

```
QUERY ([Lidé])
If (OK = 1)
   PRINT LABEL ([Lidé]; Char(1))
End if
```

Dále si přečtěte

PRINT SELECTION, REPORT.

Příkazy a odkazy pro Tisk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

PRINT SELECTION (TISKNOUT VÝBĚR)

Příkazy a odkazy pro Tisk

Verze 3

PRINT SELECTION ({tabulka}{; }{*})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka k tisku nebo výchozí tabulka pokud vynecháno
*		→	Zakázat okna tisku

Popis

Příkaz PRINT SELECTION vytiskne platný výběr záznamů pro tabulku pomocí platného výstupního formuláře. Tento příkaz provádí stejnou akci jako položka Tisk z nabídky Soubor v Prostředí uživatele. Pokud je výběr záznamů prázdný, PRINT SELECTION neprovede nic.

Jako výchozí příkaz PRINT SELECTION zobrazí dialogová okna pro tisk. Pokud nechcete tato okna zobrazit, můžete zadat parametr * a tím je zakázat zobrazit. Pokud uživatel zruší některé z oken tisku, je příkaz přerušen a výběr není vytištěn. S použitím volitelného parametru * bude výběr tištěn s nastavením které bylo při vytvoření formuláře, nebo které jste nastavili pomocí příkazu PAGE SETUP.

Během tisku jsou prováděny metody formuláře a metody objektů, které mají označené události pro tisk. Tyto události se zadávají v oknech Vlastnosti objektu a Vlastnosti formuláře. Jsou to následující události:

- V záhlaví - je generováno před tiskem záhlaví
- Při tisku obsahu - je generováno před tiskem záznamu
- Při tisku zlomu - je generováno před tiskem zlomu
- Při tisku zápatí - je generováno před tiskem záhlaví

Můžete testovat kdy PRINT SELECTION tiskne první záhlaví testováním Before selection během události V záhlaví. Můžete také testovat tisk posledního zápatí pomocí End selection během události Při tisku zápatí. Jestli chcete vědět více informací, přečtěte si popis k těmto příkazům, stejně jako Form event a Level.

Pro tisk tříděného výběru se zlomy a mezisoučty pomocí příkazu PRINT SELECTION, musíte nejdříve výběr setřídit. Pak do oblasti každého zlomu umístit proměnnou jejíž Metoda objektu k ní přiřadí mezisoučet. Můžete použít příkazy jako Sum a Average k přiřazení hodnoty do této proměnné. Jestli chcete vědět více informací, přečtěte si popis k příkazům Subtotal, BREAK LEVEL a ACCUMULATE.

Upozornění: Nepoužívejte příkaz PAGE BREAK s příkazem PRINT SELECTION. PAGE BREAK je navržen pro použití s příkazem PRINT FORM.

Po zavolání příkazu PRINT SELECTION je proměnná OK nastavena na 1, jestliže byl tisk správně dokončen. Pokud byl tisk přerušen, je proměnná OK nastavena na 0 (zero) (to znamená, že uživatel klepne na Zrušit v jednom z oken tisku).

Příklad

Následující příklad vybere všechny záznamy z tabulky [Lidé]. Pak zobrazí záznamy pomocí příkazu DISPLAY SELECTION a umožní uživateli označit záznamy k tisku. Nakonec použije vybrané záznamy pomocí příkazu USE SET a pak je vytiskne příkazem PRINT SELECTION:

```
ALL RECORDS([Lidé]) ` Vybere všechny záznamy
DISPLAY SELECTION ([Lidé]; *) ` Zobrazí záznamy
```

USE SET ("UserSet") ` Použije pouze záznamy označené uživatelem
PRINT SELECTION([Lidé]) ` Vytiskne záznamy

Dále si přečtěte

ACCUMULATE, BREAK LEVEL, level, PAGE SETUP, Subtotal.

Příkazy a odkazy pro Tisk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Printing page (Tištěná stránka)

Příkazy a odkazy pro Tisk

Verze 3

Printing page → Číslo

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Číslo	←	Číslo stránky, která se tiskne

Popis

Printing page vrátí číslo tištěné stránky. Může být použit pouze když tisknete pomocí příkazu PRINT SELECTION nebo položky Tisk z nabídky Soubor v prostředí uživatele.

Příklad

Následující příklad mění pozici čísel stránek ve zprávě tak, že může být zpráva tištěna jako dvoustránkový formát. Formulář pro zprávu má dvě proměnné pro zobrazení čísel stránek. Proměnná v levém spodním rohu je (vLeftPageNum) bude tisknout sudá čísla stránky a proměnná v pravém spodním rohu bude (vRightPageNum) bude tisknout lichá čísla. Metoda otestuje stránky a podle toho vymaže nebo doplní do proměnné:

```
Case of
   ÷ (Form event=On Printing Footer)
      If ((Printing page % 2) = 0) ` Modul je 0, pokud je stránka sudá
         vLeftPageNum:=String(Printing page) ` Nastaví číslo levé stránky
         vRightPageNum:="" ` Vymaže číslo pravé stránky
      Else ` Jinak je to lichá stránka
         vLeftPageNum:="" ` Vymaže číslo levé stránky
         vRightPageNum:=String (Printing page) ` Nastaví číslo pravé stránky
      End if
End case
```

Dále si přečtěte

PRINT SELECTION.

Příkazy a odkazy pro Tisk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

BREAK LEVEL (ÚROVEŇ ZLOMU)

Příkazy a odkazy pro Tisk

Verze 3

BREAK LEVEL (úroveň {; zlomStr})

Parametr	Typ		Popis
úroveň	Číslo	→	Počet úrovní prováděných zlomů
zlomStr	Číslo	→	Úroveň zlomu pro který provést zlom stránky

Popis

Příkaz BREAK LEVEL definuje počet zlomů, které chcete provést při tisku pomocí PRINT SELECTION.

Upozornění: Ve zkompilovaném módu **musíte** provést BREAK LEVEL a ACCUMULATE před každou zprávou pro kterou chcete zlomy. Tyto příkazy aktivují provádění zlomů pro zprávu. Podívejte se na příkaz Subtotal.

Parametr úroveň určuje nejhlubší úroveň, pro kterou chcete zlomy provádět. Záznamy musíte mít setříděné alespoň na tento počet úrovní. Pokud máte tříděno na více úrovní, budou záznamy tištěny tříděné, ale nebudou pro ně zobrazovány a počítány zlomy.

Každý zlom který je vytvořen bude tisknout patřičné oblasti zlomu a záhlaví vytvořené ve formuláři. Musíte mít vytvořeno alespoň tolik oblastí zlomu kolik jste zadali úrovní zlomu. Pokud máte více oblastí zlomu, jsou ignorovány a nejsou tištěny.

Druhý parametr, volitelný, ZlomStr je použit k provedení zlomů stránek během tisku.

Příklad

Následující příklad vytiskne zprávu s dvěmi úrovněmi zlomu. Výběr je tříděn na 4 úrovně, ale příkaz PAGE BREAK definuje zlomy pouze na dvě úrovně. Jedno pole je akumulováno příkazem ACCUMULATE:

```
ORDER BY ([Emp]Dept;>;[Emp]Title;>;[Emp]Last;>;[Emp]First;>) `Třídit 4 úrovně
BREAK LEVEL (2) ` Zapnout zlomy na 2 úrovně (Dept a Title)
ACCUMULATE ([Emp]Plat) ` Acumulovat platy
OUTPUT FORM ([Emp];"Dept Plat") ` Vybrat formulář zprávy
PRINT SELECTION([Emp]) ` Tisknout zprávu
```

Dále si přečtěte

ACCUMULATE, ORDER BY, PRINT SELECTION, Subtotal.

Příkazy a odkazy pro Tisk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ACCUMULATE (AKUMULOVAT)

Příkazy a odkazy pro Tisk

Verze 3

ACCUMULATE (data{; data2; ...;dataN})

Parametr	Typ		Popis
data	Pole n. proměnná	→	Číslené pole nebo proměnná které akumulovat hodnoty

Popis

ACCUMULATE definuje pole nebo proměnnou která se má akumulovat během tisku pomocí PRINT SELECTION.

Upozornění: Ve zkompilovaném módu **musíte** provést BREAK LEVEL a ACCUMULATE před každou zprávou pro kterou chcete zlomy. Tyto příkazy aktivují provádění zlomů pro zprávu. Podívejte se na příkaz Subtotal.

Použijte příkaz ACCUMULATE když chcete mít mezisoučty v tištěné zprávě. ACCUMULATE řekne 4th Dimension aby prováděla mezisoučty na každá Data zadaná do parametrů. Mezisoučty jsou provedeny pro každou úroveň zlomu kterou definujete příkazem BREAK LEVEL.

Příkaz ACCUMULATE proveďte před tiskem zprávy pomocí příkazu PRINT SELECTION.

Použijte příkaz Subtotal v metodě formuláře a objektu k získání mezisoučtů z parametrů Data.

Příklad

Přečtěte si příklady k příkazu BREAK LEVEL.

Dále si přečtěte

BREAK LEVEL, ORDER BY, PRINT SELECTION, Subtotal.

Příkazy a odkazy pro Tisk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Subtotal (Mezisoučet)

Příkazy a odkazy pro Tisk

Verze 3

Subtotal (data{; zlomstr}) → Číslo

Parametr	Typ		Popis
data	Pole n. proměnná	→	Číslené pole nebo proměnná, ze které provést mezisoučet
zlomstr	Číslo	→	Úroveň zlomu, pro který provést zlom stránky
Výsledek funkce	Číslo	←	Mezisoučet dat

Popis

Subtotal vrací mezisoučet pro platný nebo poslední zlom. Pracuje pouze když je výběr setříděný a tištěný příkazem PRINT SELECTION nebo položkou Tisk z nabídky Soubor v Prostředí uživatele. Parametr data musí být typu Real, Integer nebo Long Integer. Výsledek funkce Subtotal přiřaďte k proměnné umístěné v oblasti zlomu formuláře.

Upozornění: Ve zkompilovaném módu **musíte** provést BREAK LEVEL a ACCUMULATE před každou zprávou pro kterou chcete zlomy. Tyto příkazy aktivují provádění zlomů pro zprávu. Podívejte se na příkaz Subtotal.

Subtotal může být umístěn v metodě formuláře nebo objektu. 4th Dimension před tiskem prohlédne metody formuláře a objektu: jestliže najde funkci Subtotal, bude zapnuto provádění zlomů (pouze v nezkompilované verzi).

Druhý, volitelný, parametr je použit k vytvoření zlomu stránky během tisku. Pokud je nastaven na 0, Subtotal neprovede zlom stránky. Pokud je zlomstr nastaven na 1, je proveden zlom pro každou úroveň zlomu 1. Pokud je nastaven na 2, je proveden zlom stránky po každém zlomu úrovně 2 a 1.

Tip: Pokud provedete Subtotal z výstupního formuláře, který je zobrazen na obrazovce, bude generována chyba a dostanete se do nekonečné smyčky mezi zobrazením formuláře a oknem chyby. K uniknutí z této smyčky, stiskněte Alt+Shift (Windows) nebo Option+Shift (Macintosh) při klepnutí na tlačítko Odstranit v okně chyby (možná bude potřeba to udělat několikrát). Dočasně tím zastavíte obnovu okna formuláře. Vyberte jiný formulář jako výstupní, jinak se chyba objeví znovu. Pokud tento formulář používáte jak pro tisk zprávy tak pro zobrazení, vraťte se do prostředí návrháře a přesuňte funkci Subtotal pod test Form event=On Printing Break.

Příklad

Následující příklad je metoda objektu v oblasti zlomu. K proměnné je přiřazena hodnota mezisoučtu pro tento zlom:

```
Case of
   ÷ (Form event=On Printing Break)
      vPlat:=Subtotal ([Zaměstnanci]Plat)
End case
```

Jestli chcete vědět více informací o navrhování formulářů pro zprávy, přečtěte si *Příručku návrháře 4th Dimension.*

Rozšířené řízení zlomů ve formuláři zprávy

Provádění zlomů ve zprávě může být aktivováno dvěma způsoby:

- Použitím funkce Subtotal
- Použitím příkazů BREAK LEVEL a ACCUMULATE.

Oba způsoby dosáhnou stejného výsledku, ale mají rozdílné možnosti.

Požití Subtotal pro zapnutí zlomů (pouze nezkompilovaná verze)

K zapnutí zlomů pomocí funkce Subtotal, musí být tato funkce použita v metodě formuláře nebo objektu a musí přiřazovat hodnotu do proměnné, která je použita v oblasti zlomu. Před tiskem zprávy prohlédne 4th Dimension metodu formuláře a metody objektu a vyhledá funkci Subtotal.

Pokud tuto funci najde, aktivuje provádění zlomů. Tato funkce nemusí být zvlášť prováděna pro zapnutí zlomů. Například může být použita v metodě objektu umístěného v zápatí a nikde předtím ani potom nemusí být použita.

Pokud je Subtotal použita k aktivování zlomů, musíte třídit výběr o jednu úroveň více, než máte zlomů ve zprávě. Například pokud chcete dvě úrovně zlomu, musíte třídit výběr podle třech polí.

Použití BREAK LEVEL a ACCUMULATE pro zapnutí zlomů

Pro zapnutí zlomů můžete také použít příkazy BREAK LEVEL a ACCUMULATE. K tomu musíte tyto dva příkazy provést před provedením tisku. V tomto případě je stále vyžadována funkce Subtotal pro výpočet mezisoučtů pro zlomy.Nepotřebujete ale třídit výběr podle jedné úrovně navíc, samozřejmě, že je potřeba setřídit výběr podle počtu zlomů, které chcete provést.

Porovnání dvou způsobů

První výhodou použití funkce Subtotal k zapnutí zlomů je, že nepotřebujete provádět metodu pro provedení tisku zprávy. Je to velmi používané v prostředí uživatele.

Zprávy z prostředí uživatele se obvykle tisknou následovně:

1. Vybrat záznamy k tisku.
2. Třídit záznamy podle jedné úrovně navíc.
3. Vybrat položku Tisk z nabídky Soubor.

4th Dimension prohlédne metodu formuláře a metody objektů, najde funkci Subtotal a zapne řízení zlomů a vytiskne zprávu. Při tomto použití funkce Subtotal jsou dvě nevýhody:

- Nemůžete použít Subtotal k aktivování zlomů ve zkompilované databázi.
- Musíte třídit pole podle jednoho pole navíc, což může zabrat určitý čas.

Při používání metod k vytvoření zprávy, je doporučeno používat příkazy BREAK LEVEL a ACCUMULATE. Proces který tiskne záznamy obvykle vypadá následovně:

1. Vybrat záznamy k tisku.
2. Třídit záznamy pomocí ORDER BY. Třídit alespoň na tolik úrovní, kolik máte úrovní zlomů.
3. Provést BREAK LEVEL a ACCUMULATE.
4. Tisknout zprávu pomocí PRINT SELECTION.

Ve zkompilované databázi musíte použít příkazy BREAK LEVEL a ACCUMULATE pro aktivaci zlomů. Nicméně funkce Subtotal je stále vyžadována pro výpočet mezisoučtů.

Dále si přečtěte

ACCUMULATE, BREAK LEVEL, level, PRINT SELECTION.

Příkazy a odkazy pro Tisk

4th Dimension 6.5, Seznam témat příkazů

Level (Úroveň)

Příkazy a odkazy pro Tisk

Verze 3

Level → Číslo

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Číslo	←	Platná úroveň zlomu nebo záhlaví

Popis

Level je určen k zjištění platné úrovně zlomu nebo záhlaví. Vrací jejich číslo během událostí V záhlaví a Při tisku zlomu.

Úroveň 0 je poslední a je určena pro tisk celkových součtů zprávy.Level vrací 1 při tisku zlomu na prvním tříděném poli, 2 když 4th Dimension tiskne zlom na druhém tříděném poli, atd.

Příklad

Tento příklad je vzor pro metodu formuláře. Ukazuje každou událost, která může nastat při tisku souhrnné zprávy, která používá výstupní formulář. Příkaz Level je volán při tisku záhlaví nebo zlomu:

```
   ` Metoda formuláře, který bude použit pro tisk souhrnné zprávy
$vpFormTabulky:=Current form table
Case of
      ` ...
   ÷ (Form event=On Header)
         ` Bude se tisknout oblast záhlaví
      Case of
         ÷ (Before selection($vpFormTabulky→))
               ` Zde bude kód pro první zlom záhlaví
         ÷ (Level = 1)
               ` Zde bude kód pro záhlaví úrovně 1
         ÷ (Level = 2)
               ` Zde bude kód pro záhlaví úrovně 2
               ` ...
      End case
   ÷ (Form event=On Printing Details)
         ` Bude se tisknout záznam
         ` Zde bude kód pro každý záznam
   ÷ (Form event=On Printing Break)
         ` Bude se tisknout oblast zlomu
      Case of
         ÷ (Level = 0)
               ` Zde bude kód pro zlom úrovně 0
         ÷ (Level = 1)
               ` Zde bude kód pro zlom úrovně 0
               ` ...
      End case
```

```
    ÷ (Form event=On Printing Footer)
        If(End selection($vpFormTabulky→))
                ` Kód pro poslední záhlaví bude zde
        Else
                ` Kód pro záhlaví bude zde
        End if
End case
```

Dále si přečtěte

ACCUMULATE, BREAK LEVEL, Form event, PRINT SELECTION.

Příkazy a odkazy pro Tisk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

PRINT RECORD (TISKNOUT ZÁZNAM)

Příkazy a odkazy pro Tisk

Verze 3

PRINT RECORD ({tabulka} {;} { *})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ze které tisknout platný záznam, pokud vynecháno, je použita platná tabulka
*		→	Potlačí dialogy pro tisk

Popis

Příkaz PRINT RECORD vytiskne platný záznam pro tabulku, bez jakohokoli zásahu do platného výběru. Pro tisk je použit platný výstupní formulář. Pokud není pro tabulku žádný platný záznam, příkaz neprovede nic.

Pomocí tohoto příkazu můžete tisknout externí objekty a podformuláře, což není možné s příkazem PRINT FORM.

Poznámka: Pokud je tento záznam změněn a ještě není uložen, příkaz vytiskne změněné hodnoty a ne hodnoty uložené v polích na disku.

Pokud předáte volitelný parametr *, nejsou zobrazeny dialogy tisku. V tomto případě je záznam tištěn s výchozím nastavením tisku, pokud neprovedete příkaz PAGE SETUP před provedením PRINT RECORD.

Příklad

Následující příklad tiskne platný záznam z tabulky [Faktury]. Je to metoda objektu pro tlačítko tisku ve vstupním formuláři. Jakmile uživatel klepne na toto tlačítko, je záznam vytištěn ve výstupním formuláři navrženém pro tento účel:

```
    ` Vybrat správný výstupní formulář pro tisk
OUTPUT FORM ([Faktury];"Tisk_ze_Vstupu dat")
    ` Tisknout fakturu tak jak je (bez zobrazení oken pro tisk)
PRINT RECORD ([Faktury];*)
    ` Vrátit předchozí výstupní formulář
OUTPUT FORM ([Faktury];"Seznam")
```

Dále si přečtěte

PRINT FORM.

Příkazy a odkazy pro Tisk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

PAGE SETUP (NASTAVENÍ STRÁNKY)

Příkazy a odkazy pro Tisk

Verze 3

PAGE SETUP ({tabulka;} formulář)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka k níž patří formulář nebo výchozí tabulka pokud vynecháno
formulář	Řetězec	→	Formulář pro který nastavit vzhled stránky

Popis

PAGE SETUP nastaví vzhled stránky pro formulář a uložit k němu tato nastavení. Vzhled stránky je uložen s formulářem když je formulář uložen v prostředí návrháře.

V následujících třech příkladech nejsou zobrazeny okna tisku a formulář je tištěn s výchozím nastavením:

- Provedení příkazu PRINT SELECTION, když k němu přidáte parametr *.
- Provedení příkazu PRINT RECORD, když k němu přidáte parametr *.
- Provedení příkazu PRINT FORM, pokud před ním neprovedete příkaz PRINT SETTINGS.

Provedení PAGE SETUP vám umožní přeskočit dialogová okna tisku A použít jiné nastavení než je výchozí.

Příklad

Pro tabulku [Design Stuff] bylo vytvořeno několik (prázdných) formulářů. K formuláři "PS100" byl přiřazen vzhled stránky se změnou měřítka 100%, k formuláři "PS90" byl přiřazen vzhled stránky se změnou měřítka 90%, atd. Následující metoda projektu vám umožní tisknout výběr záznamů se změnou měřítka, bez použití oken tisku pro toto nastavení při každém tisku:

```
    ` Metoda projektu ZMĚNA MĚŘÍTKA
    ` ZMĚNA MĚŘÍTKA ( Ukazatel ; Řetězec {; Long Integer } )
    ` ZMĚNA MĚŘÍTKA ( →[Tabulka]; "Výstupní formulář" {; Změna měřítka } )
If (Count parameters>=3)
   PAGE SETUP([Design Stuff];"PS"+String($3))
   If (Count parameters>=2)
      OUTPUT FORM($1→;$2)
   End if
End if
If (Count parameters>=1)
   PRINT SELECTION($1→;*)
Else
   PRINT SELECTION(*)
End if
```

Po uložení této metody do vaší aplikace ji můžete použít následovně:

```
    ` Hledat faktury
QUERY ([Faktury];[ Faktury]Placeno=False)
    ` Tisknout zprávu s 90% zmenšením
ZMĚNA MĚŘÍTKA (→[Faktury];"Souhrnná zpráva";90)
    ` Tisknout zprávu s 50% zmenšením
ZMĚNA MĚŘÍTKA (→[Faktury];"Podrobná zpráva";50)
```

Dále si přečtěte

PRINT FORM, PRINT RECORD, PRINT SELECTION.

Příkazy a odkazy pro Tisk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

PRINT SETTINGS (NASTAVENÍ TISKU)

PRINT SETTINGS

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz PRINT SETTINGS zobrazí okna pro nastavení tisku. Nejdříve zobrazí okno Nastavení tisku a pak okno Tisk.

Tento příkaz může umístit skupinu příkazů PRINT FORM. Tento příkaz nemá žádný vliv na tisk pomocí jiných příkazů.

Okno Tisk obsahuje zaškrtávací políčko Na obrazovku, které vám umožní vytisknout dokument nejdříve na obrazovku. Toto tlačítko můžete předem určit nebo odstranit pomocí příkazu SET PRINT PREVIEW před příkazem PRINT SETTINGS.

Příklad

Podívejte se na příklad u příkazu PRINT FORM.

Systémové proměnné a Sady

Pokud uživatel klepne na tlačítko OK, je proměnná OK nastavena na 1. Jinak je nastavena na 0.

Dále si přečtěte

PAGE BREAK, PRINT FORM, SET PRINT PREVIEW.

SET PRINT PREVIEW (NASTAVIT NÁHLED TISKU)

Verze 3

SET PRINT PREVIEW (náhled)

Parametr	Typ		Popis
náhled	Logické	→	Náhled tisku na obrazovku (TRUE), nebo žádný náhled (FALSE)

Popis

Příkaz SET PRINT PREVIEW vám umožní programově nastavit možnost tisk Na obrazovku v okně Tisk. Pokud do parametru náhled předáte True, bude toto políčko označeno a vytiskne se náhled na obrazovku, pokud předáte False, náhled se nebude tisknout. Toto nastavení je pouze pro jeden proces a nemá žádný vliv na nastavení v jiných procesech.

Příklad

Následující příklad označí políčko Na obrazovku k zobrazení výsledku hledání na obrazovku a pak jej zase odznačí.

```
QUERY([Zákazníci])
If (OK=1)
   SET PRINT PREVIEW (True)
   PRINT SELECTION ([Zákazníci] ; *)
   SET PRINT PREVIEW (False)
End if
```

Dále si přečtěte

PRINT RECORD, PRINT SELECTION, PRINT SETTINGS.

Příkazy a odkazy pro Tisk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

PRINT FORM (TISKNOUT FORMULÁŘ)

Příkazy a odkazy pro Tisk

Verze 3

PRINT FORM ({tabulka;} formulář)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka k níž patří formulář nebo výchozí tabulka pokud vynecháno
formulář	Řetězec	→	Formulář který vytisknout

Popis

PRINT FORM vytiskne formulář s platnými hodnotami polí a proměnných. Tiskne pouze oblast obsahu (oblast mezi čárou záhlaví a obsahu). Obvykle se používá pro tisk složitých zpráv, které vyžadují kompletní kontrolu nad průběhem tisku. PRINT FORM neprovádí žádné řízení záznamů, zlomů, zlomy stránek, záhlaví nebo zápatí. Všechny tyto operace musíte naprogramovat. PRINT FORM pouze vytiskne pole a proměnné v pevném rámečku.

Protože PRINT FORM neprovede žádný zlom za vytištěním formuláře, je jednoduché vložit na jednu stránku více formulářů. Tento příkaz je perfektní pro zprávy, kde vyžadujete informace z více tabulek na různých formulářích. Pro provedení zlomu stránky mezi formuláři použijte příkaz PAGE BREAK.

Okna nastavení tisku se nebudou objevovat při použití tohoto příkazu a zpráva ani nepoužije nastavení tisku které bylo přiřazeno k formuláři v prostředí návrháře. Máte dvě možnosti jak definovat nastavení tisku pro tento příkaz:

- Provést příkaz PRINT SETTINGS, čímž umožníte uživateli nastavit vlastní možnosti.
- Provést příkaz PAGE SETUP. V tomto případě nastavujete vlastnosti programově.

PRINT FORM vytváří každou tištěnou stránku nejdříve v paměti. Jakmile je tato stránka plná nebo provedete příkaz PAGE BREAK, je stránka vytištěna. Pro vytištění poslední stránky musíte použít příkaz PAGE BREAK. Protože pokud tato stránka nebude zaplněna celá, zůstane v paměti a nevytiskne se.

Upozornění: Podformulář a externí objekty nejsou tištěny pomocí tohoto příkazu. Pokud chcete vytisknout pouze jeden formulář s celým obsahem, použijte příkaz PRINT RECORD.

Příkaz PRINT FORM vytvoří pro metodu formuláře pouze událost Při tisku obsahu.

Příklad

Následující příklad provede stejnou akci jako PRINT SELECTION. Nicméně zpráva používá dva formuláře podle toho jestli je záznam pro šek nebo depozit:

```
QUERY([Register]) ` Vybrat záznamy
If (OK=1)
   ORDER BY([Register]) ` Třídit záznamy
   If (OK=1)
      PRINT SETTINGS ` Zobrazit okna tisku
         If (OK=1)
            For ($vlZaznam; 1; Records in selection([Register]))
               If ([Register]Type = "Check")
                        ` Použít formulář pro šeky
                  PRINT FORM ([Register]; "Check Out")
               Else
                        ` Použít jiný formulář pro depozit
                  PRINT FORM ([Register]; "Deposit Out")
```

```
                End if
                NEXT RECORD([Register])
            End for
        PAGE BREAK ` Ujistěte se že poslední stránka je vytištěna
    End if
  End if
End if
```

Dále si přečtěte

PAGE BREAK, PAGE SETUP, PRINT SETTINGS.

Příkazy a odkazy pro Tisk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

PAGE BREAK (ZLOMIT STRÁNKU)

Verze 3

PAGE BREAK ({* | >})

Parametr	Typ		Popis
* | >		→	* zruší tiskovou úlohu PRINT FORM, nebo > dá výhradní prioritu jen této tiskové úloze

Popis

PAGE BREAK provádí tisk dat která byla poslána na tiskárnu a vysune stránku. PAGE BREAK se používá s PRINT FORM k řízení tisku stránek a k tisku poslední stránky. Nepoužívejte tento příkaz spolu s PRINT SELECTION. Místo toho použijte Subtotal a BREAK LEVEL s jejich volitelnými parametry pro zlom stránky.

Jak parametr * tak > jsou volitelné.

Parametr * vám umožní zrušit tisk započatý příkazem PRINT FORM. Provedení okamžitě zastaví tisk.

Parametr > upraví způsob chování příkazu PAGE BREAK. Tento zápis má dva efekty:

- Zastaví tisk do doby než je opět použit přikaz PAGE BREAK bez parametrů.
- Dá výhradní prioritu této tiskové úloze. Nemohou být prováděny jiné tisky, dokud není tento dokončen.

Druhá možnost je využitelná zejména při navíjeném tisku. Tento parametr zajistí, že tisk bude spojen do jednoho souboru. Velmi tím omezíte tisk potřebný k tisku.

Příklad

Přečtěte si příklad u příkazu PRINT FORM.

Dále si přečtěte

PRINT FORM.

Příkazy a odkazy pro Tisk

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Obrázky

Příkazy a odkazy pro Obrázky

Verze 6.0

Podporované formáty

Následující tabulky ukazují různé typy formátů na Macintosh a na Windows.

Vyjmout a Vložit: Podporované formáty

	PICT	***EMF***	***WMF***	***BITMAP***
Macintosh	*Ano*	*-*	*-*	*-*
Windows	*Ano*	*Ano* *Předáné v* *PicComment*	*Ano* *Předáné v* *PicComment*	*Ano* *Převedeno do* *Macintosh PICT*

Zobrazení: Podporované formáty

	PICT	***QuickTime***	***předáné WMF***	***Předáné EMF***
Macintosh	*Ano*	*Ano*	*Ne*	*Ne*
Windows	*Ano*	*Ano* *NT & WIN 95* *+ QT 32 bitů*	*Ano*	*Ano*

ACI PACK podporované formáty (přečtěte si poznámku)

	PICT	***BMP***	***WMF***	***EMF***	***JPEG***
Macintosh	*Ano*	*Ano* *převedeno do* *Mac PICT*	*Ne*	*Ne*	*Ano* *rozbaleno do* *Mac PICT*
Windows	*Ano*	*Ano* *převedeno do* *Mac PICT*	*Ano* *předáno v* *PicComment*	*Ano* *předáno v* *PicComment*	*Ano* *rozbaleno do* *Mac PICT*

Poznámka: ACI Pack je 4D Plug-in od ACI, obsažený ve 4th Dimension.

Apple QuickTime komprese

Apple používá QuckTime k předání nových technologií komprese, jako je JPEG. Apple má předáný překladač na originální PICT, takže aplikace na Macintoshi mohou zacházet s obrázky v QuckTime bez dalších úprav. Pokud se aplikace ptá systému po nakreslení obrázku který obsahuje předáná QuickTime data, bitmapa je rozbalena a zobrazena, jestliže máte QuckTime: pokud nemáte instalovaný QuickTime je operační kód QuckTime ignorován. Tato techologie je výhodná pro uživatele a zabere minimum paměti, protože obrázek o velikosti 1 MB může být uložen do 40 KB PICT, a nepotřebuje být rozbalen před zobrazením.

Kompresní typy QuickTime

Následující seznam ukazuje kompresní typy QuickTime:

• **Foto komprese** (`jpeg`): Používá Joint Photographic Experts Group (JPEG)algoritmus pro kompresi obrázků.

JPEG je mezinárodní standart pro trvalou kompresi obrázků.
• **Video komprese** (`rpza`): Umožní velmi rychlé rozbalení při práci s obrázkem s vysokou kvalitou.
• **Komprese animací** (`rle`): Používá se na animace a počítačem vygenerované video
• **Čistá komprese** (`raw`): Omezí požadavky na uložení obrázku konvertováním hloubky bodů obrázku.
• **Grafická komprese** (`smc`): Alternativa ke kompresi animací, ale má nižší kvalitu.
• **Kompaktní video komprese** (sdvc): Stejné jako video komprese, ale nabízí větší kompresi, vyšší kvalitu a rychlejší přehrání.

Při definování typu komprese,se ujistěte o předání mezery do parametru metody. Pokud je metoda prázdný řetězec, je obrázek otevřen ale není zabalen.

Chyby při kompresi obrázků

Pokud použijete příkaz pro kompresi obrázků a nemáte ve svém systému instalovaný QuickTime, 4th Dimension vrátí chybu -9955. Z QuckTime mohou být vráceny i jiné chyby.Můžete tyto chyby zachytit pomocí příkazu ON ERR CALL.

Použití Apple QuickTime na Windows

• Protože jak Altura tak 4th Dimension jsou 32-bitové aplikace, potřebujete nainstalovat 32-bitový QuckTime pro Windows (verze 2.1.1 b50 nebo vyšší)
• QuickTime pro Windows nepodporuje Windows 3.11 a proto není žádný způsob jak zobrazit QuickTime obrázky na Windows 3.11.
• Další pro zobrazení obrázku je, že QuickTime pro Windows musí používat pracovní soubor na disku. Při každém zobrazení QuickTime obrázku je nejdříve jeho kopie uložena na disk a pak je vymazána. Obvykle zůstane tento dočasný soubor v cache a neukládá se přímo na disk. Proto jsou tyto operace velmi rychlé. Nicméně na pomalém PC s nedostatkem paměti budou tyto operace pomalejší.
• QuickTime pro Windows může zobrazit pouze QuckTime obrázky - není žádný způsob jak zabalit jiné obrázky. Toto je důvod, proč příkazy 4th Dimension pracující s QuickTime kompresí nefungují na Windows (a nebudou dokud QuickTime pro Windows nebude podporovat kompresi). Na Windows nefungují následující příklady: COMPRESS PICTURE, COMPRESS PICTURE FILE, LOAD COMPRESSED PICTURE FROM FILE.

SAVE PICTURE TO FILE nepoužívá QuickTime, a tak pracuje na Windows stejně jako na Macintoshi. Vytvoří Macintosh PICT soubor, který může být na Windows otevřen rozšířenými aplikacemi jako je Photoshop.

Dále si přečtěte

COMPRESS PICTURE, COMPRESS PICTURE FILE, LOAD COMPRESSED PICTURE FROM FILE, PICTURE PROPERTIES, Picture size, SAVE PICTURE TO FILE.

Příkazy a odkazy pro Obrázky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

COMPRESS PICTURE (ZABALIT OBRÁZEK)

Verze 3

COMPRESS PICTURE (obrázek; metoda; kvalita)

Parametr	Typ		Popis
obrázek	Obrázek	→	Obrázek k zabalení
		←	Zabalený obrázek
metoda	Řetězec	→	4 znakový řetězec metody komprese
kvalita	Číslo	→	Kvalita komprese (1...1000)

Popis

Příkaz COMPRESS PICTURE zabalí obrázek obsažený v poli nebo v proměnné.

Parametr metoda je 4 znakový řetězec definující typ komprese.

Parametr kvalita je Integer mezi 1 a 1000, který definuje kvalitu zabalení obrázku. Nižší kvalita samozřejmě dovolí větší zabalení.

Upozornění: Rozsah zabalení je závislý na velikosti a typu obrázku. Komprese malého obrázku nemusí přinést žádné zmenšení velikosti.

Dále si přečtěte

COMPRESS PICTURE FILE, LOAD COMPRESSED PICTURE FROM FILE, Obrázky.

Příkazy a odkazy pro Obrázky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

LOAD COMPRESSED PICTURE FROM FILE

(NAČÍST ZABALENÝ OBRÁZEK ZE SOUBORU)

Příkazy a odkazy pro Obrázky

Verze 3

LOAD COMPRESSED PICTURE FROM FILE (dokument; metoda; kvalita; obrázek)

Parametr	Typ		Popis
dokument	DocReh	→	Číslo odkazu na dokument
metoda	Řetězec	→	4 znakový řetězec kompresní metody
kvalita	Číslo	→	Kvalita komprese (1...1000)
obrázek	Obrázek	←	Zabalený obrázek

Popis

Tento příkaz načte a zabalí obrázek ze souboru na disku.

Můžete otevřít dokument PICT s použitím příkazu Open document. Můžete pak použít odkaz na dokument, který vrátí tento příkaz k načtení a zabalení obrázku PICT nalezeného v dokumentu. Příkaz načte obrázek do paměti, zabalí jej metodou kterou vyberete a pak jej uloží na obrázek.

Obrázek je načten do paměti před tím, než je zabalen. Pokud nemáte dostatek paměti k načtení obrázku, použijte příkaz COMPRESS PICTURE FILE před provedením LOAD COMPRESSED PICTURE FROM FILE.

Parametr metoda je 4 znakový řetězec definující metodu zabalení.

Parametr kvalita je Integer hodnota mezi 1 a 1000 která definuje kvalitu komprese. Nižší kvalita samozřejmě dovolí větší zabalení.

Upozornění: Rozsah zabalení je závislý na velikosti a typu obrázku. Komprese malého obrázku nemusí přinést žádné zmenšení velikosti.

Příklad

Následující příklad umožní vybrat soubor formátu PICT. Tento obrázek je načten do paměti, zabalen a pak uložen do proměnné. Pak je soubor uzavřen.

```
vRef:=Open document ("";"PICT")
If (OK=1)
   LOAD COMPRESS PICTURE FROM FILE(vRef;"jpeg";500;vPict)
   CLOSE DOCUMENT(vRef)
End if
```

Dále si přečtěte

COMPRESS PICTURE, COMPRESS PICTURE FILE, Obrázky, SAVE PICTURE TO FILE.

Příkazy a odkazy pro Obrázky

4th Dimension 6.5, Seznam témat příkazů

COMPRESS PICTURE FILE (ZABALIT SOUBOR OBRÁZKU)

Verze 3

COMPRESS PICTURE FILE (dokument; metoda; kvalita)

Parametr	Typ		Popis
dokument	DocReh	→	Číslo odkazu na dokument
metoda	Řetězec	→	4 znakový řetězec kompresní metody
kvalita	Číslo	→	Kvalita komprese (1...1000)

Popis

Příkaz zabalí soubor obrázku na disku. Použijte tento příkaz na zabalení obrázku, který nemůžete načíst do paměti. Jednou zabalený, může být načten příkazem LOAD COMPRESSED PICTURE FROM FILE.

Parametr metoda je 4 znakový řetězec definující metodu zabalení.

Parametr kvalita je Integer hodnota mezi 1 a 1000 která definuje kvalitu komprese. Nižší kvalita samozřejmě dovolí větší zabalení.

Upozornění: Rozsah zabalení je závislý na velikosti a typu obrázku. Komprese malého obrázku nemusí přinést žádné zmenšení velikosti.

Příklad

Následující příklad umožní vybrat soubor formátu PICT. Budou zobrazeny pouze obrázky PICT. Tento obrázek je zabalen, načten do paměti a pak uložen do proměnné. Pak je soubor uzavřen.

```
vRef:=Open document ("";"PICT")
If (OK=1)
   COMPRESS PICTURE FILE (vRef; "jpeg";500)
   LOAD COMPRESS PICTURE FROM FILE(vRef;"jpeg";500;vPict)
   CLOSE DOCUMENT(vRef)
End if
```

Dále si přečtěte

COMPRESS PICTURE, LOAD COMPRESSED PICTURE FROM FILE, SAVE PICTURE TO FILE.

Příkazy a odkazy pro Obrázky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SAVE PICTURE TO FILE (ULOŽIT OBRÁZEK DO SOUBORU)

Verze 3

SAVE PICTURE TO FILE (dokument; obrázek)

Parametr	Typ		Popis
dokument	DocRef	→	Číslo odkazu na dokument
obrázek	Obrázek	→	Obrázek k uložení

Popis

Tento příkaz uloží obrázek do dokumentu který byl vytvořen pomocí Create document.

Příklad

Následující příklad vytvoří dokument a uloží do něj obrázek:

```
vRef:=Create document("";"PICT")
If (OK=1)
   SAVE PICTURE TO FILE(vRef;vPict)
   CLOSE DOCUMENT(vRef)
End if
```

Dále si přečtěte

COMPRESS PICTURE FILE, LOAD COMPRESSED PICTURE FROM FILE.

Příkazy a odkazy pro Obrázky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

PICT TO GIF (PICT DO GIF)

Příkazy a odkazy pro Obrázky

Verze 6.5

PICT TO GIF (obr; blobGIF)

Parametr	Typ		Popis
obr	Obrázek	→	Obrázkové pole nebo proměnná
blobGIF	BLOB	←	BLOB obsahující GIF obrázek

Popis

Příkaz PICT TO GIF vám umožní převést PICT obrázek uložený v proměnné nebo poli do GIF obrázku.

Do parametru obr vložte obrázkové polo nebo proměnnou do parametru blobGIF vložte BLOB. Po provedení příkazu bude blobGIF obsahovat převedený obrázek.

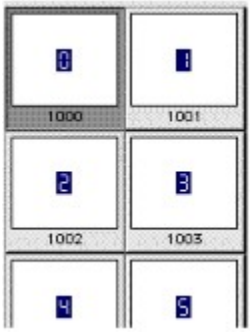
Poznámka: GIF obrázek nemůže obsahovat více jak 256 barev. Pokud originální PICT obrázek obsahuje více barev, mohou být některé ztracené. GIF vytvořený tímto příkazem je optimalizovaný na barvy v originálním obrázku. GIF typ vytvořeného obrázku je typu 87a (neprůhledný) nebo normal (neprokládaný).

Obrázek uložený v blobGIF pak můžete uložit do souboru pomocí příkazu BLOB TO DOCUMENT, nebo jej můžete publikovat na Webu.

Pokud byl převod správně dokončen, je proměnná OK nastavena na 1 jinak je nastavena na 0.

Příklad

Uvažujme, že potřebujete vytvořit GIF obrázek za běhu zobrazením čítače připojení. Do knihovny obrázků databáze umístíte všechna čísla jako obrázky:



Do metody databáze Při Web spojení předáte následující kód:

```
If (Web Context)
   ...
Else
   C_BLOB ($blob)
   Case of
      ...
      ÷ ($1="/4dcgi/counter") `Vytvoření GIF čítače
            `Když 4D detekuje tuto URL při posílání statické stránky
         $blob:=gifcounter (<>nbHits) `vypočítá GIF obrázek
            ` Proměnná <>nbHits obsahuje počet spojení
```

```
            SEND HTML BLOB ($blob;"image/gif")
               `Vložit obrázek a poslat na prohlížeč
            ...
      End case
   End if
```

Zde je metoda gifcounter:

```
   C_LONGINT($1)
   C_PICTURE($img)
   C_BLOB($0)
   If ($1=0)
      $ndigits:=1
   Else
      $ndigits:=1+Length(String($1))
   End if
   If ($ndigits<5)
      $ndigits:=5
   End if
   $div:=10^($ndigits-1)
   For ($i;1;$ndigits)
      $ref:=Int($1/$div)%10
      GET PICTURE FROM LIBRARY($ref+1000;picture)
      $img:=$img+picture
      $div:=$div/10
   End for
   PICT TO GIF($img;$0)
```

Při poslání stránky na Web prohlížeč, 4D zobrazí GIF obrázek který může vypadat následovně:

Příkazy a odkazy pro Obrázky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Picture size (Velikost obrázku)

Příkazy a odkazy pro Obrázky

Verze 3

Picture size (obrázek) → Číslo

Parametr	Typ		Popis
obrázek	Obrázek	→	Obrázek pro který zjistit velikost
Výsledek funkce	Číslo	←	Velikost obrázku v bytech

Popis

Výsledek této funkce je velikost obrázku udaná v bytech.

Dále si přečtěte

PICTURE PROPERTIES.

Příkazy a odkazy pro Obrázky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

PICTURE PROPERTIES (VLASTNOSTI OBRÁZKU)

Verze 6.0

PICTURE PROPERTIES (obrázek; šířka; výška{; hPosun{; vPosun{; mod}}})

Parametr	Typ		Popis
obrázek	Obrázek	→	Obrázek (pole | proměnná), pro který chceme informaci
šířka	Číslo	←	Šířka obrázku vyjádřená v bodech
výška	Číslo	←	Výška obrázku vyjádřená v bodech
hPosun	Číslo	←	Horizontální posun při zobrazení na pozadí
vPosun	Číslo	←	Vertikální posun při zobrazení na pozadí
mod	Číslo	←	Mód přenosu při zobrazení na pozadí

Popis

Příkaz PICTURE PROPERTIES vrací informace o obrázku, který předáte do parametru obrázek.

Parametry šířka a výška vrací šířku a výšku obrázku.

Parametry hPosun, vPosun a mod vrací horizontální a vertikální pozici a mód převodu obrázku při zobrazení na pozadí formuláře.

Dále si přečtěte

Picture size.

Příkazy a odkazy pro Obrázky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

PICTURE LIBRARY LIST (SEZNAM OBRÁZKŮ Z KNIHOVNY)

Příkazy a odkazy pro Obrázky

Verze 6.0.2

PICTURE LIBRARY LIST (picRefs; picNázvy)

Parametr	Typ		Popis
picRefs	Číselný array	→	Čísla odkazů grafik v knihovně obrázků
picNázvy	Řetězcový array	←	Názvy grafik v knihovně obrázků

Popis

Příkaz PICTURE LIBRARY LIST vrací čísla odkazů a názvy obrázků které jsou uložené v knihovně obrázků.

Po provedení příkazu dostanete tyto odkazy do array picRefs a názvy obrázků budou v array picNázvy. Tyto dvě array jsou synchronizované: n-tý prvek v array picRefs je odkaz na obrázek, jehož název je uložen v n-tém prvku array picNázvy.

Array picRefs může být Real, Integer nebo Long Integer. Pokud nebyl tento array definován před naplněním příkazem PICTURE LIBRARY LIST, je jako výchozí v nezkompilované verzi nastaven na Real.

Array picNázvy může být řetězec nebo text. Pokud nebyl tento array definován před naplněným příkazem PICTURE LIBRARY LIST, je jako výchozí v nezkompilované verzi nastaven na Text.

Maximální délka názvu obrázku v Knihovně obrázků je 31 znaků. Pokud k array picNázvy použijete typ řetězec, vytvořte jeho délku tak, aby nemohlo dojít k tomu, že se název do prvku nevejde.

Pokud v knihovně nejsou žádné obrázky, jsou oba array vráceny prázdné.

Pokud budete chtít zjistit, kolik je obrázků v knihovně, můžete použít příkaz Size of array k zjištění velikosti jedné z array.

Příklady

1. Následující kód vrací katalog Knihovny obrázků v array alPicRef a asPicNázev:

```
PICTURE LIBRARY LIST (alPicRef; asPicNázev)
```

2. Následující příklad testuje jestli je knihovna obrázků prázdná:

```
PICTURE LIBRARY LIST(alPicRef;asPicName)
If (Size of array(alPicRef)=0)
   ALERT("The Picture Library is empty.")
Else
   ALERT("Knihovna obrázků obsahuje "+String(Size of array(alPicRef))+" obrázků.")
End if
```

3. Následující přílad exportuje Knihovnu obrázků na disk:

```
PICTURE LIBRARY LIST($alPicRef;$asPicName)
$vlNbPictures:=Size of array($alPicRef)
If ($vlNbPictures>0)
   SET CHANNEL(12;"")
   If (OK=1)
      $vsTag:="4DV6PICTURELIBRARYEXPORT"
      SEND VARIABLE($vsTag)
```

```
            SEND VARIABLE($vlNbPictures)
            gError:=0
            For($vlPicture;1;$vlNbPictures)
                $vlPicRef:=$alPicRef{$vlPicture}
                $vsPicName:=$asPicName{$vlPicture}
                GET PICTURE FROM LIBRARY(alPicRef{$vlPicture};$vgPicture)
                If (OK=1)
                    SEND VARIABLE($vlPicRef)
                    SEND VARIABLE($vsPicName)
                    SEND VARIABLE($vgPicture)
                Else
                    $vlPicture:=$vlNbPictures+1
                    gError:=-108
                End if
            End for
            SET CHANNEL(11)
            If (gError#0)
                ALERT("Knihovna obrázků nemůže být exportována, opakujte s více pamětí.")
                DELETE DOCUMENT (Document)
            End if
        End if
    Else
        ALERT("Knihovna obrázků je prázdná.")
    End if
```

Dále si přečtěte

GET PICTURE FROM LIBRARY, REMOVE PICTURE FROM LIBRARY, SET PICTURE TO LIBRARY.

Příkazy a odkazy pro Obrázky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GET PICTURE FROM LIBRARY (ZÍSKAT OBRÁZEK Z KNIHOVNY)

Verze 6.0.2

GET PICTURE FROM LIBRARY (picRef; obrázek)

Parametr	Typ		Popis
picRef	Číslo	→	Číslo odkazu grafiky v knihovně obrázků
obrázek	Obr. proměnná	←	Obrázek z knihovny obrázků

Popis

Příkaz GET PICTURE FROM LIBRARY vrací obrázek z knihovny obrázků definovaný parametrem PicRef do parametru obrázek.

Pokud není žádný obrázek se zadaným číslem, zůstane obrázek nezměněný.

Příklady

1. Následující příklad vrací do vgMůjObr obrázek, jehož číslo odkazu je uloženo v proměnné $vlPicRef:

```
GET PICTURE FROM LIBRARY($vlPicRef; vgMůjObr)
```

2. Podívejte se na třetí příklad u příkazu PICTURE LIBRARY LIST.

Dále si přečtěte

PICTURE LIBRARY LIST, REMOVE PICTURE FROM LIBRARY, SET PICTURE TO LIBRARY.

Systémové proměnné a Sady

Pokud obrázek v knihovně existuje, je proměnná OK nastavena na 1, jinak je nastavena na 0.

Ovládání chyb

Pokud není dostatek paměti k uložení obrázku, je generována chyba -108. Můžete tuto chybu zachytit pomocí metody pro zachytávání chyb.

SET PICTURE TO LIBRARY (PŘIDAT OBRÁZEK DO KNIHOVNY)

Příkazy a odkazy pro Obrázky

Verze 6.0.2

SET PICTURE TO LIBRARY (obrázek; picRef; picNázev)

Parametr	Typ		Popis
Obrázek	Obrázek	→	Nový obrázek
picRef	Číslo	→	Číslo odkazu na obrázek v knihovně obrázků
picNázev	Řetězec	→	Nový název obrázku v knihovně obrázků

Popis

Příkaz SET PICTURE TO LIBRARY vytvoří nový obrázek nebo nahradí exitující v Knihovně obrázků.

Před provedením předáte:

• Číslo odkazu na obrázek do picRef (v rozsahu 1...32767)
• obrázek samotný do obrázek
• Název obrázku do picNázev (maximální délka: 31 znaků).

Pokud v Knihovně obrázků exituje obrázek se stejným číslem, je jeho obsah nahrazen novým obrázkem a je přejmenován na název který zadáte do parametru picNázev.

Pokud neexistuje takový obrázek, je vytvořen nový a je určeno jeho číslo a název podle parametrů které zadáte.

4D Server: Příkaz SET PICTURE TO LIBRARY nemůže být použit v metodě, která se provádí na serveru (uložená procedura nebo trigger). Pokud provedete SET PICTURE TO LIBRARY na serveru, nestane se nic a příkaz je ignorován.

Upozornění: Objekty návrháře (hierarchické seznamy, položky nabídky, atd) se mohou odkazovat na různé obrázky v knihovně obrázků. Použijte varování při upravování obrázků v knihovně obrázků pomocí jazyka.

Poznámka: Pokud do parametru obrázek nepředáte nic, nebo do picRef předáte nulu nebo zápornou hodnotu, příkaz neprovede nic.

Příklady

1. Nezáleží na tom, co je obsaženo v knihovně, následující příklad vloží nový obrázek pod jedinečné číslo, které si vytvoří.

```
PICTURE LIBRARY LIST($alPicRef;$asPicNames)
Repeat
   $vlPicRef:=1+Abs(Random)
Until (Find in array($alPicRef;$vlPicRef)<0)
SET PICTURE TO LIBRARY(vgPicture;$vlPicRef;"New Picture")
```

2. Následující příklad naimportuje do Knihovny obrázků obrázky (uložené na disku) vytvořené třetím příkladem u příkazu PICTURE LIBRARY LIST:

```
SET CHANNEL(10;"")
If (OK=1)
   RECEIVE VARIABLE($vsTag)
   If ($vsTag="4DV6PICTURELIBRARYEXPORT")
      RECEIVE VARIABLE($vlNbPictures)
      If ($vlNbPictures)
```

```
            For($vlPicture;1;$vlNbPictures)
               RECEIVE VARIABLE($vlPicRef)
               If (OK=1)
                  RECEIVE VARIABLE($vlPicName)
               End if
               If (OK=1)
                  RECEIVE VARIABLE ($vgPicture)
               End if
               If (OK=1)
                  SET PICTURE TO LIBRARY($vgPicture;$vlPicRef;$vlPicName)
               Else
                  $vlPicture:=$vlNbPictures+1
                  ALERT("Tento soubor vypadá jako poničený.")
               End if
            End for
         Else
            ALERT("Tento soubor vypadá jako poničený.")
         End if
      Else
         ALERT("Soubor ""+Document+"" není exportní soubor Knihovny obrázků.")
      End if
      SET CHANNEL(11)
   End if
```

Dále si přečtěte

GET PICTURE FROM LIBRARY, PICTURE LIBRARY LIST, REMOVE PICTURE FROM LIBRARY.

Systémové proměnné a Sady

Nic není ovlivněno.

Ovládání chyb

Pokud není dostatek paměti k předání obrázku do Knihovny obrázků, je generována chyba -108. Nezapomeňte že mohou být také vytvořeny chyby 1/0 (tzn. že je uzamčena struktura). Můžete tyto chyby zachytit pomocí metody na ovládání chyb.

Příkazy a odkazy pro Obrázky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

REMOVE PICTURE FROM LIBRARY (ODSTRANIT OBRÁZEK Z KNIHOVNY)

Verze 6.0.2

REMOVE PICTURE FROM LIBRARY (picRef)

Parametr	Typ		Popis
picRef	Číslo	→	Číslo odkazu na obrázek v knihovně obrázků

Popis

Příkaz REMOVE PICTURE FROM LIBRARY odstraní z Knihovny obrázků obrázek, jehož číslo jste zadali pomocí parametru picRef.

Pokud neexistuje obrázek s takovým číslem, příkaz neprovede nic.

4D Server: Příkaz SET PICTURE TO LIBRARY nemůže být použit v metodě, která se provádí na serveru (uložená procedura mebo trigger). Pokud provedete SET PICTURE TO LIBRARY na serveru, nestane se nic a příkaz je ignorován.

Upozornění: Objekty návrháře (hierarchické seznamy, položky nabídky, atd) se mohou odkazovat na různé obrázky v knihovně obrázků. Použijte varování při upravování obrázků v knihovně obrázků pomocí jazyka.

Příklady

1. Následující příklad vymaže obrázek číslo #4444 z knihovny obrázků.

```
REMOVE PICTURE FROM LIBRARY (4444)
```

2. Následující příklad vymaže z knihovny obrázků všechny obrázky, jejichž názvy začínají na znak ($):

```
PICTURE LIBRARY LIST($alPicRef;$asPicName)
For($vlPicture;1;Size of array($alPicRef))
   If ($asPicName{$vlPicture}="$@")
      REMOVE PICTURE FROM LIBRARY($alPicRef{$vlPicture})
   End if
End for
```

Dále si přečtěte

GET PICTURE FROM LIBRARY, PICTURE LIBRARY LIST, SET PICTURE TO LIBRARY.

Příkazy a odkazy pro Obrázky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Semaphore (Semafor)

Příkazy a odkazy pro Procesy (komunikace)

Verze 3

Semaphore (semafor{; počet tiků}) → Logické

Parametr	Typ		Popis
semafor	Řetězec	→	Semafor který bude testován a nastaven
počet tiků	Integer	→	maximální čas, po který se bude čekat na uvolnění
Výsledek funkce	Logické	←	Semafor byl úspěšně nastaven (False) nebo semafor již existuje (True)

Popis

Semafor je "příznak" přenášený na všechny stanice (každý počítač uživatelů) a na jedné stanici do všech procesů. Semafor jednoduše existuje nebo neexistuje.Každá metoda uživatele může testovat existenci semaforu. Vytvořením a testováním semaforů, mohou metody komunikovat mezi pracovními stanicemi.

Funkce Semaphore vrací True, pokud semafor existuje. Pokud neexistuje, je vytvořen a funkce vrátí False. Pouze jeden uživatel najednou může semafor vytvořit. Pokud Semaphore vrátí False, znamená to, že semafor neexistoval, ale také to znamená že byl nastaven pro proces ve kterém byl proveden příkaz.

Semaphore vrací False, pokud semafor nebyl nastaven. Také vrací False jestliže byl proces nastaven pro stejný proces ve kterém byla provedena funkce Semaphore. Název semaforu je omezen na 30 znaků včetně předpon (<>, $). Pokud předáte delší řetězec, Semaphore bude testovat oříznutý řetězec.

Volitelný parametr počet tiků vám umožní nastavit čekací dobu (v ticích) jestli byl semafor nastaven. V tomto případě bude funkce čekat dokud semafor neuvolní, nebo bude čekat dokud nevrátí True.

Ve 4th Dimension jsou dva typy semaforů. Místní a globální.

Místní semafor je dostupný ze všech procesů na jednom počítači. Místní semafor vytvoříte tím, že před jeho název předáte znak dolaru ($). Místní semafory se používají k hlídání operací na jednom počítači. Například může být místní semafor použit k hlídání použití meziprocesní array dostupné všem procesům na jednom počítači v jednouživatelské databázi.

Globální semafory jsou dostupné všem uživatelům ve všech jejich procesech. Globální semafory se používají k řízení operací pro všechny uživatele na víceuživatelské databázi.

Globální a místní semafory jsou stejné ve svém základu. Rozdíl je v jejich rozsahu. Na 4D Server jsou globální semafory dostupné ve všech procesech všech klientů. Místní semafory jsou dostupné pouze z procesů běžících na jedné stanici klienta, kde byly vytvořeny.

Ve 4th Dimension jsou globální a místní semafory stejné, protože jsou použity pouze na jednoho uživatele. Nicméně pokud plánujete databázi využít pro více uživatelů, ujistěte se o jejich použití a vyberte typ podle toho, jak budou používány.

Semafory nemusíte používat k hlídání přístupu k záznamům, protože to automaticky dělá 4th Dimension i 4D Server. Použijte je k zabránění několika uživatelům provádět stejnou operaci ve stejný čas.

Příklady

1. V tomto příkladu chcete zabránit dvěma uživatelům, aby zároveň prováděli obnovování cen v tabulce Produkty. Následující metoda používá semofory,aby tomu zabránila:

```
If (Semaphore("UpdateCenas")) ` Zkusit vytvořit semafor
```

```
    ALERT("Jiný uživatel již upravuje ceny. Opakujte později.")
  Else
    DoUpdateCenas ` Upravit všechny ceny
    CLEAR SEMAPHORE("UpdateCenas")) ` Odstranit semafor
  End if
```

2. Následující příklad používá místní semafor. V databázi s mnoha procesy, chcete udržovat seznam UDĚLAT. Chcete mít tento seznam v meziprocesní array a ne v tabulce. Použijete semafor k zabránění vícerého přístupu. V tomto případě stačí použít místní semafor, protože jste jediný uživatel.

Meziprocesní array je nastavena v metodě databáze Při provádění:

```
ARRAY TEXT (<>UDĚLATSeznam;0) ` Na začátku je tento seznam prázdný
```

Zde je metoda pro předání položek do seznamu UDĚLAT:

```
    ` Metoda projektu Přidat do UDĚLATSeznam
    ` Přidat do UDĚLATSeznam ( Text )
    ` Přidat do UDĚLATSeznam ( položka seznamu UDĚLAT )
C_TEXT($1)
If(Not(Semaphore("$AccessToDoList";300)))
      ` Počkat 5 sekund jestliže semafor existuje
  $vlElem:=Size of array(<>UDĚLATSeznam)+1
  INSERT ELEMENT(<>UDĚLATSeznam;$vlElem)
  <>UDĚLATSeznam{$vlElem}:=$1
  CLEAR SEMAPHORE("$AccessToDoList") ` Odstranit semafor
End if
```

Tuto metodu můžete volat ze všech procesů.

Dále si přečtěte

CLEAR SEMAPHORE, Te

Příkazy a odkazy pro Procesy (komunikace)

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CLEAR SEMAPHORE (ODSTRANIT SEMAFOR)

Verze 3

CLEAR SEMAPHORE (semafor)

Parametr	Typ		Popis
semafor	Řetězec	→	Semafor k odstranění

Popis

CLEAR SEMAPHORE odstraní semafor, který byl před tím vytvořený pomocí Semaphore.

Všechny semafory, které byly vytvořeny lze i odstranit. Pokud nejsou odstraněny, zůstávají v paměti dokud proces, který je vytvořil, není ukončen. Proces může odstranit pouze procesy, které sám vytvořil. Pokud se pokusíte odstranit proces, který nebyl v tomto procesu vytvořen, nestane se nic.

Příklad

Podívejte se na příklad u příkazu Semaphore.

Dále si přečtěte

Semaphore.

Příkazy a odkazy pro Procesy (komunikace)

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Test semaphore (Testovat semafor)

Příkazy a odkazy pro Procesy (komunikace)

Verze 6.5

Test semaphore (semafor) → Logické

Parametr	Typ		Popis
semafor	Řetězec	→	Název semaforu k testování
Výsledek funkce	Logické	←	True = semafor existuje False = Semafor neexistuje

Popis

Příkaz Test semaphore vám umožní testovat existenci semaforu.

Rozdíl mezi funkcemi Semaphore a Test semaphore je v tom, že Test semaphore semafor nevytvoří pokud neexistuje. Pokud semafor existuje, pak funkce vrátí True, jinak vrací False.

Příklad

Následující příklad umožní znát stav procesu (v našem případě, ve chvíli kdy mění kód) bez změnění semaforu:

```
$Win:=Open window (x1;x2;y1;y2;-Palette window)
Repeat
   If (Test semaphore("Encrypting code"))
      POSITION MESSAGE ($x3;$y3)
      MESSAGE("Kódovací kód je upravován.")
   Else
      POSITION MESSAGE($x3;$y3)
      MESSAGE("Upravování kódování bylo dokončeno.")
   End if
Until (StopInfo)
CLOSE WINDOW
```

Dále si přečtěte

Semaphore.

Příkazy a odkazy pro Procesy (komunikace)

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CALL PROCESS (ZAVOLAT PROCES)

Příkazy a odkazy pro Procesy (komunikace)

Verze 3

CALL PROCESS (proces)

Parametr	Typ		Popis
proces	Číslo	→	Číslo procesu

Popis

CALL PROCESS zavolá formulář v okně na popředí procesu.

Důležité: CALL PROCESS pracuje pouze mezi procesy na stejném počítači.

Pokud voláte proces který neexistuje, nestane se nic.

Pokud je cílový proces zobrazený na popředí, nic se nestane. Formulář zobrazený v cílovém procesu vrátí událost Při vnějším volání. Tato událost musí být aktivována v okně **Vlastnosti formuláře** v prostředí návrháře a musíte jí řídit v metodě formuláře. Pokud tato událost není označena pro proces, nebo k ní není přiřazen žádný kód v metodě formuláře, nic se nestane.

Volající proces (proces ze kterého je proveden příkaz CALL PROCESS) nečeká - CALL PROCESS má okamžitý účinek. Pokud je to potřeba, musíte napsat čekací smyčku pro počkání na odpověď z volaného procesu. Použijte k tomu meziprocesní proměnnou nebo procesní proměnnou (vytvořenou pro tento účel) kterou můžete měnit mezi jednotlivými procesy (s použitím GET PROCESS VARIABLE a SET PROCESS VARIABLE).

Pro komunikaci mezi procesy které nezobrazují formulář, použijte příkazy GET PROCESS VARIABLE a SET PROCESS VARIABLE.

CALL PROCESS má alternativní zápis CALL PROCESS(-1).

Ab se zbytečně nezpomaloval průběh metod, 4th Dimension nepřekresluje všechny meziprocesní proměnné ihned jak jsou změněny. Pokud předáte -1 místo čísla procesu, příkaz nezavolá žádný proces, ale obnoví všechny meziprocesní proměnné, které jsou zobrazeny ve formulářích na jednom počítači.

Příklad

Podívejte se na příklad u metody databáze Při ukončení.

Dále si přečtěte

Form event, GET PROCESS VARIABLE, SET PROCESS VARIABLE.

Příkazy a odkazy pro Procesy (komunikace)

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GET PROCESS VARIABLE (ZÍSKAT PROMĚNNOU Z PROCESU)

Příkazy a odkazy pro Procesy (komunikace)

Verze 6.0

GET PROCESS VARIABLE (proces; zdrojProm; cílProm{; zdrojProm2; cílProm2; ...; zdrojPromN; cílPromN})

Parametr	Typ		Popis
proces	Číslo	→	Číslo zdrojového procesu
zdrojProm	Proměnná	→	Zdrojová proměnná
cílProm	Proměnná	←	Cílová proměnná

Popis

Příkaz GET PROCESS VARIABLE přečte proměnnou zdrojProm (zdrojProm2, atd) ze zdrojového procesu, jehož číslo jste zadali do parametru proces, a umístí její hodnotu do proměnná cílProm (cílProm2, atd) umístěné v platném procesu.

Každá zdrojová proměnná může být proměnná, array nebo prvek array. Podívejte se na rozdělení dále v této části.

Každá dvojice proměnných zdrojProm; cílProm musí být stejného typu, jinak hodnota, kterou přenášíte, nemusí být správná.

Platbý proces si proměnnou "vytáhne" ze zdrojového procesu - zdrojový proces není nikdy upozorněn, že je čtena jeho proměnná.

4D Server: S použitím 4D Client můžete napsat proměnné v cílovém procesu prováděné na stroji serveru (uložené procedury). Aby jste to udělali, vložte znaménko mínus před číslo procesu.

TIP: Pokud nastavíte ID procesu serveru, můžete stále použít meziprocesní proměnnou serveru. K tomu můžete použít jakékoli záporné číslo procesu. Jinými slovy není nutné znát číslo procesu k použití meziprocesní proměnné v příkazu Set process variable. Je to použitelné pokud je uložená procedura spuštěna s použitím metody databáze Při spuštění serveru. Počítač klienta neví číslo tohoto procesu, může být použita jakákoli záporná hodnota v parametru proces.

Omezení

GET PROCESS VARIABLE nepřijímá místní proměnnou jako zdrojovou proměnnou.

Na druhou stránku může být cílová proměnná meziprocesní, procesní i místní. Hodnotu můžete ukládat pouze do proměnných, nikoli do pole.

GET PROCESS VARIABLE přijímá všechny typy proměnných, mimo:

- Ukazatele
- Array ukazatelů
- Dvojrozměrné array

Zdrojový proces musí být proces uživatele: nemůže to být kernel proces. Pokud zdrojový proces neexituje, příkaz neprovede nic.

Poznámka: V nezkompilované databázi, pokud zdrojová proměnná neexistuje, je přenesena hodnota Nedefinováno. Cílovou proměnnou proto můžete testovat příkazem Type.

Příklady

1. Tento řádek kódu přečte hodnotu textové proměnné vtCurStatus z procesu jehož číslo je v $vlProces. Vrátí její hodnotu do proměnné procesu vtInfo umístěné v platném procesu:

```
GET PROCESS VARIABLE($vlProces;vtCurStatus;vtInfo)
```

2. Tento řádek provede to samé, ale uloží hodnotu do místní proměnné $vtInfo pro metodu, která příkaz provádí:

```
GET PROCESS VARIABLE($vlProces;vtCurStatus;$vtInfo)
```

3. Následující řádek provede to samé, ale uloží hodnotu do proměnné procesu vtCurStatus pro proces ze kterého je příkaz proveden:

```
GET PROCESS VARIABLE($vlProces;vtCurStatus;vtCurStatus)
```

Poznámka: První výskyt proměnné vtCurStatus je proměnná ve zdrojovém procesu a druhá proměnná vtCurStatus je proměnná ve zdrojovém procesu.

4. Následující příklad postupně čte prvky array z procesu definovaného parametrem $vlProces:

```
GET PROCESS VARIABLE($vlProces;vl_IPCom_Array;$vlVelikost)
For($vlElem;1;$vlVelikost)
   GET PROCESS VARIABLE($vlProces;at_IPCom_Array{$vlElem};$vtElem)
      ` Udělat něco s $vtElem
End for
```

Poznámka: V tomto příkladu obsahuje proměnná vl_IPCom_Array velikost array vt_IPCom_Array a musí být uložena ve zdrojovém procesu.

5. Tento příklad provede to samé jako předchozí, ale přečte array jako celek, místo načítání jednotlivých prvků:

```
GET PROCESS VARIABLE ($vlProcess;at_IPCom_Array;$anArray)
For($vlElem;1;Size of array($anArray))
      ` Udělat něco s $anArray{$vlElem}
End for
```

6. Tento příklad přečte zdrojové proměnné v1, v2 a v3 a uloží jejich hodnoty do proměnných stejného názvu v platném procesu:

```
GET PROCESS VARIABLE ($vlProces;v1;v1;v2;v2;v3;v3)
```

7. Podívejte se na příklad u příkazu DRAG AND DROP PROPERTIES.

Dále si přečtěte

CALL PROCESS, Táhnout a vsadit, DRAG AND DROP PROPERTIES, Procesy, SET PROCESS VARIABLE, VARIABLE TO VARIABLE.

Příkazy a odkazy pro Procesy (komunikace)

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET PROCESS VARIABLE (NASTAVIT PROMĚNNOU DO PROCESU)

Příkazy a odkazy pro Procesy (komunikace)

Verze 6.0

SET PROCESS VARIABLE (proces; cílProm; výraz{; cílProm2; výraz2; ...; cílPromN; výrazN})

Parametr	Typ		Popis
proces	Číslo	→	Číslo cílového procesu
cílProm	Proměnná	→	Cílová proměnná k nastavení
výraz	Proměnná	→	Zdrojový výraz (nebo proměnná) přenášející hodnotu

Popis

Příkaz SET PROCESS VARIABLE zapíše do cílové proměnné cílProm (cílProm2, atd) v cílovém procesu, jehož číslo předáte do parametru proces hodnotu obsaženou ve výraz (výraz2, atd.).

Každá cílová proměnná může být proměnná nebo prvek array. Přečtěte si dále seznam omezení.

Pro každou dvojici cílProm;výraz musí platit, že jsou stejného typu, jinak může být přenášená hodnota pozměněna nebo nemusí být přenesena. Pokud, v nezkompilované verzi, neexistuje cílová proměnná, je vytvořena a je do ní dosazena hodnota výrazu.

Platný proces vloží hodnotu do cílové proměnné - cílový proces není upozorněn, že jiný proces zapisuje do jeho proměnné.

4D Server: Při používání 4D Client, můžete zapsat proměnnou do cílového procesu spuštěného na počítači serveru (uložená procedura). Pro tento účel vložte znaménko mínus před číslo procesu v parametru proces.

TIP: Pokud nevíte číslo procesu na serveru, můžete stále použít meziprocesní proměnnou serveru. K tomu použijte jakékoli záporné číslo v čísle procesu. Jinými slovy není nutné znát číslo procesu k použití meziprocesní proměnné serveru v příkazu Set process variable. Je to použitelné pokud je uložená procedura spuštěna s použitím metody databáze Při spuštění serveru. Počítač klienta neví číslo tohoto procesu, může být použita jakákoli záporná hodota v parametru proces.

Omezení

SET PROCESS VARIABLE nepřijímá místní proměnnou jako cílovou proměnnou.

SET PROCESS VARIABLE přijímá jakýkoli typ meziprocesních a procesních proměnných, mimo:

- Ukazatelů
- Array jakéhokoli typu. K zapsání array jako celku z jednoho procesu do druhého, použijte příkaz VARIABLE TO VARIABLE. Nicméně nezapomeňte,že SET PROCESS VARIABLE přijímá jednotlivé prvky array.
- Nemůžete zapisovat prvky array ukazatelů nebo dvojrozměrné array.

Cílový proces musí být proces uživatele: nemůže to být jeden ze základních procesů. Pokud cílový proces neexistuje, je generována chyba. Můžete tuto chybu zachytit pomocí příkazu ON ERR CALL.

Příklady

1. Tento řádek kódu nastaví (na prázdný řetězec) textovou proměnnou vtCurStatus v procesu jehož číslo je v $vlProces:

```
SET PROCESS VARIABLE ($vlProces;vtCurStatus;"")
```

2. Tento řádek kódu nastaví textovou proměnnou vtCurStatus která je v procesu s číslem $vlProces na hodnotu proměnné $vtInfo prováděné metody v platném procesu:

SET PROCESS VARIABLE ($vlProces;vtCurStatus;$ctInfo)

3. Tento řádek kódu nastaví proměnnou vtCurStatus která je v procesu s číslem $vlProces na hodnotu stejné proměnné v platném procesu:

SET PROCESS VARIABLE ($vlProces;vtCurStatus; vtCurStatus)

Poznámka: První proměnná vtCurStatus je proměnná v cílovém procesu a druhá proměnná vtCurStatus je v platném procesu.

4. Tento příklad postupně nastaví na velká písmena všechny prvky procesní array z procesu $vlProces

```
GET PROCESS VARIABLE($vlProcess;vl_IPCom_Array;$vlVelikost)
For($vlElem;1;$vlVelikost)
   GET PROCESS VARIABLE($vlProcess;at_IPCom_Array{$vlElem};$vtElem)
   SET PROCESS VARIABLE($vlProcess;at_IPCom_Array{$vlElem}; Uppercase($vtElem))
End for
```

Poznámka: V tomto příkladu obsahuje proměnná vl_IPCom_Array velikost array vt_IPCom_Array a musí být uložena ve zdrojovém/cílovém procesu.

5. Tento příklad zapíše proměnné v1, v2 a v3 v cílovém procesu pomocí proměnných se stejným názvem v platném procesu:

GET PROCESS VARIABLE ($vlProces;v1;v1;v2;v2;v3;v3)

Dále si přečtěte

CALL PROCESS, GET PROCESS VARIABLE, Procesy, VARIABLE TO VARIABLE.

Příkazy a odkazy pro Procesy (komunikace)

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

VARIABLE TO VARIABLE (PROMĚNNÁ DO PROMĚNNÉ)

Příkazy a odkazy pro Procesy (komunikace)

Verze 6.0.2

VARIABLE TO VARIABLE (proces; cílProm; zdrProm{; cílProm2; zdrProm2; ...; cílPromN; zdrPromN})

Parametr	Typ		Popis
proces	Číslo	→	Číslo cílového procesu
cílProm	Proměnná	→	Cílová proměnná
zdrProm	Proměnná	→	Zdrojová proměnná

Popis

Příkaz VARIABLE TO VARIABLE zapíše do proměnné cílProm která je v cílovém procesu s číslem zadaným v parametru proces hodnotu z proměnné zdrProm.

VARIABLE TO VARIABLE má stejnou funkci jako SET PROCESS VARIABLE s následujícími rozdíly:

• Do SET PROCESS VARIABLE předáte zdrojový výraz a proto nemůžete vložit array jako celek. Do VARIABLE TO VARIABLE musíte vložit proměnnou a proto můžete vložit i array jako celek.
• Každá cílová proměnná v příkazu SET PROCESS VARIABLE může být proměnná nebo prvek array, ale nemůže být array jako celek. Každá cílová proměnná v příkazu VARIABLE TO VARIABLE může být proměnná, array nebo prvek array.

Pro každou dvojici cílProm;zdrProm musí platit, že jsou stejného typu, jinak nemusí být obsah správně přenesen. Pokud v nezkompilované verzi, neexistuje cílová proměnná, je vytvořena a je do ní dosazena hodnota zdrojové proměnné.

Platný proces vloží hodnotu do cílové proměnné - cílový proces není upozorněn, že jiný proces zapisuje do jeho proměnné.

Omezení

VARIABLE TO VARIABLE nepřijímá místní proměnnou jako cílovou proměnnou.

VARIABLE TO VARIABLE přijímá jakýkoli typ meziprocesních a procesních proměnných, mimo:

• Ukazatele
• Array ukazatelů
• Dvojroměrné array

Cílový proces musí být proces uživatele: nemůže to být jeden ze základních procesů. Pokud cílový proces neexistuje, je generována chyba. Můžete tuto chybu zachytit pomocí příkazu ON ERR CALL.

Příklad

Následující příklad přečte procesní array z procesu s číslem $vlProces a postupně nastaví všechny položky na velká písmena a pak pošle array zpět jako celek:

```
GET PROCESS VARIABLE($vlProces;at_IPCom_Array;$anArray)
For($vlElem;1;Size of array($anArray))
   $anArray{$vlElem}:=Uppercase($anArray{$vlElem})
End for
VARIABLE TO VARIABLE($vlProcess;at_IPCom_Array;$anArray)
```

Dále si přečtěte

GET PROCESS VARIABLE, SET PROCESS VARIABLE.

Příkazy a odkazy pro Procesy (komunikace)

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

HIDE PROCESS (SKRÝT PROCES)

Příkazy a odkazy pro Procesy (rozhraní)

Verze 3

HIDE PROCESS (proces)

Parametr	Typ		Popis
proces	Číslo	→	Číslo procesu který se má skrýt

Popis

Příkaz HIDE PROCESS skryje všechna okna která k procesu náleží. Všechny prvky rozhraní procesu jsou skryté, dokud není zavolán příkaz SHOW PROCESS. Skrytá jsou též záhlaví nabídek procesu. To znamená, že otevření okna ve chvíli kdy je proces skrytý, neprovede překreslení nebo obnovu obrazovky. Pokud je proces skrytý, nebudou mít příkazy žádný efekt.

Jedinou výjimkou je okno Ladění. Pokud je okno ladění zobrazeno pro proces, který je skrytý, přesune se proces na popředí a zobrazí se.

Pokud nechcete proces zobrazit při jeho vytvoření, bude příkaz HIDE PROCESS prvním příkazem metody procesu. Tímto příkazem nemohou být skryty procesy Uživatele/Aplikace a Cache manažer.

Ačkoli byl proces skrytý, stále je v funkční.

Příklad

Následující příklad skryje okna platného procesu:

```
HIDE PROCESS(Current process)
```

Dále si přečtěte

Process state, SHOW PROCESS.

Příkazy a odkazy pro Procesy (rozhraní)

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SHOW PROCESS (UKÁZAT PROCES)

Příkazy a odkazy pro Procesy (rozhraní)

Verze 3

SHOW PROCESS (proces)

Parametr	Typ		Popis
proces	Číslo	→	Číslo procesu který se má ukázat

Popis

Příkaz SHOW PROCESS zobrazí všechna okna patřící k procesu. Tento příkaz nepřenese proces na popředí. K tomu musíte použít příkaz BRING TO FRONT.

Pokud je proces již zobrazený, příkaz neprovede nic.

Příklad

Následující příklad zobrazí proces s názvem Zákazníci, pokud byl před tím skrytý. Číslo odkazu na proces je uloženo v meziprocesní proměnné <>Zákazníci:

```
SHOW PROCESS (<>Zákazníci)
```

Dále si přečtěte

BRING TO FRONT, HIDE PROCESS, Process state.

Příkazy a odkazy pro Procesy (rozhraní)

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

BRING TO FRONT (PŘENÉST NA POPŘEDÍ)

Příkazy a odkazy pro Procesy (rozhraní)

Verze 3

BRING TO FRONT (proces)

Parametr	Typ		Popis
proces	Číslo	→	Číslo procesu, který se má přenést na popředí

Popis

Příkaz BRING TO FRONT přenese všechna okna procesu na popředí. Pořadí oken je zachováno. Pokud je proces již na popředí, příkaz neprovede nic. Pokud je proces skrytý, musíte nejdříve provést SHOW PROCESS k zobrazení procesu a pak teprve BRING TO FRONT.

Procesy Uživatele/Aplikace a Návrháře mohou být přeneseny na popředí pomocí tohoto příkazu.

Příklad

Následující příklad může být použit jako metoda volaná z nabídky. Nejprve otestuje jestli je proces na popředí a pokud není, tak jej přenese na popředí.

```
If (Frontmost process#<>vlAddCust_PID)
   BRING TO FRONT (<>vlAddCust_PID)
End if
```

Dále si přečtěte

HIDE PROCESS, Process state, SHOW PROCESS.

Příkazy a odkazy pro Procesy (rozhraní)

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Frontmost process (Proces na popředí)

Příkazy a odkazy pro Procesy (rozhraní)

Verze 3

Frontmost process {(*)} → Integer

Parametr	Typ		Popis
*		→	Číslo procesu pro první neplovoucí okno
Výsledek funkce	Integer	→	Číslo procesu, jehož okno je na popředí

Popis

Frontmost process vrací číslo procesu, jehož okno (okna) je na popředí.

Pokud máte otevřeno jedno nebo více plovoucích oken, jsou dvě verstvy oken:

• Normální okna

• Plovoucí okna

Pokud je příkaz Frontmost process spuštěn z metody formuláře nebo objektu v plovoucím okně, je vráceno číslo procesu na popředí plovoucího okna. Pokud předáte parametr *, jsou přeskočena plovoucí okna je vráceno číslo procesu na popředí pro normální okna.

Příklad

Podívejte se na příkad u příkazu BRING TO FRONT.

Dále si přečtěte

BRING TO FRONT, WINDOW LIST.

Příkazy a odkazy pro Procesy (rozhraní)

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Procesy

Příkazy a odkazy pro Procesy

Verze 6.0

Multi-tasking ve 4th Dimension je možnost, která vám umožní provádět najednou více operací databáze. Tyto operace se nazývají procesy.

Více procesů je skoro to samé jako více použití jednoho počítače, každý pracuje podle vlastních kritérií. To znamená, že každá metoda může být spuštěna v oddělené části databáze.

Tato část popisuje následující:

- Vytváření a mazání procesů
- Prvky procesu
- Procesy uživatele
- Procesy vytvořené 4th Dimension
- Místní a globální procesy
- Zamykání záznamů mezi procesy

Poznámka: Tato část nepopisuje uložené procedury. Přečtěte si část Uložené procedury (Stored procedures) v *Příručce 4D Server*.

Vytváření a mazání procesů

Jsou tři způsoby vytvoření procesu:

- Spuštění metody v prostředí uživatele po zaškrtnutí políčka **Nový proces** po vybrání položky **Provést metodu**. Metoda provedená z okna Provést metodu je pak metodou procesu.
- Proces může být spuštěn vybráním položky nabídky. V **Editoru nabídek** musíte vybrat položku nabídky a označit u ní vlastnost **Začít nový proces**. Metoda přiřazená k položce nabídky bude metoda procesu.
- Použít funkci New process. Metoda předáná jako parametr do tohoto příkazu bude metoda procesu.

Proces bude ukončen po splnění jedné z následujících podmínek. První dvě jsou automatické:

- Pokud metoda procesu skončí.
- Pokud uživatel vypne databázi.
- Pokud proces zastavíte metodou nebo stisknete tlačítko Odstranit v ladění.
- Pokud vyberete položku **Odstranit** z nabídky **Proces** v prostředí návrháře.

Proces může vytvořit jiný proces. Procesy nejsou řazeny hierarchicky - všechny procesy jsou rovnocené, včetně procesu ze kterého byli vytvořeny. Jakmile jeden proces vytvoří jiný, stává se nový proces nezávislý na procesu ze kterého byl vytvořen a funguje i po skončení "rodičovského" procesu.

Prvky procesu

Každý proces obsahuje specifické prvky. Existují tří typy prvků, které jsou od sebe odlišné:

- **Prvky rozhraní**: Prvky které jsou nutné pro zobrazení procesu.
- **Datové prvky**: Informace které jsou spojené s daty v procesu.
- **Prvky jazyka**: Prvky které jsou použity procedurálně nebo jsou nutné pro vývoj vaší aplikace.

Prvky rozhraní

Prvky rozhraní obsahují následující:

- **Záhlaví nabídek**: Každý proces může mít svoje vlastní platné záhlaví nabídek. Platná nabídky procesu na

popředí je platná nabídka databáze.
• **Jedno nebo více oken**: Každý proces může mít najednou otevřeno více než jedno okno. Na druhou stránku jsou procesy,které nemusí mít otevřené žádné okno.
• **Aktivní (na popředí) okno**: Ačkoli může mít proces otevřeno více oken, pouze jedno okno je aktivní. Aby jste měli více aktivních oken, musíte mít otevřeno více procesů.

Datové prvky
Prvky dat se odkazují na data použitá databází. Jsou to:
• **Platný výběr pro tabulku**: Každý proces má svůj platný výběr záznamů. Jedna tabulka může mít jeden výběr pro jeden proces.
• **Platný záznam pro tabulku**: Každá tabulka může mít jiný platný záznam pro každý proces.
Poznámka: Tento popis datových prvků je platný pokud vaše procesy jsou globální v rozsahu. Jako výchozí je globální každý proces. Přečtěte si část "Globální a místní procesy".

Prvky jazyka
Prvky jazyka pro proces jsou prvky spojené s programováním ve 4th Dimension.
• **Proměnné**: Každý proces má svoje vlastní proměnné procesu. Pro více informací si přečtěte část Proměnné. Proměnné procesu jsou poznatelné pouze v procesu ve kterém byli vytvořeny.
• **Výchozí tabulka**: Každý proces má svojí výchozí tabulku. Nicméně si povšimněte že příkaz DEFAULT TABLE je pouze konvence zápisu pro programování.
• **Vstupní a výstupní formulář**: Výchozí vstupní a výstupní formuláře mohou být nastaveny pro každou tabulku a každý proces.
• **Sady procesu**: Každý proces má svoje sady. UserSet a LockedSet jsou sady procesu. Sady procesu jsou vymazány jakmile je metoda procesu ukončena.
• **Zachytávání chyb pro proces:** Každý proces může mít svojí metodu pro zachytávání chyb.
• **Okno ladění**: Každý proces může mít svoje vlastní okno ladění.

Procesy uživatele

Procesy uživatele vytváříte pro provedení různých akcí. Je jim počítán stejný čas procesu jako základním procesům. Jako příklad jsou procesy Web ve spojení s procesy uživatele.

Procesy vytvořené 4th Dimension

Následující procesy jsou vytvořeny a řízeny 4th Dimension:
• **Proces uživatel/aplikace**: Proces uživatel/aplikace se skládá z prostředí uživatele a Vlastní nabídky. Výchozí Úvodní obrazovka v prostředí Vlastních nabídek je také součást procesu uživatel/aplikace. Tento proces je vytvořen ihned po zapnutí 4th Dimension.
• **Proces návrháře**: Tento proces obsahuje pouze prostředí návrháře, které běží jako samostatný proces. Může být uzavřen pomocí položky nabídky **Opustit prostředí návrháře** z nabídky **Soubor** v prostředí navrháře. Ve zkompilované databázi neexistuje proces návrháře. Tento proces je vytvořen když uživatel poprvé vstoupí do prostředí návrháře. Pokud se databáze otevírá do prostředí Vlastních nabídek nebo uživatele, není tento proces ihned vytvořen.
• **Proces Web server**: Tento proces je nastartován pokud je aplikace publikována na Webu. Jestli chcete vědět více informací, přečtěte si část Web služby a Procesy Web spojení.
• **Proces Cache manažer**: Proces Cache manažer řídí disk I/ O pro databázi. Tento proces je vytvořen ihned po zapnutí 4th Dimension nebo 4D Server.
• **Proces indexování**: Tento proces řídí indexování polí jako oddělenou akci. Je vytvořen pokud je nějaké pole indexované nebo jsou indexy mazány.
• **Proces Event manažer**: Tento proces je vytvořen když je nainstalována metoda pro zachytávání událostí pomocí

příkazu ON EVENT CALL. Provede metodu instalovanou příkazem ON EVENT CALL kdykoli je vytvořena nějaká událost. Tato metoda je metoda procesu pro tento proces. Tento proces je prováděn neustále, dokonce i když neběží žádná metoda. Ovládání událostí je také prováděno v prostředí návrháře.

Místní a globální procesy

Proces může být ve svém rozsahu místní nebo globální. Jako výchozí jsou všechny procesy globální.

Globální proces může provádět různé akce včetně přístupu a práce s daty. Ve většině případů budete používat globální procesy.

Místní procesy mohou být použity pouze pro operace bez přístupu k datům. Například můžete použít místní proces k spuštění metody pro ovládání událostí nebo pro kontrolu prvků rozhraní jako jsou plovoucí okna.

Jestli bude proces místní určujete jeho názvem. Název procesu musí začínat znakem dolaru ($).

Upozornění: Pokud budete z místního procesu zpřístupňovat data, budete to dělat přes proces uživatele/aplikace, čímž budete riskovat konflikty s operacemi probíhajícími v tomto procesu.

4D Server: Používání místních procesů na klientovi pro akce které nevyžadují přístup k datům ušetří mnoho času procesům serveru.

Zamykání záznamů mezi procesy

Záznam je uzamčen, pokud nějaký jiný proces má tento záznam otevřený pro upravování dat. Zamčený záznam může být načten jiným procesem, ale nemůže být změněn. Může být odemčen pouze z procesu, který jej upravuje. Pro zamykání záznamů, musí být tabulka v módu číst/psát.

Dále si přečtěte

Metody, Metody projektu, Proměnné.

Příkazy a odkazy pro Procesy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

New process (Nový proces)

Příkazy a odkazy pro Procesy

Verze 6.0 (Upraveno)

New process (metoda; stack{; název{; param{; param2; ...; paramN}{; *}}}) → Číslo

Parametr	Typ		Popis
metoda	Řetězec	→	Metoda která bude spuštěná v procesu
stack	Číslo	→	Velikost paměti stack pro proces v bytech
název	Řetězec	→	Název vytvořeného procesu
param	Výraz	→	Parametry, jejichž hodnota bude předána do procesu
*		→	Proces bude jedinečný
Výsledek funkce	Číslo	←	Číslo odkazu na proces

Popis

Příkaz New proces zapne nový proces (na stejném stroji) a vrátí číslo odkazu na tento proces.

Pokud nový proces nemůže být vytvořen (například pro nedostatek paměti) vrátí příkaz nulu (0) a je generována chyba. Tuto chybu můžete zachytit pomocí příkazu ON ERR CALL.

Metoda procesu: Do parametru metoda, předáte název metody procesu pro nový proces. Po té co 4D nastaví obsah procesu, spustí tuto metodu, která se stane metodou procesu.

Stack procesu: Do parametru stack předáte paměť která bude vyhrazena pro tento proces. Tato paměť je použita pro volání metody, místní proměnné, parametry v podprogramech a načtené záznamy. Toto číslo je vyjádřeno v bytech: obvykle nebudete vkládat více než 32K (kolem 32000 bytů), ale můžete samozřejmě vložit i více pro procesy, ve kterých budete provádět složité řetězové volání (podprogramy volají podprogramy kaskádovitě). Můžete například vložit 200K (kolem 200000 bytů).

Poznámka: Stack NENÍ celková paměť procesu. Procesy obsahují paměť pro záznamy, meziprocesní proměnné, atd. Také používá extra paměť pro ukádání vlastních proměnných procesu. Stack obsahuje různé informace 4D: počet informací předáných do Stack záleží na počtu do sebe zapadajících metod které proces volá, počtu formulářů které otevře před tím než je uzavře a na počtu a velikosti místních proměnných použitých v každé metodě která je spuštěna z procesu.

Název procesu: Do parametru název předáte název nového procesu. Tento název se objeví v **Seznamu procesů** v prostředí návrháře a bude vraceno příkazem PROCESS PROPERTIES. Název procesu může být až 31 znaků dlouhý. Tento parametr můžete vynechat. Pokud to uděláte, bude název procesu prázdný řetězec. Pokud tento název bude začínat znakem dolaru ($), bude tento proces vytvořen jako místní proces.

Důležité: Nezapomeňte, že ve struktuře klient/server nemá místní proces přístup k datům.

Parametry do metody procesu: Od verze 6 můžete do metody procesu vložit parametry. Vkládat je můžete stejným způsobem jako do podprogramu. Nicméně jsou zde rozdíly - nemůžete jako parametr vložit ukazatel. Nezapomeňte, že také array nemohou být předány jako parametr do metody. Jakmile je metoda procesu spuštěna, můžete tyto parametry získat z proměnných $1, $2, atd.

Poznámka: Jestliže předáte parametr do metody procesu, musíte vložit parametr název: v tomto případě již nemůže být vynechán.

Volitelný parametr *: Přidáním tohoto posledního parametru, řeknete 4D aby zkontrolovala, jestli již neběží proces se stejným názvem, jaký jste zadali do parametru název. Pokud takový proces již existuje, 4D nezaloží nový proces, ale vrátí číslo tohoto procesu.

Příklad

Mějme následující metodu:

```
   ` PŘIDAT ZÁKAZNÍKA
MENU BAR (1)
Repeat
   ADD RECORD ([Zákazníci];*)
Until (OK=0)
```

Pokud tuto metodu přiřadíte k položce nabídky a označíte u ní vlastnost **Začít nový proces**, 4D automaticky začne nový proces pro tuto metodu. Příkaz MENU BAR (1) přiřadí toto záhlaví k novému procesu. Protože chybí otevření okna (pomocí Open window), příkaz ADD RECORD si jej automaticky vytvoří.

Pokud chcete spustit nový proces pro tuto metodu z tlačítka, můžete napsat následující:

```
   ` Metoda objektu bPřidatZák - tlačítko
$vlIDProces:=New process("PŘIDAT ZÁKAZNÍKA";32*1024;"Přidt zákazníka")
```

Tlačítko provede stejnou akci jako položka nabídky.

Pokud chcete při vybrání položky nabídky nebo klepnutí na tlačítko začít nový proces (pokud již neexistuje) nebo jej přenést na popředí (pokud již běží), můžete napsat následující metodu PŘIDÁNÍ ZÁKAZNÍKŮ:

```
   ` PŘIDÁNÍ ZÁKAZNÍKŮ
$vlIDProces:=New process("PŘIDAT ZÁKAZNÍKA";32*1024;"Přidt zákazníka";*)
If ($vlIDProces#0)
   BRING TO FRONT ($vlIDProces)
End if
```

Metoda tlačítka pro přidání zákazníka bude vypadat následovně:

```
   ` Metoda objektu bPřidatZák - tlačítko
PŘIDÁNÍ ZÁKAZNÍKŮ
```

V editoru nabídek nahraďte metodu PŘIDAT ZÁKAZNÍKA za PŘIDÁNÍ ZÁKAZNÍKŮ a odznačte možnost **Začít nový proces**.

Dále si přečtěte

Execute on server, Metody, Procesy, Metody projektu, Proměnné.

Příkazy a odkazy pro Procesy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Execute on server (Provést na serveru)

Příkazy a odkazy pro Procesy

Verze 6.0

Execute on server (metoda; stack{; název{; param{; param2; ...; paramN}{; *}}}) → Číslo

Parametr	Typ		Popis
metoda	Řetězec	→	Uložená procedura, která bude v procesu na serveru prováděna
stack	Číslo	→	Velikost paměti stack pro proces v bytech
název	Řetězec	→	Název vytvořeného procesu uložené procedury
param	Výraz	→	Parametry, jejichž hodnota bude předána procesu
*		→	Proces bude jedinečný
Výsledek funkce	Číslo	←	Číslo odkazu na proces

Popis

Příkaz Execute on server zapne nový proces na počítači serveru (pokud je proveden v architektuře klient/server) nebo na stejném počítači (pokud je volán v jednouživatelské verzi) a vrátí číslo procesu.

Tento příkaz se používá k zapnutí uložených procedur. Jestli chcete vědět více informací o uložených procedurách, přečtěte si část Uložené procedury (Stored procedures) v *Příručce 4D Server*.

Pokud provedete příkaz Execute on server na počítači klienta, příkaz vrátí záporné číslo procesu. Pokud tento příkaz provedete na počítači serveru, příkaz vrátí kladné číslo procesu. Všimněte si, že příkaz New process provedený na serveru, provede to samé jako Execute on server.

Pokud nový proces nemůže být vytvořen (například pro nedostatek paměti) vrátí příkaz nula (0) a je generována chyba. Tuto chybu můžete zachytit pomocí příkazu ON ERR CALL.

Metoda procesu: Do parametru metoda, předáte název metody procesu pro nový proces. Po té co 4D nastaví obsah procesu, spustí tuto metodu, která se stane metodou procesu.

Stack procesu: Do parametru stack předáte paměť která bude vyhrazena pro tento proces. Tato paměť je použita pro volání metody, místní proměnné, parametry v podprogramech a načtené záznamy. Toto číslo je vyjádřeno v bytech: obvykle nebudete vkládat více než 32K (kolem 32000 bytů), ale můžete samozřejmě vložit i více pro procesy, ve kterých budete provádět složité řetězové volání (podprogramy volají podprogramy kaskádovitě). Můžete například vložit 200K (kolem 200000 bytů).

Poznámka: Stack NENÍ celková paměť procesu. Procesy obsahují paměť pro záznamy, meziprocesní proměnné, atd. Také používá extra paměť pro ukádání vlastních proměnných procesu. Stack obsahuje různé informace 4D: počet informací předáných do Stack záleží na počtu do sebe zapadajících metod které proces volá, počtu formulářů které otevře před tím než je uzavře a na počtu a velikosti místních proměnných použitých v každé metodě která je spuštěna z procesu.

Název procesu: Do parametru název předáte název nového procesu. Na jednouživatelské verzi se tento název objeví v **Seznamu procesů** v prostředí návrháře a bude vraceno příkazem PROCESS PROPERTIES. Ve víceuživatelské verzi se objeví modře v seznamu Uložených procedur.

Název procesu může být až 31 znaků dlouhý. Tento parametr můžete vynechat. Pokud to uděláte, bude název procesu prázdný řetězec.

UPOZORNĚNÍ: Na rozdíl od příkazu New process, se nepokoušejte vytvářet místní proces tím, že jeho název začne znakem dolaru ($) pokud proces spoušíte pomocí Execute on server. Bude to pracovat na jednouživatelské verzi, protože Execute on server v tomto prostředí pracuje jako New process. Na druhou stránku v architektuře

klient/server vám vznikne chyba.

Parametry do metody procesu: Od verze 6 můžete do metody procesu vložit parametry. Vkládat je můžete stejným způsobem jako do podprogramu. Nicméně jsou zde rozdíly - nemůžete jako parametr vložit ukazatel. Nezapomeňte, že také array nemohou být předány jako parametr do metody. Jakmile je metoda procesu spuštěna, můžete tyto parametry získat z proměnných $1, $2, atd.

Poznámka: Jestliže předáte parametr do metody procesu, musíte vložit parametr název: v tomto případě již nemůže být vynechán.

Volitelný parametr *: Přidáním tohoto posledního parametru, řeknete 4D aby zkontrolovala, jestli již neběží proces se stejným názvem, jaký jste zadali do parametru název. Pokud takový proces již existuje, 4D nezaloží nový proces, ale vrátí číslo tohoto procesu.

Příklad

1. Následující příklad ukazuje jak může být urychlen import v prostředí klient/server. Metoda Normální import, ukázaná zde, vám ukáže jak dlouho trvá import na počítači klienta s použitím příkazu IMPORT TEXT:

```
   ` Metoda projektu Normální import
$vhDokumRef:=Open document("")
If (OK=1)
   CLOSE DOCUMENT($vhDokumRef)
   INPUT FORM([Tabulka1];"Import")
   $vhStartTime:=Current time
   IMPORT TEXT([Tabulka1];Document)
   $vhEndTime:=Current time
   ALERT("Trvolo to "+String(0+($vhEndTime-$vhStartTime))+" sekund.")
End if
```

Při normálním importu dat, 4D Client provede rozbor textového souboru, pak pro každý záznam vytvoří nový záznam, naplní jej informacemi z textového souboru a pošle záznam na server aby mohl být předán do databáze. Příliš mnoho datazů jde přes síť. Způsob k optimalizaci této operace je použít uloženou proceduru k provedení této akce přímo na serveru. Na klientovi se načte textový soubot do BLOBu a zapne uloženou proceduru s tímto BLOBem jako parametrem. Uložená procedura uloží tento blob jako dokument na disk serveru a pak naimportuje data. Import dat je pak proveden místně jako na jednouživatelské verzi - rychlostí, protože jsou vynechány složité úkoly přes síť. Zde je metoda KLIENT IMPORT, která probíhá na 4D Client a zapne uloženou proceduru SERVER IMPORT, která je zobrazena níže:

```
   ` Metoda projektu KLIENT IMPORT
   ` KLIENT IMPORT ( Ukazatel ; Řetězec )
   ` KLIENT IMPORT ( → [Tabulka] ; Vstupní formulář )
C_POINTER($1)
C_STRING(31;$2)
C_TIME($vhDokumRef)
C_BLOB($vxData)
C_LONGINT(spErrCode)
   ` Označit dokument k importu
$vhDokumRef:=Open document("")
If (OK=1)
      ` Pokud byl dokument označen, již jej nepotřebujeme otevřený
   CLOSE DOCUMENT($vhDokumRef)
   $vhStartTime:=Current time
      ` Zkusit jej načíst do paměti
   DOCUMENT TO BLOB(Document;$vxData)
```

```
    If (OK=1)
        ` Pokud může být dokument načten do BLOBu,
        ` Začít uloženou proceduru, která naimportuje data na serveru
      $spProcessID:=Execute on server("SERVER IMPORT";32*1024;
                    "Import záznamů";Table($1);$2;$vxData)
        ` V tomto procesu již nepotřebujeme BLOB
      CLEAR VARIABLE($vxData)
        ` Počkat dokud uložená procedura nedokončí import
      Repeat
        DELAY PROCESS( Current process;300)
        GET PROCESS VARIABLE($spProcessID;spErrCode;spErrCode)
        If (Undefined(spErrCode))
            ` Poznámka: Pokud uložená procedura nevytvořila vlastní proměnnou
            ` spErrCode, můžeme dostat nedefinovanou proměnnou
          spErrCode:=1
        End if
      Until (spErrCode<=0)
        ` Říct uložené proceduře, že jsme zareagovali
      spErrCode:=1
      SET PROCESS VARIABLE($spProcessID;spErrCode;spErrCode)
      $vhEndTime:=Current time
      ALERT("Trvalo to "+String(0+($vhEndTime-$vhStartTime))+" sekund.")
    Else
      ALERT("Není dostatek paměti k načtení dokumentu.")
    End if
  End if
```

Zde je metoda SERVER IMPORT, která je spuštěna jako uložená procedura:

```
    ` Metoda projektu SERVER IMPORT
    ` SERVER IMPORT ( Long ; Řetězec ; BLOB )
    ` SERVER IMPORT ( Číslo tabulky ; Vstupní formulář ; Import dat )
  C_LONGINT($1)
  C_STRING(31;$2)
  C_BLOB($3)
  C_LONGINT(spErrCode)
    ` Operace ještě není dokončena, nastavit spErrCode na 1
  spErrCode:=1
  $vpTable:=Table($1)
  INPUT FORM($vpTable→;$2)
  $vsNazev:="Import File "+String(1+Random)
  DELETE DOCUMENT($vsNazev)
  BLOB TO DOCUMENT($vsNazev;$3)
  IMPORT TEXT($vpTable→;$vsNazev)
  DELETE DOCUMENT($vsNazev)
    ` Akce je dokončena, nastavit spErrCode na 0
  spErrCode:=0
    ` Počkat dokud klient nepošle odpověď
  Repeat
    DELAY PROCESS( Current process;1)
  Until (spErrCode>0)
```

Jakmile jsou tyto dvě metody zapsány do databáze, můžete je použít následovně:

KLIENT IMPORT (→[Tabulka];"Import")

S malými rozdíly zjistíte, že s použitím tohoto způsobu, můžete provádět import až 60x rychleji než u normálního importu.

Dále si přečtěte

New process, Uložené procedury.

Příkazy a odkazy pro Procesy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DELAY PROCESS (ODLOŽIT PROCES)

Příkazy a odkazy pro Procesy

Verze 3

DELAY PROCESS (proces; trvání)

Parametr	Typ		Popis
proces	Číslo	→	Číslo procesu
trvání	Číslo	→	Trvání vyjádřené v ticích

Popis

Příkaz DELAY PROCESS odloží provádění procesu o počet tiků (60 tiků = 1 sekunda). Během tohoto času nepotřebuje proces žádný čas procesu. I když je proces odložen, je stále v paměti.

Pokud je proces ještě odložen, příkaz jej odloží znovu. Parametr trvání se nenastaví na zbývající čas, ale nahradí se novou hodnotou. Pokud chcete odložení přerušit, vložte 0 (nula) do parametru trvání.

Pokud proces neexistuje, příkaz neprovede nic.

Upozornění: DELAY PROCESS nemá žádný vliv na základnmí procesy (všechna prostředí) včetně procesu uživatele/aplikace.

Tip: K odložení procesu uživatele/aplikace, napište malou metodu, která bude obsahovat smyčku pro měření času pomocí Current time, nebo Tickount nebo Milliseconds. Například pokud chcete na určitý čas zobrazit zprávu v okně které pro tento účel otevřete a opět zavřete.

Příklady

1. Podíveje se na příklad u Zamykání záznamů.

2. Podívejte se na příklad u příkazu Process number.

Dále si přečtěte

HIDE PROCESS, PAUSE PROCESS.

Příkazy a odkazy pro Procesy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

PAUSE PROCESS (POZASTAVIT PROCES)

Příkazy a odkazy pro Procesy

Verze 3

PAUSE PROCESS (proces)

Parametr	Typ		Popis
proces	Číslo	→	Číslo procesu

Popis

PAUSE PROCESS přeruší provádění procesu do té doby, dokud není zavolán příkazem RESUME PROCESS.Během tohoto času nezabírá proces žádný čas na počítači. I když je proces přerušen, je stále v paměti.

Pokud je již proces přerušený, PAUSE PROCESS neprovede nic. Pokud je proces odložen příkazem DELAY PROCESS, je proces přerušen. RESUME PROCESS navrátí proces ihned.

Ve chvíli kde je proces přerušen, nejsou okna procesu dostupná. V tomto případě je dobré proces před uživatelem skrýt. Pokud proces neexistuje, příkaz neprovede nic.

Upozornění: Použijte PAUSE PROCESS pouze na procesy, které jste zapnuli. Tento příkaz nefunguje na originální proces uživatele/aplikace.

Dále si přečtěte

DELAY PROCESS, HIDE PROCESS, RESUME PROCESS.

Příkazy a odkazy pro Procesy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

RESUME PROCESS (NAVRÁTIT PROCES)

Verze 3

RESUME PROCESS (proces)

Parametr	Typ		Popis
proces	Číslo	→	Číslo procesu

Popis

RESUME PROCES navrátí proces jehož průběh byl pozastaven nebo odložen. Pokud proces není ani pozastaven ani odložen,příkaz neprovede nic.

Pokud byl proces odložen, podívejte se na příkaz PAUSE PROCESS nebo DELAY PROCESS. Pokud proces neexistuje, příkaz neprovede nic.

Dále si přečtěte

DELAY PROCESS, PAUSE PROCESS.

Příkazy a odkazy pro Procesy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Process aborted (Proces odstraněn)

Příkazy a odkazy pro Procesy

Verze 6.5

Process aborted → Logické

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Logické	←	True = chystá se odstranit, False = proces se nechystá odstranit

Popis

Příkaz Process aborted vrací True pokud proces, ze kterého je volán, se chystá skončit, což znamená že proces bude skončen jiným, než normálním způsobem. To se může stát po provedení příkazu QUIT 4D.

Příklad

Příkaz může být použit jako součást programování pro Web Server, pouze ve zkompilované verzi. Pokud použijete k poslání Web stránek smyčku jako While...End while, mechanismus Web serveru vám neumožní zastavit smyčku v případě časovače na prohlížeči Webu. Pokud není Web prohlížeč uzavřen, je jeho obsah stále používán. Umístění příkazu Process aborted do testu smyčky bude vracet True v případě časovače. Smyčka pak může být přerušena a proces odstraněn. Zde je metoda, která může být použita k poslání HTML stránky. Ve zkompilované databázi nemůže být tato smyčka přerušena v případě časovače:

```
While (True)
   SEND HTML FILE (HTMLSoubor)
End while
```

Použití Process aborted vám umožní stejný typ metody, smyčka může být ukončena a proces přerušen v případě časovače.

```
While (Not (Process aborted))
   SEND HTML FILE (HTMLSoubor)
End while
```

Příkazy a odkazy pro Procesy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Current process (Platný proces)

Příkazy a odkazy pro Procesy

Verze 3

Current process → Číslo

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Číslo	←	Číslo procesu

Popis

Příkaz Current proces vrací číslo procesu, ze kterého je tento příkaz volán.

Příklad

Podívejte se na příklad u příkazu DELAY PROCESS a PROCESS ATTRIBUTES.

Dále si přečtěte

Process number, PROCESS PROPERTIES, Process state.

Příkazy a odkazy pro Procesy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Process state (Stav procesu)

Příkazy a odkazy pro Procesy

Verze 3

Process state (proces) → Číslo

Parametr	Typ		Popis
proces	Číslo	→	Číslo procesu
Výsledek funkce	Číslo	←	Stav procesu

Popis

Příkaz Process state vrací stav procesu, jehož číslo jste zadali do parametru.

Výsledek funkce může být jedna z následujících konstant:

Konstanta	*Typ*	*Hodnota*
Aborted	*Long Integer*	*-1*
Delayed	*Long Integer*	*1*
Does not exist	*Long Integer*	*-100*
Executing	*Long Integer*	*0*
Hidden modal dialog	*Long Integer*	*6*
Paused	*Long Integer*	*5*
Waiting for input output	*Long Integer*	*3*
Waiting for internal flag	*Long Integer*	*4*
Waiting for user event	*Long Integer*	*2*

Jestliže proces neexistuje (tzn, že jste napředali číslo od 1 do Count tasks), Process state vrátí -100.

Příklad

Následující příklad vloží název a číslo každého procesu do array asProcName a aiProcNum. Metoda vyzkouší jestli nebyly nějaké procesy odstraněny. V tomto případě není název procesu a jeho číslo předáno do array.

```
$vlNbTasks:=Count tasks
ARRAY STRING(31;arProcName; $vlNbTasks)
ARRAY INTEGER(aiProcNum; $vlNbTasks)
$vlActualCount:=0
For ($vlProcess;1; $vlNbTasks)
   If (Process state($vlProcess)>=Executing)
      $vlActualCount:=$vlActualCount+1
      PROCESS PROPERTIES($vlProcess; asProcName{$vlActualCount}; $vlState;$vlTime)
      aiProcNum{$vlActualCount}:=$vlProcess
   End if
End for
   ` Odstranit nepoužité prvky
ARRAY STRING(31;asProcName;$vlActualCount)
ARRAY INTEGER(aiProcNum;$vlActualCount)
```

Dále si přečtěte

Count tasks, PROCESS PROPERTIES.

Příkazy a odkazy pro Procesy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

PROCESS PROPERTIES (VLASTNOSTI PROCESU)

Příkazy a odkazy pro Procesy

Verze 6.0 (Upraveno)

PROCESS PROPERTIES (proces; procNázev; procStav; procČas{; procVidět{; jedinečnéID{; původ}}})

Parametr	Typ		Popis
proces	Číslo	→	Číslo procesu
procNázev	Řetězec	←	Název procesu
procStav	Číslo	←	Stav procesu
procČas	Číslo	←	Celkový čas procesu v ticích
procVidět	Logické	←	Viditelné (True) nebo Skryté (False)
jedinečnéID	Integer	←	Jedinečné číslo procesu
původ	LongInt	←	Původ procesu

Popis

Příkaz PROCESS PROPERTIES vrací informace o procesu jehož číslo jste zadali do parametru proces.

Po provedení příkazu:

• procNázev bude obsahovat název procesu. Nějaké poznámky o názvu procesu:
- Pokud byl proces zapnutý z prostředí uživatele pomocí položky **Provést metodu** v nabídce Zvláštní, jeho název začíná na "P_" následované číslem.
- Pokud byl proces zapnut pomocí položky nabídky pro níž byla označena možnost **Začít nový proces**, začíná název procesu na "M_" nebo "ML_" následovaný číslem.
- Pokud byl proces odstraněn a jeho číslo ještě nebylo použito, je vrácen název bývalého procesu. K zjištění jestli byl proces odstraněn testujte procStav=-1.

• procStav vrací stav procesu v okamžiku volání. Tento parametr může vrátit jednu z následujícících konstant:

Konstanta	*Typ*	*Hodnota*
Aborted	*Long Integer*	*-1*
Delayed	*Long Integer*	*1*
Does not exist	*Long Integer*	*-100*
Executing	*Long Integer*	*0*
Hidden modal dialog	*Long Integer*	*6*
Paused	*Long Integer*	*5*
Waiting for input output	*Long Integer*	*3*
Waiting for internal flag	*Long Integer*	*4*
Waiting for user event	*Long Integer*	*2*

• procČas vrací kumulativní čas který byl procesem využit od jeho spuštění. Je udáno v ticích (1 sekunda = 60 tiků).

• provVidět, pokud bylo zadáno, vrací True pokud je proces viditelný a False pokud je skrytý.

• jedinečnéID, pokud je zadáno, vrací jedinečné číslo procesu. Nově od 4D verze 6.5 má každý proces číslo procesu stejně jako jedinečné číslo procesu pro počítač. Jedinečné číslo vám umožní rozlišit mezi dvěma procesy nebo dvěma spuštěními procesu.

• původ, pokud je definováno, vrací hodnotu která definuje původ procesu. 4th Dimension k tomu obsahuje následující konstanty:

Konstanta	*Typ*	*Hodnota*
Web Process with License	*Longint*	*-11*
Other 4D Process	*Longint*	*-10*

External Task	*Longint*	*-9*
Event Manager	*Longint*	*-8*
Apple Event Manager	*Longint*	*-7*
Serial Port Manager	*Longint*	*-6*
Indexing Process	*Longint*	*-5*
Cache Manager	*Longint*	*-4*
Web Process with no License	*Longint*	*-3*
Design Process	*Longint*	*-2*
User or Custom Menus Process	*Longint*	*-1*
None	*Longint*	*0*
Created from Programming	*Longint*	*1*
Created from Menu Command	*Longint*	*2*
Created from User Mode	*Longint*	*3*
Other User Process	*Longint*	*4*

Poznámka: Procesy vytvořené 4D vrací záporná čísla a procesy vytvořené uživatelem vrací čísla kladná.

Pokud proces neexistuje, PROCESS PROPERTIES vrátí proměnné nezměněné.

Příklady

1. Následující příklad vrací název, stav a čas procesu do proměnných vName, vState a vTime v platném procesu:

```
C_STRING(80; vName) ` Nastavit proměnné
C_INTEGER(vState)
C_INTEGER(vTime)
PROCESS PROPERTIES (Current process; vName; vState; vTimeSpent)
```

2. Podívejte se na příklad u metody databáze Při ukončení..

Dále si přečtěte

Count tasks, Process state.

Příkazy a odkazy pro Procesy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Process number (Číslo procesu)

Příkazy a odkazy pro Procesy

Verze 6.0

Process number (název{; *}) → Číslo

Parametr	Typ		Popis
název	Řetězec	→	Název procesu pro který chcete číslo procesu
*		→	Vrátit číslo procesu ze serveru
Výsledek funkce	Číslo	←	Číslo procesu

Popis

Process number vrátí číslo procesu jehož název zadáte do parametru název. Pokud nebyl nalezen žádný proces vrátí příkaz 0.

Volitelný parametr * vám umožní ze 4D Cient získat číslo procesu který běží na serveru (uložená procedura). V tomto případě je vrácená hodnota záporná. Tato možnost je využitelná s příkazy PROCESS VARIABLE a SET PROCES VARIABLE. Přečtěte si popis k těmto příkazům pro podrobnější informace.

Pokud je příkaz proveden na serveru s parametrem *, je vráceno kladné číslo.

Příklad

Vytvoříte si vlastní plovoucí okno v odděleném procesu, do kterého předáte vaše vlastní nástroje pro práci v prostředí návrháře. Například když označíte položku hierarchického seznamu klíčových slov, chcete vložit nějaký text do okna na popředí v prostředí návrháře. K tomu můžete využít Schránku, ale událost předání se musí stát v procesu návrháře. Následující malá funkce vrací číslo procesu návrháře (pokud je zapnutý):

```
    ` Metoda projektu Číslo procesu návrháře
    ` Číslo procesu návrháře → LongInt
    ` Číslo procesu návrháře → Číslo procesu návrháře
$0:=Process number(Get indexed string(170;3))
    ` Název procesu návrháře je uložen ve zdroji 'STR#" ID=170,
    ` řetězec #3 ve 4D
    ` Poznámka: Zde může vzniknout v budoucnu chyba, pokud se zdroj změní
```

S použitím této funkce vloží následující metoda projektu text předaný jako parametr do předního okna v prostředí návrháře (jestliže je to možné):

```
    ` Metoda projektu PASTE TEXT TO DESIGN
    ` PASTE TEXT TO DESIGN ( Text )
    ` PASTE TEXT TO DESIGN ( Text k předání do předního okna prostředí návrháře )
C_TEXT($1)
C_LONGINT($vlDesignPID;$vlCount)
$vlDesignPID:=Design process number
If ($vlDesignPID # 0)
        ` Vložit text do schránky
    SET TEXT TO CLIPBOARD($1)
        ` Zachytit událost Ctrl-V / Command-V
    POST KEY(Ascii("v");Command key mask;$vlDesignPID)
        ` Opakovaně volat DELAY PROCESS dokud nebude možné
        ` provést událost v procesu návrháře
```

```
    For ($vlCount;1;5)
        DELAY PROCESS( Current process;1)
    End for
End if
```

Dále si přečtěte

GET PROCESS VARIABLE, PROCESS PROPERTIES, Process state, SET PROCESS VARIABLE.

Příkazy a odkazy pro Procesy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Count users (Počet uživatelů)

Příkazy a odkazy pro Procesy

Verze 3

Count users → Integer_

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Integer	←	Počet uživatelů připojených k serveru

Popis

Funkce Count users vrací počet uživatelů připojených k databázi. Pokud je na serveru spuštěna jedna nebo více uložených procedur, Count users vrátí počet uživatelů plus 1.

Pokud používáte jednouživatelskou verzi 4th Dimension, vrátí Count users -1.

Dále si přečtěte

Count tasks, Count user processes.

Příkazy a odkazy pro Procesy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Count tasks (Počet procesů)

Příkazy a odkazy pro Procesy

Verze 3

Count tasks → Integer

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Integer	←	Počet otevřených procesů (včetně procesů vytvořených 4D)

Popis

Count tasks vrací počet procesů spuštěných na počítači jednouživatelské 4th Dimension.

Vrácené číslo obsahuje všechny procesy, včetně těch, které řídí 4th Dimension. To jsou procesy uživatele/aplikace, návrháře, Cache manažer, Indexování a Web server.

Číslo vracené funkcí Count tasks zahrnuje také procesy které byly odstraněny.

Příklad

Podívejte se na příklad u metody databáze Při ukončení.

Dále si přečtěte

Count user processes, Count users, PROCESS PROPERTIES, Process state.

Příkazy a odkazy pro Procesy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Count user processes (Počet procesů uživatele)

Příkazy a odkazy pro Procesy

Verze 3

Count user processes → Integer

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Integer	←	Počet otevřených procesů (mimo procesů vytvořených 4th Dimension)

Popis

Příkaz Count user processes vrací počet otevřených procesů, mimo těch které automaticky řídí 4th Dimension.

Proces uživatele/aplikace, Návrháře a Web server jsou považovány za procesy uživatele, protože je může uživatel ukončit. Proto budou tyto procesy zahrnuty do součtu procesů uživatele.

Dále si přečtěte

Count tasks, Count users.

Příkazy a odkazy pro Procesy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

EXECUTE ON CLIENT (PROVÉST NA KLIENTOVI)

Verze 6.5

EXECUTE ON CLIENT (NázevKlient; metoda{; param}{; param2; ...; paramN})

Parametr	Typ		Popis
NázevKlient	Řetězec	→	Registrovaný název 4D Client
metoda	Řetězec	→	Název metody ke spuštění
param		→	Parametry metody

Popis

Příkaz EXECUTE ON CLIENT vám umožní provést metodu metoda s parametry param....paramN na registrovaném 4D Clinetovi NázevKlient. Název klienta se definuje příkazem REGISTER CLIENT.

Tento příkaz může být volán jak z klienta tak z uložené procedury.

Pokud si metoda vyžaduje jeden nebo více parametrů, vložte je do parametrů za název metody.Spuštění metody na straně 4D Client je provedeno v nově vytvořeném globálním procesu a jeho název bude registrovaný název 4D klienta.

Pokud je tento příkaz volán mnohokrát ze stejného klienta, pořadí spouštení bude nahromaděno. Proto budou metody prováděny jedna za druhou. Více metod které jsou nahromaděny, zpomalí práci 4D Client. Pomocí příkazu GET REGISTERED CLIENT můžete zjistit počet metod které budou spuštěny na klientu.

Poznámka: Hromadění metod nemůže být zastaveno nebo upraveno dokud není 4D Client odhlášen příkazem UNREGISTER CLIENT.

Jednu metodu můžete najednou spustit na více nebo na všech registrovaných klientech. K tomu použijte znak výběru náhradou (@) místo názvu klienta.

Systémová proměnná OK je nastavena na 1, pokud 4D Sever správně obdržel prováděcí požadavek od rodičovského 4D Client; nicméně to neznamená, že byla metoda správně spuštěna na cílovém 4D Client.

Příklady

1. Předpokládejme že chcete spustit metodu "GenerateNums" na klientovi "Client1":

```
EXECUTE ON CLIENT("Client1";"GenerateNums";12;$a;"Text")
```

2. Pokud chcete na všech klientech provést metodu "EmptyTemp":

```
EXECUTE ON CLIENT ("@";"EmptyTemp")
```

3. Podívejte se na příklad u příkazu REGISTER CLIENT.

Dále si přečtěte

GET REGISTERED CLIENTS, REGISTER CLIENT, UNREGISTER CLIENT.

REGISTER CLIENT (REGISTROVAT KLIENTA)

Příkazy a odkazy pro Procesy

Verze 6.5

REGISTER CLIENT (NázevKlienta{; perioda{; *}})

Parametr	Typ		Popis
NázevKlienta	Řetězec	→	Název spuštěné sekce 4D Client
perioda	LongInt	→	perioda dotazování Serveru na úlohy (v sekundách) není-li uvedena, 2 sekundy
*	*	→	Proces bude místní

Popis

Příkaz REGISTER CLIENT "zaregistruje" na serveru sekci 4D Client pod názvem který jste vložili do parametru NázevKlienta a tím umožnit jiným klientů nebo případně 4D Serveru (s použitím uložených procedur) spuštění metod na tomto klientu s použitím příkazu EXECUTE ON CLIENT. Jakmile je jednou zaregistrován, může 4D Client provádět jednu nebo více metod pro ostatní klienty.

Poznámka: Všechny klienty můžete registrovat hned při spuštění 4D Serveru pokud zaškrtnete možnost "Registrovat klienty při přihlášení" v okně Předvolby databáze.

Jakmile provedete tento příkaz, je na klientovi nastartován proces s názvem NázevKlienta. Tento proces může být odstraněn pouze pomocí příkazu UNREGISTER CLIENT. Pokud předáte parametr *, je vytvořený proces místní. 4D automaticky přidá znak dolaru ($) před název procesu. Jinak bude proces globální.

Po spuštění tohoto procesu se bude 4D Client pravidelně ptát na 4D Server jestli jej nějaký jiný klint nevolal. Jako výchozí je tento dotaz prováděn každé dvě sekundy. Tuto periodu můžete změnit zadáním hodnoty do volitelného parametru perioda. Minimální hodnota je jedna sekunda.

Poznámka: Pokud bude tento příkaz použit v jednouživatelské verzi 4th Dimension, nestane se nic.

Jakmile je jednou příkaz proveden, nemůžete za běhu změnit název klienta nebo periodu dotazů. Jediný způsob jak to udělat je odregistrovat klienta pomocí UREGISTER CLIENT a pak jej opět zaregistrovat příkazem REGISTER CLIENT.

Poznámka: Stejný registrační název může mít více klientů.

Pokud je klient správně zaregistrován, je proměnná OK nastavena na 1. Pokud je již 4D Client registrovaný, příkaz neprovede nic a do proměnné OK doplní 0.

Příklady

V následujícím příkladu vytvoříme jednoduchý komunikační systém pro komunikace mezi klienty.

1. Metoda *Registrace* vám umožní zaregistrovat klienta a připravit jej pro příjem zpráv z jiných klientů:

```
   ` Před přihlášením pod jiným názvem, je potřeba nejdříve klienta odregistrovat
UNREGISTER CLIENT
Repeat
   vPseudoName:=Request("Vložte vaše jméno:";"Uživatel";"OK";"Zrušit")
Until ((OK=0) | (vPseudoName # ""))
If (OK=0)
   ...` Nedělat nic
Else
```

```
    REGISTER CLIENT(vPseudoName)
End if
```

2. Následující řádek vám umožní vytvořit seznam registrovaných klientů. Může být umístěn do metody databáze Při spuštění:

```
prClientList:=New process ("4D Client List";32000;Seznam registrovaných klientů")
```

3. Metoda *4D Client List* vám umožní obnovit seznam všech registrovaných klientů a těch kteří mohou přijímat zprávy:

```
If (Application type=4D Client)
      ` následující kód se provede pouze na aplikaci 4D Client
   $Ref:=Open window(100;100;300;400;-(Palette window+Has window title);"Registrovaní klienti")
   Repeat
      GET REGISTERED CLIENTS($ClientList;$ListeCharge)
         `dosadit registrované klienty do $ClientList
      ERASE WINDOW($Ref)
      GOTO XY(0;0)
      For ($p;1;Size of array($ClientList))
         MESSAGE($ClientList{$p}+Char(Carriage return))
      End for
      `Zobrazit každou sekundu
      DELAY PROCES(Current process;60)
   Until (False) ` nekonečná smyčka
End if
```

4. Následující metoda vám umožní poslat zprávu jinému registrovanému klientovi. Volá metodu *Display_Message*:

```
$Addressee:=Request("Adresát zprávy:";"")
   ` Vložte jméno člověka zobrazené v okně vytvořeném
   ` metodou databáze Při spuštění
If (OK # 0)
   $Message:=Request("Zpráva:") ` Zpráva
   If (OK # 0)
      EXECUTE ON CLIENT($Addressee;"Display_Message";$Message) ` Poslat zprávu
   End if
End if
```

5. Zde je metoda *Display:Message*:

```
C_TEXT($1)
ALERT($1)
```

6. Nakonec umožní tato metoda odregistrovat klienta, aby již nebyl viditelný z jiných strojů a nepřijímal zprávy:

```
UNREGISTER CLIENT
```

Dále si přečtěte

Systémové proměnné a Sady

Pokud je 4D Client správně zaregistrován, je proměnná OK nastavena na 1. Pokud již byl klient registrovaný, příkaz neprovede nic a nastaví proměnnou OK 0.

Příkazy a odkazy pro Procesy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

UNREGISTER CLIENT (ODREGISTROVAT KLIENTA)

UNREGISTER CLIENT

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz UNREGISTER CLIENT odregistruje klienta. Klient musel být dříve zaregistrován příkazem REGISTER CLIENT.

Poznámka: Klient je automaticky odregistrován při vypnutí aplikace na klientovi.

Pokud nebyl počítač klienta zaregistrován, nebo je příkaz proveden v jednouživatelské verzi, příkaz neprovede nic.

Pokud je klient správně odhlášen, je proměnná OK nastavena na 1. Pokud klient nebyl předtím registrován, je proměnná OK nastavena na 0.

Příklad

Podívejte se na příklad u příkazu REGISTER CLIENT.

Dále si přečtěte

EXECUTE ON CLIENT, GET REGISTERED CLIENTS, REGISTER CLIENT.

Příkazy a odkazy pro Procesy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GET REGISTERED CLIENTS (ZÍSKAT REGISTROVANÉ KLIENTY)

Verze 6.5

GET REGISTERED CLIENTS (seznamKlientů; metody)

Parametr	Typ		Popis
seznamKlientů	Textový array	←	Seznam uložených 4D Clientů
metody	LongInt array	←	Seznam počtu metod, které budou prováděny na klientu pomocí příkazu Execute on client

Popis

Příkaz GET REGISTERED CLIENTS naplní dvě array:

• seznamKlientů obsahuje seznam klientů registrovaných pomocí příkazu REGISTER CLIENT.
• metody zastupuje seznam toho, co má klient provést. Tento seznam je seznam metod, které musí klient ještě provést z volání příkazu EXECUTE ON CLIENT (Jestli chcete vědět více informací přečtěte si popis u příkazu EXECUTE ON CLIENT).

Poznámka: Pokud byla akce správně dokončena, je proměnná OK nastavena na 1.

Příklady

1. Řekněme že potřebujete seznam všech registrovaných klientů a seznam jejich metod ke spuštění:

```
ARRAY TEXT($clients;0)
ARRAY LONGINT($methods;0)
GET REGISTERED CLIENTS($clients;$methods)
```

2. Podívejte se na příklad u příkazu REGISTER CLIENT.

Dále si přečtěte

EXECUTE ON CLIENT, REGISTER CLIENT, UNREGISTER CLIENT.

Příkazy a odkazy pro Procesy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

QUERY BY EXAMPLE (DOTAZ DLE PŘÍKLADU)

Verze 3

QUERY BY EXAMPLE ({tabulka}{;}{*})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro kterou provést hledání. Pokud vynecháno, použít výchozí tabulku
*		→	Pokud předáno, bude zobrazen posuvník

Popis

Příkaz QUERY BY EXAMPLE provede stejnou akci jako položka nabídky Dotaz dle příkladu v prostředí uživatele. Zobrazí platný vstupní formulář jako dialog pro hledání.

QUERY BY EXAMPLE bude hledat podle dat, která zadá uživatel do okna hlednání. Formulář musí obsahovat pole podle kterých chcete uživatelům umožnit hledání. Hledání je optimalizováno: indexovaná pole jsou automaticky použita pro optimalizaci.

O položce nabídky Dotaz dle příkladu si přečtěte v *Příručce uživatele 4th Dimension.*

Příklad

Metoda v tomto příkladu zobrazí formulář VlastníHledání. Pokud uživatel potvrdí formulář a tím provede hledání (to znamená že se proměnná OK doplní za 1), zobrazí se nalezené záznamy.

```
INPUT FORM ([Lidé]; "VlastníHledání") ` Přepnout na formulář pro hledání
QUERY BY EXAMPLE ([Lidé]) ` Zobrazit formulář a provést hledání
If (OK=1) ` Pokud uživatel provedl hledání
   DISPLAY SELECTION ([Lidé]) ` Zobrazit záznamy
End if
```

Dále si přečtěte

ORDER BY, QUERY.

Systémové proměnné a Sady

Pokud uživatel klepne na tlačítko Přijmout nebo stiskne klávesu Enter, je proměnná OK nastavena na 1 a je provedeno hledání. Jestliže uživatel klepne na Zrušit, nebo stiskne kombinaci kláves pro zrušení, je systémová proměnná OK nastavena na 0 a hledání je zrušeno.

4th Dimension 6.5, Seznam témat konstant

QUERY (DOTAZ)

Příkazy a odkazy pro Dotazy

Verze 3

QUERY ({tabulka}{; argumentdotazu}{ ; *})

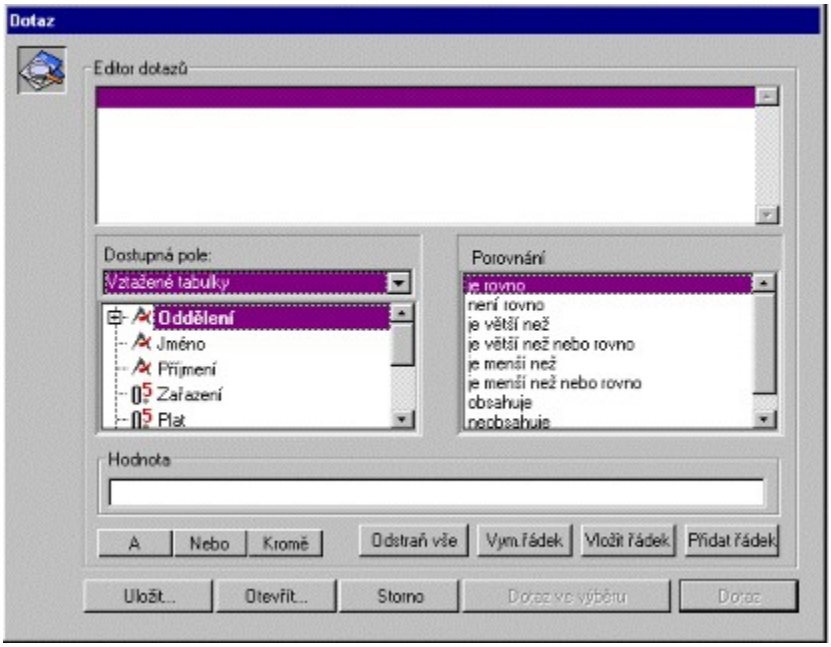
Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro kterou vrátit výběr záznamů, pokud vynecháno, použít Výchozí tabulku
argumentdotazu		→	Jeden řádek dotazu
*		→	Příznak pokračování dotazu

Popis

Příkaz QUERY vyhledá záznamy vyhovující zadaným podmínkám a zobrazí je ve výběru záznamů pro tabulku. QUERY změní platný výběr pro tabulku v jednom procesu a z prvního nalezeného záznamu udělá platný záznam.

Pokud je vynechán parametr tabulka, je použita výchozí tabulka. Pokud nebyla nastavena žádná výchozí tabulka, je generována chyba.

Pokud nedefinujete argument dotazu nebo *příkaz zobrazí Editor dotazů pro tabulku. Editor dotazů pro prostředí uživatele je zde zobrazen:



Jestli chcete vědět více informací o Editoru dotazů, přečtěte si *Příručku uživatele 4th Dimension.*

Uživatel vytvoří dotaz a pak klepne na tlačítko Dotaz nebo Dotaz ve výběru pro provedení dotazu. Pokud je dotaz proveden bez přerušení, je proměnná OK nastavena na 1. Pokud uživatel klepne na Storno v okně dotazu, zruší tím hledání a proměnná OK je nastavena na 0.

Příklady

1. Následující příklad zobrazí Editor dotazů pro tabulku [Produkty]:

```
QUERY ([Produkty])
```

2. Následující příklad zobrazí Editor dotazů pro výchozí tabulku (pokud byla nastavena):

QUERY

Pokud zadáte argumentdotazu, není okno dotazů zobrazeno a hledání je zadáno programově. Pro jednoduché hledání pdle jednoho pole většinou nastavíte argumentdotazu. Pro složitější hledání (hledání podle několika polí nebo několika podmínek), provedete příkaz QUERY tolikrát, kolik je potřeba argumentů pro dotaz a definujete parametr * kromě posledního provedení příkazu. Argumentdotazu je popsán dále v této části.

Příklady

3. Následující příklad bude hledat lidi jejichž jméno začíná na "a":

QUERY ([Lidé];[Lidé]Jméno="a@")

4. Následující příklad bude hledat lidi jejichž jméno začíná na "a" nebo "b":

QUERY ([Lidé];[Lidé]Jméno="a@";*) ` * značí že jsou ještě další podmínky hledání
QUERY ([Lidé]; | ;[Lidé]Jméno="b@") ` Bez * značí že je konec definování a dotaz může být spuštěn

Definování argumentu dotazu

• Parametr argumentdotazu používá následující zápis:
{spojení ; } pole operátor Hodnota
• Spojení je použito pro spojení jednotlivých řádků dotazu. Jsou dostupná stejná spojení jako v Editoru dotazů:

Spojení	***Symbol užívaný v QUERY***
A	*&*
Nebo	|
Kromě	#

Spojení je volitelné a nepoužívá s v prvním příkazu QUERY nebo pokud je dotaz jednoduchý.

• pole je pole k hledání. Může patřit k jiné tabulce pokud náleží tabulce Jedince vztažené k tabulce automatickým vztahem. Tabulka ve které je dotaz prováděn musí být tabulkou skupiny.

• operátor je znak pro porovnání mezi polem a hodnotou. Musí to být jeden z následujících symbolů:

Operátor	***Symbol pro použití v QUERY***
Je rovno	=
Není rovno	#
Menší než	<
Větší než	>
Menší nebo rovno	<=
Větší nebo rovno	>=

• Hodnota jsou data proti kterým bude pole porovnáváno. Hodnota může být jakýkoli výraz který je stejného typu jako data v poli. Hodnota je zadána jednou na začátku hledání. Pro hledání řetězce obsaženého v řetězci použijte v hodnotě znak výběru náhradou (@).

Zde jsou pravidla pro vytvoření složitého dotazu:
• První argument dotazu nesmí obsahovat operátor.
• Každý následující argument dotazu musí začínat operátorem.
• První a všechny další, mimo posledního, musí končit znakem *.
• K provedení dotazu nezadávejte znak * do posledního řádku QUERY.

Můžete také provést poslední příkaz QUERY bez parametrů mimo tabulky (Editor dotazů se nezobrazí dříve než bude dokončen zadaný dotaz).

Poznámka: Každá tabulka může mít svoje vlastní hledání. To znamená že můžete najednou provádět více dotazů ve více tabulkách. Musíte použít porametr tabulka nebo vždy definovat výchozí tabulku.

Nezáleží na tom, jak byl dotaz vytvořen:

• Pokud zadaný dotaz bude trvat určitou dobu, 4th Dimension automaticky zobrazí okno s teploměrem. Toto upozornění můžete vypnout nebo zapnout příkazy MESSAGES ON nebo MESSAGES OFF. Pokud je teploměr zobrazen, může uživatel klepnout na tlačítko Stop a tím přerušit dotaz. Pokud je dotaz dokončen je proměnná OK nastavena na 1, jinak je nastavena na 0.
• Pokud je vybráno nějaké indexované pole, pokud je to možné je dotaz automaticky optimalizován (podle indexovaného pole je hledáno nejdříve) aby se při hledání ušetřilo co nejvíce času.

Příklady

5. Následující příklad najde všechny lidi s příjmením Kovář:

```
QUERY ([Lidé];[Lidé]Přijmení="Kovář")
```

Poznámka: Pokud je pole Příjmení indexováno, příkaz QUERY automaticky použije indexy pro hledání.

Připomínka: Toto hledání nalezne všechny záznamy jako "kovář", "Kovář", "KOVÁŘ", atd. Aby jste odlišili rozdíly mezi malými a velkými písmeny, použijte ASCII kódy.

6. Následující příklad vyhledá všechny lidi se jménem Jan Kovář. Přijmení je indexované, Jméno není:

```
QUERY ([Lidé];[Lidé]Přijmení="Kovář";*) ` Nalezne všechny osoby s příjmením Kovář
QUERY ([Lidé]; & ;[Lidé]Jméno="Jan") ` a se jménem Jan
```

Když je dotaz prováděn, je nejdříve provedeno rychlé indexované hledání v poli Příjmení a omezí se výběr záznamů na nalezené zaznamy a pak je provedeno postupné hledání v tomto výběru podle pole Jméno.

7. Následující příklad najde lidi s příjmením "Kovář" nebo "Novák". Pole Příjmení je indexované:

```
QUERY ([Lidé];[Lidé]Přijmení="Kovář";*) ` Nalezne všechny osoby s příjmením Kovář
QUERY ([Lidé]; | ;[Lidé]Přijmení ="Novák") ` ... nebo Novák
```

Příkaz QUERY použije rychlé indexované hledání pro obě hodnoty. Dva dotazy jsou provedeny a jejich výsledek je předán do jednotlivých Sad. Obě dvě sady jsou pak zkombinovány.

8. Následující příklad nalezne všechny lidi kteří nemají zadaný název firmy. Bude se hledat prázdné pole (prázdný řetězec):

```
QUERY ([Lidé]; | ;[Lidé]Firma ="") ` Najít všechny lidi bez firmy
```

9. Následující příklad nalezne všechny lidi s Příjmením "Kovář" kteří pracují ve firmě se sídlem v Praze. Druhý dotaz používá hledání v jiné tabulce. Toto hledání může být provedeno, protože tabulka [Lidí] je vztažena k tabulce [Firmy] automatickým vztahem skupiny k jedinci:

```
    ` Nalézt všechny osoby s Příjmením Kovář...
QUERY ([Lidé];[Lidé]Přijmení="Kovář";*)
    ` kteří pracují ve firmě se sídlem v Praze.
QUERY ([Lidé]; & ;[Firmy]Město="Praha")
```

10. Následující příklad nalezne všechny osoby jejichž jméno začíná od A do N:

```
QUERY ([Lidé];[Lidé]Jméno < "n") ` Najít všechny se jménem mezi A a N
```

11. Následující příklad nalezne všechny lidi kteří bydlí v Praze 4 nebo v Praze 5 (PSČ začínající na 14 nebo 15)

```
QUERY ([Lidé];[Lidé]PSČ = "14@"; *) ` Najít všechny lidi z Prahy 4
QUERY ([Lidé]; | ;[Lidé]PSČ = "15@") ` ...nebo Prahy 5
```

12. Následující příklad hledá podle indexovaného podpole. Dotaz vrátí výběr rodičovských záznamů (záznamů v

tabulce [Lidé] k nimž patří podzáznamy). Není vytvořen výběr podzáznamů. Výsledek bude výběr všech lidí kteří mají dítě jménem "Martin":

```
QUERY ([Lidé];[Lidé]Děti'Jméno = "Martin") ` Nalezne lidi s dítětem Martin
```

13. Následující příklad nalezne záznamy pro číslo faktury zadaný do okna žádosti:

```
vFind:=Request("Najít fakturu číslo:") ` Dostat číslo faktury od uživatele
If (OK = 1) ` Pokud uživatel klepne na OK
      ` Najít fakturu s číslem uloženým v vFind
   QUERY ([Faktury]; [Faktury]Číslo = vFind)
End if
```

14. Následující příklad nalezne všechny faktury vytvořené v roce 1998. To znamená faktury po 31.12.97 a před 1.1.99:

```
QUERY ([Faktury];[ Faktury]Datum > !31.12.97!;*) ` Najít faktury po 31.12.97
QUERY ([Faktury]; & ;[ Faktury]Datum < !1.1.99!) ` a před 1.1.99
```

15. Následující příklad nalezne všechny zaměstnance jejichž plat je mezi 10000 Kč a 50000 Kč. Dotaz nalezne zaměstnance s platem nad 10000 Kč včetně ale pod 50000 Kč:

```
QUERY ([Zaměstnanci];[ Zaměstnanci]Plat >= 10000;*)
QUERY ([Zaměstnanci]; & ;[ Zaměstnanci]Plat < 50000)
```

16. Následující příklad nalezne všechny zaměstnance v marketingovém oddělení, jejichž plat je nad 20000 Kč. Nejdříve je vyhledáváno podle pole Plat, protože je indexované. Druhé hledání je z jiné tabulky. Je to možné protože tabulka [Oddělení] je vztažena k tabulce [Zaměstnanci] automatickým vztahem skupiny k jedinci. Ačkoli je pole [Oddělení]Název indexované, není prováděno indexované hledání, protože vztah musí být aktivován pro každý záznam zvlášť:

```
    ` Nalézt zaměstnance s platem přes 20000 Kč a ...
QUERY ([Zaměstnanci];[ Zaměstnanci]Plat > 20000;*)
    ` pracuje v oddělení marketing.
QUERY ([Zaměstnanci]; & ;[ Oddělení]Název = "Marketing")
```

17. Následující příklad vyhledá informace obsažené v proměnné mojeProm:

```
QUERY ([Zákony];[Zákony]Text = MojeProm) ` Najde všechny zákony které vyhovují proměnné
```

Hledání může mít několik výsledků podle toho, jaká je hodnota proměnné MojeProm. Dotaz bude rozdílný například pro:

• Pokud mojeProm bude obsahovat "Copyright@", bude výběr obsahovat všechny záznamy jejichž pole Text začíná na Copyright.

• Pokud bude MojeProm obsahovat "@Copyright@", bude výběr obsahovat všechny záznamy jejich pole Text obsahuje Copyright.

Dále si přečtěte

QUERY SELECTION.

Příkazy a odkazy pro Dotazy

4th Dimension 6.5, Seznam témat příkazů

QUERY SELECTION (DOTAZ VE VÝBĚRU)

Příkazy a odkazy pro Dotazy

Verze 3

QUERY SELECTION ({tabulka} {; argumentdotazu} {; *})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro kterou vrátit výběr záznamů, pokud vynecháno, použít Výchozí tabulku
argumentdotazu		→	Jeden řádek dotazu
*		→	Příznak pokračování dotazu

Popis

QUERY SELECTION vyhledá záznamy v tabulce. Mění platný výběr pro tabulku v platném procesu a první nalezený záznam udělá platným záznamem.

QUERY SELECTION provádí to samé jako QUERY. Rozdíl mezi těmito příkazy je v rozsahu hledání:

- QUERY hledá ve všech záznamech pro tabulku.
- QUERY SELECTION hledá pouze v platném výběru záznamů.

Jestli chcete vědět více informací, přečtěte si příkaz QUERY.

Poznámka: Příkaz SET DATABASE PARAMETER vám umožní zvolit jestli bude QUERY SELECTION používat indexy podle počtu záznamů ve výběru.

Příklad

Tento příklad ukazuje rozdíl mezi příkazem QUERY a QUERY SELECTION. Zde jsou dvě hledání:

```
    ` Najít VŠECHNY firmy se sídlem v Praze
QUERY ([Firmy]; [Firmy]Město="Praha")
    ` Najít VŠECHNY firmy které se zabývají Bankovnictvím
    ` bez rozdílu kde mají sídlo
QUERY ([Firmy]; [Firmy]Zaměření="Bankovnictví")
```

Nezapomeňte že druhé hledání ignoruje výběr z prvního hledání:

```
    ` Najít VŠECHNY firmy se sídlem v Praze
QUERY ([Firmy]; [Firmy] Město="Praha")
    ` Najít firmy které se zabývají Bankovnictvím
    ` se sídlem v Praze
QUERY SELECTION ([Firmy]; [Firmy] Zaměření="Bankovnictví")
```

QUERY SELECTION bude hledat pouze v záznamech vybraných prvním hledáním. V tomto případě firmy se sídlem v Praze.

Dále si přečtěte

QUERY, SET DATABASE PARAMETER.

Příkazy a odkazy pro Dotazy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

QUERY BY FORMULA (DOTAZ DLE VÝRAZU)

Příkazy a odkazy pro Dotazy

Verze 3

QUERY BY FORMULA ({tabulka} {; VýrazDotazu})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro kterou vrátit výběr záznamů, pokud vynecháno, použít Výchozí tabulku
VýrazDotazu		→	Výraz pro dotaz

Popis

QUERY BY FORMULA vyhledá záznamy v tabulce. Mění platný výběr pro tabulku v platném procesu a první nalazený záznam udělá platným záznamem.

QUERY BY FORMULA a QUERY SELECTION BY FORMULA pracují stejným způsobem s tím rozdílem, že QUERY BY FORMULA hledá ve všech záznamech a QUERY SELECTION BY FORMULA hledá pouze v platném výběru záznamů.

Oba příkazy použijí VýrazDotazu na každý záznam v tabulce nebo výběru. VýrazDotazu je logický výraz který musí vracet True nebo Flase. Pokud vrátí VýrazDotazu True, je záznam zařazen do nového výběru.

VýrazDotazu může být jednoduchý, například porovnání pole s hodnotou, nebo složitý, například provedení nějakého výpočtu nebo použití hodnot ze vztažených tabulek. VýrazDotazu může být funkce 4th Dimension (příkaz) nebo funkce (metoda) nebo výraz který vytvoříte. Při hledání v Alfa nebo Textových polích můžete použít znaky výběru náhradou.

Jakmile je dotaz dokončen, načte se první záznam výběru do paměti a stane se platným záznamem pro tabulku.

Tyto příkazy vždy provádějí postupné a neindexované hledání. Při použití na indexované pole jsou příkazy QUERY BY FORMULA a QUERY SELECTION BY FORMULA jsou pomalejší než QUERY. Čas dotazu je násoben podle počtu záznamů v tabulce nebo výběru.

4D Server: Server neprovádí výraz dotazu. Každý záznam je poslán na stanici a výraz je proveden až na klientovi. To dělá tento příkaz na 4D Serveru o mnoho pomalejší než příkaz QUERY.

Příklady

1. Následující příklad najde záznamy všech faktur které byli zadány v měsíci prosinci jakéhokoli roku. Udělá to tím, že použije funkci Mont of na každý záznam. Toto hledání nemůže být provedeno jiným způsobem, pokud by se nevytvořilo specielní pole pro měsíc:

```
` Najít faktury zadané v Prosinci
QUERY BY FORMULA ([Faktury]; Month of ([Faktury]Zadáno) = 12)
```

2. Následující příklad nalezne všechny lidi kteří mají jméno delší než 10 znaků:

```
` Nalézt lidi s jménem delším než 10 znaků
QUERY BY FORMULA ([Lidé]; Length ([Lidé]Jméno)>10)
```

Dále si přečtěte

QUERY, QUERY SELECTION, QUERY SELECTION BY FORMULA.

Příkazy a odkazy pro Dotazy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

QUERY SELECTION BY FORMULA (DOTAZ VE VÝBĚRU DLE VÝRAZU)

Verze 3

QUERY SELECTION BY FORMULA ({tabulka} {; VýrazDotazu })

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro kterou vrátit výběr záznamů, pokud vynecháno, použít Výchozí tabulku
VýrazDotazu		→	Výraz pro dotaz

Popis

QUERY SELECTION BY FORMULA vyhledá záznamy v tabulce. Mění platný výběr pro tabulku v platném procesu a první nalezený záznam udělá platným záznamem.

QUERY SELECTION BY FORMULA provádí stejnou akci jako QUERY BY FORMULA. Rozdíl mezi těmito příkazy je v rozsahu hledání:

- QUERY BY FORMULA hledá ve všech záznamech pro tabulku.
- QUERY SELECTION BY FORMULA hledá pouze v platném výběru záznamů.

Jestli chcete vědět více informací, přečtěte si příkaz QUERY BY FORMULA.

Dále si přečtěte

QUERY, QUERY BY FORMULA, QUERY SELECTION.

Příkazy a odkazy pro Dotazy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET QUERY DESTINATION (NASTAVIT CÍLENÍ DOTAZŮ)

Příkazy a odkazy pro Dotazy

Verze 6.0

SET QUERY DESTINATION (cílovýTyp{; cílovýObjekt})

Parametr	Typ		Popis
CílovýTyp	číslo	→	0 platný výběr 1 sada (set) 2 pojmenovaný výběr (named selection) 3 proměnná
cílovýObjekt	řetězec | proměnná	→	Název sady nebo Název pojmenovaného výběru nebo Proměnná

Popis

Příkaz SET QUERY DESTINATION vám umožní nastavit kam v platném procesu 4th Dimension umístí výsledek hledání.

Typ umístění definujete v parametru CílovýTyp. 4th Dimension obsahuje následující předdefinované konstanty:

Konstanta	***Typ***	***Hodnota***
Into current selection	*Long Integer*	*0*
Into set	*Long Integer*	*1*
Into named selection	*Long Integer*	*2*
Into variable	*Long Integer*	*3*

Výsledek dotazu se vloží do objektu který definujete ve volitelném parametru cílovýObjekt. Tento parametr nastavujte podle následující tabulky:

Cílový typ	***Cílový objekt***
0 (platný výběr)	*Vynecháte tento parametr*
1 (sada)	*Předáte název sady (exitující nebo k vytvoření)*
2 (pojmenovaný výběr)	*Předáte název pojmenovaného výběru (exitující nebo k vytvoření)*
3 (proměnná)	*Předáte číselnou proměnnou (exitující nebo k vytvoření)*

S nastavením:

```
SET QUERY DESTINATION (Into current selection)
```

Nalezené záznamy se vloží do nového platného výběru záznamů.

S nastavením:

```
SET QUERY DESTINATION (Into set;"mojeSada")
```

Nalezené záznamy se vloží do sady s názvem "mojeSada". Platný výběr a platný záznam pro tabulku zůstanou nezměněné.

S nastavením:

```
SET QUERY DESTINATION (Into named selection;"PojmVýb")
```

Nalezené záznamy budou předány do pojmenovaného výběru s názvem "PojmVýb". Platný výběr a platný záznam pro tabulku zůstanou nezměněné.

S nastavením:

SET QUERY DESTINATION (Into variable;$vlVýsledek)

Počet nalezených záznamů bude předán do proměnné $vlVýsledek. Platný výběr a platný záznam pro tabulku zůstanou nezměněné.

Upozornění: SET QUERY DESTINATION nastaví všechny dotazy prováděné v platném procesu. NEZAPOMEŇTE vždy při použití příkazu SET QUERY DESTINATION (kde cílový typ # 0) provést SET QUERY DESTINATION(0) pro nastavení cílení na normální.

SET QUERY DESTINATION mění chování následujících příkazů:

- QUERY
- QUERY SELECTION
- QUERY BY EXAMPLE
- QUERY BY FORMULA
- QUERY SELECTION BY FORMULA

Příkaz SET QUERY DESTINATION nemění chování ostatních příkazů které mění platný výběr jako ALL RECORDS, RELATE MANY, atd.

Příklady

1. Vytvoříte formulář, který bude zobrazovat záznamy z tabulky [Telefony]. Vytvoříte ovládací kartu s názvem asRolodex (s 26 písmeny abecedy) a podformulář zobrazující záznamy z tabulky [Telefony]. Vybrání stránky na ovládací kartě zobrazí záznamy telefonního seznamu začínající na patřičné písmeno.

Ve vaší aplikaci, tabula [Telefony] obsahuje pevná data, a tak nechcete (nebo potřebujete) provádět dotaz pokaždé když vyberete jinou stránku ovládací karty. Tímto způsobem můžete ušetřit čas motoru databáze.

K tomu můžete přesměrovat vaše dotazy do pojmenovaného výběru, který můžete použít znovu když to bude potřeba. Napíšete metodu objektu pro ovládací kartu následovně:

```
   ` Metoda objektu asRolodex
Case of
   ÷ (Form event=On Load)
         ` Před tím, než se formulář objeví na obrazovce,
         ` nastavíte kartu a logické array které nám řekne
         ` jestli byl dotaz pro patřičné písmeno již proveden nebo ne
      ARRAY STRING(1;asRolodex;26)
      ARRAY BOOLEAN(abQueryDone;26)
      For ($vlElem;1;26)
         asRolodex{$vlElem}:=Char(64+$vlElem)
         abQueryDone{$vlElem}:=False
      End for
   ÷ (Form event=On Clicked)
         ` Pokud bude klepnuto na kartu,
         ` otestovat jestli byl patřičný
         ` dotaz proveden
      If (Not(abQueryDone{asRolodex}))
            ` Pokud ne, přesměrovat další dotaz(y) do pojmenovaného výběru
         SET QUERY DESTINATION(Into named selection; "Rolodex"+asRolodex{asRolodex})
            ` Provést dotaz
         QUERY([Telefony];[ Telefony]Příjmení=asRolodex{asRolodex}+"@")
            ` Vrátiti normální mód hledání
```

```
            SET QUERY DESTINATION(Into current selection)
                ` Podruhé když vybereme toto písmeno, nebudeme provádět dotaz
            abQueryDone{asRolodex}:=True
        End if
            ` Použijte pojmenovaný výběr pro platný výběr
            ` podle vybrané stránky karty
        USE NAMED SELECTION("Rolodex"+asRolodex{asRolodex})
    ÷ (Form event=On Unload)
            ` Po odstranění formuláře z obrazovky
            ` odstranit pojmenované výběry, které jsme vytvořili
        For ($vlElem;1;26)
            If(abQueryDone{$vlElem})
                CLEAR NAMED SELECTION("Rolodex"+asRolodex{$vlElem})
            End if
        End for
            ` Vymazat i dva array které již nepotřebujeme
        CLEAR VARIABLE(asRolodex)
        CLEAR VARIABLE(abQueryDone)
End case
```

2. Metoda projektu Jedinečná hodnota z tohoto příkladu otestuje jedinečnost hodnoty pro číselná pole tabulky. Platný záznam může být existující nebo nový.

```
    ` Metoda projektu Jedinečná hodnota
    ` Jedinečná hodnota ( Ukazatel ; Ukazatel { ; Ukazatel... } ) → Logické
    ` Jedinečná hodnota ( →Tabulka ; →Pole { ; →Pole2... } ) → Ano nebo Ne
C_BOOLEAN($0;$2)
C_POINTER($1)
C_LONGINT($vlPole;$vlNbPoles;$vlFound;$vlCurrentRecord)
$vlNbPoles:= Count parameters-1
$vlCurrentRecord:=Record number($1→)
If ($vlNbPoles>0)
    If ($vlCurrentRecord#-1)
        If ($vlCurrentRecord<0)
                ` Platný záznam je nový neuložený (číslo záznamu je -3)
                ` Proto můžeme zastavit hledání ve chvíli kdy je nalezen první záznam
            SET QUERY LIMIT(1)
        Else
                ` Platný záznam je existující záznam, proto můžeme zastavit hledání
                ` ve chvíli, kdy jsou nalezeny první dva záznamy
            SET QUERY LIMIT(2)
        End if
            ` Dotaz bude vracet výsledek do $vlFound
            ` bez změny platného výběru nebo záznamu
        SET QUERY DESTINATION(Into variable;$vlFound)
            ` Upravit hledání podle toho, kolik polí bylo specifikováno
        Case of
            ÷ ($vlNbPoles=1)
                QUERY($1→;$2→=$2→)
            ÷ ($vlNbPoles=2)
                QUERY($1→;$2→=$2→;*)
```

```
            QUERY($1→; & ;$3→=$3→)
        Else
            QUERY($1→;$2→=$2→;*)
            For ($vlPole;2;$vlNbPoles-1)
                QUERY($1→; & ;${1+$vlPole}→=${1+$vlPole}→;*)
            End for
            QUERY($1→; & ;$1+${vlNbPoles}→=${1+$vlNbPoles}→)
        End case
            ` Vrátit hledání do normálního módu
        SET QUERY DESTINATION(Into current selection)
        SET QUERY LIMIT(0) ` Již neomezovat hledání
            ` Zhodnotit výsledky hledání
        Case of
            ÷ ($vlFound=0)
                $0:=True ` Žádné dvojité hodnoty
            ÷ ($vlFound=1)
                If ($vlCurrentRecord<0)
                    ` Nalezen záznam se stejnou hodnotou
                    ` jakou má neuložený záznam
                    $0:=False
                Else
                    $0:=True ` Žádné duplikované hodnoty, pouze nalezen velmi podobný záznam
                End if
            ÷ ($vlFound=2)
                $0:=False ` Ať už se děje cokoli, jsou hodnoty dvojité
        End case
    Else
        If (<>DebugOn) ` Nic se neděje, signál je pouze ve vývojové verzi
            TRACE ` UPOZORNĚNÍ! Jedinečná hodnota je volána BEZ platného záznamu
        End if
        $0:=False ` Nemůže se zrušit výsledek
    End if
Else
    If (<>DebugOn) ` Nic se neděje, signál je pouze ve vývojové verzi
        TRACE ` UPOZORNĚNÍ! Jedinečná hodnota je volána BEZ podmínky dotazu
    End if
    $0:=False ` Nemůže se zaručit výsledek
End if
```

Po uložení této metody do vaší databáze můžete napsat:

```
` ...
If (Jedinečná hodnota (→[Kontakty];→[Kontakty]Firma;→[ Kontakty]Příjmení; → [Kontakty]Jméno)
    ` Provést akci pro záznamy které mají jedinečnou hodnotu
Else
    ALERT (Již existuje kontakt s názvem této firmy.")
End if
` ...
```

Dále si přečtěte

QUERY, QUERY BY EXAMPLE, QUERY BY FORMULA, QUERY SELECTION, QUERY SELECTION BY FORMULA, SET QUERY LIMIT.

Příkazy a odkazy pro Dotazy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET QUERY LIMIT (NASTAVIT OMEZENÍ DOTAZU)

Verze 6.0

SET QUERY LIMIT (limit)

Parametr	Typ		Popis
limit	Číslo	→	Počet záznamů, nebo 0 pro zrušení limitu dotazu

Popis

SET QUERY LIMIT vám umožní nastavit 4th Dimension aby zastavila hledání při nalezení tolika záznamů, kolik předáte do parametru limit.

Pokud například zadáte číslo 1, každé následující hledání se zastaví pokud najde jeden vyhovující záznam.

Aby jste navrátili hledání zpět do normálu, proveďte znovu přikaz SET QUERY LIMIT s parametrem limit 0.

Upozornění: SET QUERY LIMIT ovlivní všechna hledání provedená v jednom procesu. NEZAPOMEŇTE po provedení příkazu SET QUERY LIMIT (kde limit >0) opět provést příkaz SET QUERY LIMIT (0) pro zpětné nastavení módu hledání.
SET QUERY LIMIT ovlivňuje následující příkazy:
- QUERY
- QUERY SELECTION
- QUERY BY EXAMPLE
- QUERY BY FORMULA
- QUERY SELECTION BY FORMULA

Příkaz SET QUERY LIMIT nemění chování ostatních příkazů které mění platný výběr jako ALL RECORDS, RELATE MANY, atd.

Příklady

1. K provedení dotazu vyhovujicí požadavku "... najdi mi deset zakazníků, jejichž obrat je vetší než 1 000 000 ...." můžete napsat:

```
SET QUERY LIMIT (10)
QUERY ([Zákazníci];[ Zákazníci]Obrat>1000000)
SET QUERY LIMIT (0)
```

2. Podívejte se na příklad u příkazu SET QUERY DESTINATION.

Příkazy a odkazy pro Dotazy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Find index key (Nalézt v indexech pole)

Příkazy a odkazy pro Dotazy

Verze 6.5

Find index key (indexovanéPole; hodnota) → LongInt

Parametr	Typ		Popis
indexovanéPole	Pole	→	Indexované pole, pro které bude hledáno
hodnota		→	Hodnota k hledání
		←	Nalezená hodnota
Výsledek funkce	LongInt	→	Počet nalezených záznamů nebo -1 jestiže nebyl nalezen žádný záznam

Popis

Příkaz Find index key vrací číslo prvního záznamu jehož hodnota indexovaného pole vyhovuje zadané hodnotě. Pokud nebyl nalezen žádný záznam, je vráceno -1.

Po provedení příkazu obsahuje parametr hodnot nalezenou hodnotu. Tato vlastnost vám umožňuje provést na Alfa polích hledání pomocí znaku (@) a pak obdržet nalezenou hodnotu.

Tento příkaz nemění platný výběr nebo platný záznam.

Tento příkaz je velmi rychlý, protože používá indexy a je používán pro zabránění zdvojeného vstupu dat.

Příklad

Uvažujme, že během zadávání dat do databáze audio CD budete chtít ověřit, jestliže jméno zpěváka je již v databázi. Protože mohou vznikat homonyma, neuděláte pole [Zpěvák]Jméno jedinečné. Proto ve vstupním formuláři napíšte pro objekt pole [Zpěvák]Jméno tuto metodu:

```
If (Form event=On Data Change)
   $RecNum:=Find index key([Zpěvák]Jméno; [Zpěvák]Jméno)
   If ($RecNum # -1) ` Pokud bylo jméno nalezeno
      CONFIRM("Již existuje zpěvák s tímto jménem. Chcete vidět tento záznam?";"Ano";"Ne")
      If (OK=1)
         GOTO RECORD([Zpěvák];$RecNum)
      End if
   End if
End if
```

Příkazy a odkazy pro Dotazy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ORDER BY (TŘÍDIT DLE)

Příkazy a odkazy pro Dotazy

Verze 3

ORDER BY ({tabulka} {; pole} {; >nebo<} {; pole2; >nebo<2; ...; poleN; >nebo<N})

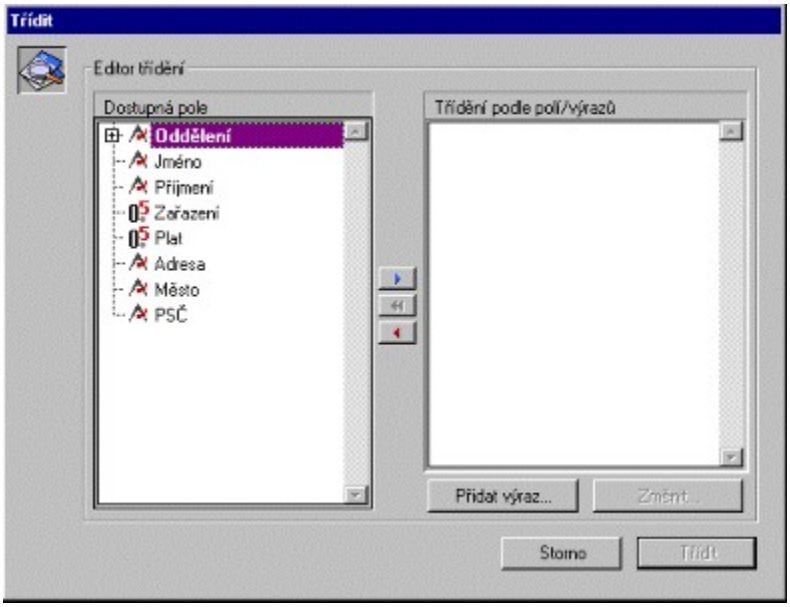
Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro kterou třídit výběr Výchozí tabulka pokud vynecháno
pole	Pole	→	Pole podle kterého třídit výběr
> nebo <			Směr třídění: > k třídění vzestupnému < k třídění sestupnému

Popis

Příkaz ORDER BY třídí platný výběr záznamů pro tabulku *tabulka* v platném procesu. Jakmile je třídění dokončeno, je první záznam ve výběru novým platným záznamem.

Pokud vynecháte parametr *tabulka*, je použita výchozí tabulka. Pokud není definována výchozí tabulka, je generována chyba.

Pokud nedefinujete parametry *pole* nebo > nebo <, příkaz zobrazí Editor třídění pro tabulku *tabulka*. Editor třídění v prostředí uživatele vypadá následovně:



Jestli chcete vědět více informací o Editoru třídění, přečtěte si *příručku uživatele 4th Dimension.*

Uživatel vytvoří seznam polí k třídění a klepne na tlačítko Třídit čímž třídění spustí. Pokud proběhne třídění bez přerušení, je proměnná OK nastavena na 1. Jestliže uživatel přeruší třídění v okně Editoru a třídění nebude provedeno, je proměnná OK nastavena na 0 (nula).

Příklady

1. Následující příklad zobrazí okno Editoru třídění pro tabulku [Produkty]:

```
ORDER BY ([Produkty])
```

2. Následující příklad zobrazí editor třídění pro výchozí tabulku (pokud je definovaná):

ORDER BY

Pokud definujete parametr *pole* a > nebo <, není zobrazen Editor třídění a třídění je definováno programově. Výběr můžete třídit na jednu úroveň nebo na více úrovní. Pro každou úroveň třídění definujte pole a pořadí třídění pomocí > nebo <. Pokud předáte symbol "větší než" (>), pořadí je vzestupné a pokud předáte "menší než" (<) je pořadí třídění sestupné.

Příklady

3. Následující řádek setřídí výběr tabulky [Produkty] podle jména ve vzestupném pořadí:

ORDER BY ([Produkty];[Produkty]Jméno;>)

4. Následující řádek setřídí výběr tabulky [Produkty] podle jména v sestupném pořadí:

ORDER BY ([Produkty];[Produkty]Jméno;<)

5. Následující řádek setřídí výběr tabulky [Produkty] podle polí typ a cena, obojí ve vzestupném pořadí:

ORDER BY ([Produkty];[Produkty]Typ;>;[Produkty]Cena;>)

6. Následující řádek setřídí výběr tabulky [Produkty] podle polí typ a cena, obojí v sestupném pořadí:

ORDER BY ([Produkty];[Produkty]Typ;<;[Produkty]Cena;<)

7. Následující řádek setřídí výběr tabulky [Produkty] podle pole typ ve vzestupném pořadí a podle pole cena v sestupném pořadí:

ORDER BY ([Produkty];[Produkty]Typ;>;[Produkty]Cena;<)

8. Následující řádek setřídí výběr tabulky [Produkty] podle pole typ v sestupném pořadí a podle pole cena ve vzestupném pořadí:

ORDER BY ([Produkty];[Produkty]Typ;<;[Produkty]Cena;>)

Pokud vynecháte parametr pro pořadí třídění, je jako výchozí použito třídění vzestupné:

9. Následující řádek setřídí výběr tabulky [Produkty] podle jména ve vzestupném pořadí:

ORDER BY ([Produkty];[Produkty]Jméno)

Pokud je vybráno pouze jedno pole pro třídění (jedna úroveň) a toto pole je indexované, je použit pro třídění index. Pokud toto pole není indexováno, nebo je použito více úrovní, probíhá třídění sekvenčně. Pole může patřit do tabulky *tabulka* definované k třídění, nebo k tabulce Jedniců vztažené k *tabulce* pomocí automatického vztahu. (Nezapomeňte že tabulka na kterou bude příkaz ORDER BY proveden musí být tabulkou Skupiny). V tomto případě je třídění vždy sekvenční (nepoužije indexy).

Příklad

10. Následující řádek provede indexované třídění pokud pole [Produkty]Jméno je indexované:

ORDER BY ([Produkty];[Produkty]Jméno;>)

11. Následující řádek provede postupné třídění, ať je pole indexované nebo není:

ORDER BY ([Produkty];[Produkty]Typ;>;[Produkty]Cena;>)

12. Následující řádek provede postupné třídění s použitím vztažených polí:

ORDER BY ([Faktury];[Firmy]Název;>) ` Faktury jsou setříděny abecedně podle názvu firmy.

Nezáleží na tom jak bylo třídění definováno, pokud je náročnější na čas, 4th Dimension automaticky zobrazí

teploměr který bude zobrazovat průběh. Tyto zprávy můžete vypnout nebo zapnout pomocí příkazů MESSAGES OFF a MESSAGES ON. Pokud je teploměr zobrazen, může uživatel klepnout na tlačítko Stop a třídění přerušit. Pokud je třídění správně dokončeno, je proměnná OK nastavena na 1, jinak je nastavena na 0.

Dále si přečtěte

ORDER BY FORMULA.

Příkazy a odkazy pro Dotazy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ORDER BY FORMULA (TŘÍDIT DLE VÝRAZU)

Verze 3

ORDER BY FORMULA ({tabulka} {; Výraz} {; >nebo<} {; Výraz2; >nebo<2; ...; VýrazN; >nebo<N})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro kterou třídit výběr Výchozí tabulka pokud vynecháno
Výraz		→	Výraz podle kterého třídit výběr (může být typu Alfa, Real, Integer, Long Integer, Datum, Čas nebo Logické)
> nebo <		→	Směr třídění: > k třídění vzestupnému < k třídění sestupnému

Popis

Příkaz ORDER BY třídí platný výběr záznamů pro tabulku v platném procesu. Jakmile je třídění dokončeno, je první záznam ve výběru novým platným záznamem.

Zde musíte definovat parametr tabulka. Nemůžete použít výchozí tabulku.

Výběr můžete třídit na jednu úroveň nebo na více úrovní. Pro každou úroveň třídění definujte výraz a pořadí třídění v > nebo <. Pokud předáte symbol "větší než" (>), pořadí je vzestupné a pokud předáte "menší než" (<) je pořadí třídění sestupné.

Parametr výraz může být typu Alfa, Real, Integer, Long Integer, Datum, Čas nebo Logické.

Nezáleží na tom jak bylo třídění definováno, pokud je náročnější na čas, 4th Dimension automaticky zobrazí teploměr který bude zobrazovat průběh. Tyto zprávy můžete vypnout nebo zapnout pomocí příkazů MESSAGES OFF a MESSAGES ON. Pokud je teploměr zobrazen, může uživatel klepnout na tlačítko Stop a třídění přerušit. Pokud je třídění správně dokončeno, je proměnná OK nastavena na 1, jinak je nastavena na 0.

4D Server: Server neprovádí výraz třídění. Každý záznam je poslán na stanici a výraz je proveden až na klientovi. To dělá tento příkaz na 4D Serveru o mnoho pomalejší než příkaz ORDER BY.

Na rozdíl od ORDER BY, ORDER BY FORMULA vždy provádí postupné třídění.

Příklad

Následující řádek setřídí záznamy v tabulce [Lidé] v sestupném pořadí založeném na délce příjmení každé osoby. Záznam s člověkem, který má nejdelší přijmení bude první v platném výběru.

```
ORDER BY FORMULA ([Lidé]; Length ([Lidé]Příjmení);<)
```

Dále si přečtěte

ORDER BY.

Zamykání záznamů

Příkazy a odkazy pro Zamykání záznamů

Verze 3

4th Dimension a 4D Server/4D Client automaticky řídí databázi aby nenastal konflikt několika uživatelů nebo procesů. Dva uživatelé nebo procesy nemohou najednou upravovat jeden záznam nebo objekt. Ale na druhou stránku si může druhý uživatel nebo proces záznam nebo objekt v tu samou chvíli prohlédnout.

Pro použití příkazů pro více uživatelů existuje mnoho důvodů:

- Upravování záznamů pomocí jazyka
- Použití vlastního rozhraní pro víceuživatelské akce
- Ukládání vztažených úprav do transakcí

Pro použití těchto příkazů ve více uživatelské databázi jsou tři základní pojmy:

- Každá tabulka je v módu číst nebo číst/psát
- Záznam se zamkne když je načten a odemkne když je uzavřen
- Zamčený záznam nemůže být upraven

Konvence v následující části je: osoba používající víceuživatelskou databázi je **místní uživatel**. Ostatní kdo používají databázi jsou **ostatní uživatelé**. Diskuze je prováděna z pohledu místního uživatele. Z pohledu víceprocesové aplikace, proces který provádí akce je **platný proces**. Všechny další zapnuté procesy jsou **další procesy**. Diskuze je prováděna z pohledu platného procesu.

Zamčené záznamy

Zamčený záznam nemůže být upraven místním uživatelem nebo platným procesem. Zamčený záznam může být načten, ale nemůže být upraven. Záznam se zamkne pokud nějaký z dalších uživatelů nebo procesů načte záznam a upravuje ho. Pouze uživatel který záznam zamknul jej může odemknout. Pro všechny ostatní uživatele s procesy je záznam zamčený a nemůže být upraven. Aby zamčený záznam byl odemčen, musí být tabulka v módu číst/psát.

Mód pouze číst a číst/psát

Každá tabulka v databázi může být v módu pouze číst nebo číst/psát pro všechny uživatele a procesy. **Pouze číst** znamená že záznamy mohou být načteny, ale nemohou být upraveny. **Číst/psát** znamená že záznamy mohou být načteny a upraveny a pro ostatní uživatele se tento záznam zamkne.

Pokud změníte mód tabulky, je tento mód použit na další načtený záznam. Pokud, ve chvíli kdy změníte mód tabulky, je načtený nějaký záznam, nebude mít nastavení vliv na tento záznam.

Mód Pouze číst

Pokud je tabulka pouze ke čtění a vy načtete záznam, tento záznam je stále uzamčený. Tento záznam může být tištěn, zobrazen a jinak s ním pracováno, ale nemůže být upraven.

Tento stav je použit pouze na již existující záznamy. Pokud předáte nový záznam, můžete jej měnit, ale po uložení se zamkne a nelze jej změnit. Záznamy můžete přidávat pomocí příkazů ADD RECORD, CREATE RECORD nebo

položkou Nový záznam v prostředí uživatele.

4th Dimension automaticky nastaví tento mód pro příkazy které nevyžadují přístup k záznamům. Jsou to tyto příkazy:

- DISPLAY SELECTION
- DISTINCT VALUES
- EXPORT DIF
- EXPORT SYLK
- EXPORT TEXT
- GRAPH TABLE
- PRINT SELECTION
- PRINT LABEL
- REPORT
- SELECTION TO ARRAY
- SELECTION RANGE TO ARRAY

Před použitím těchto příkazů, 4th Dimension uloží platný mód tabulky pro platný proces. Jakmile tento příkaz skončí je opět nastaven předchozí mód.

Mód Číst/psát

Pokud je tabulka v módu číst/psát a je načten záznam, tento záznam je odemčený, pokud jej nějaký uživatel neotevřel dříve. Pokud byl záznam zamčen jiným uživatelem, je záznam načtěný jako zamčený a nemůže být místním uživatelem upraven.

Tabulka musí být nastavena do módu číst/psát a načtený záznam musí být odemčený a tím pádem jej bude možno upravit.

Uživatel načte záznamy z tabulky, která je v módu číst/psát, a ostatní uživatelé nemohou tyto záznamy měnit.. Nicméně mohou ostatní uživatelé přidávat záznamy pomocí příkazů ADD RECORD nebo CREATE RECORD a pomocí položky Nový záznam v prostředí uživatele.

Mód číst/psát je výchozí nastavení každé tabulky při otevření nového procesu.

Změna módu tabulky

K upravení módu tabulky můžete použít příkazy READ WRITE a READ ONLY. Pokud chcete změnit mód tabulky podle toho jestli má být záznam zamčený nebo odemčený, použijte jeden z těchto příkazů před otevřením záznamu. Na záznam který je již načtený nebudou mít tyto příkazy vliv.

Každý proces má své vlastní nastavení tabulek.

Načítání, Upravení a Zavření záznamů

Před tím než místní uživatel bude moci upravit záznam, musí být tabulka v módu číst/psát a záznam musí být načtený jako odemčený.

Každý z příkazů které načítají záznam (pokud je nějaký) - jako NEXT RECORD, QUERY, ORDER BY, RELATE ONE, atd. - nastavují záznamy jako zamčený a odemčený.

Záznam je načten podle aktuálního módu tabulky a stavu záznamu. Záznam může být načten ze vztažené tabulky jedním z příkazů, které vytvářejí automatické vztahy.

Pokud je tabulka v módu pouze číst, pak je načtený záznam zamčený. Zamčený záznam nemůže být uložen nebo vymazán. Mód pouze číst je doporučovaný, protože umožňuje ostatním uživatelům záznam načíst,upravit a potom uložit.

Pokud je potřeba záznam upravit, můžete použít funkci Locked k testování jestli není zamčen jiným uživatelem nebo procesem. Pokud je záznam zamčen (Locked vrátí True) načtěte záznam pomocí LOAD RECORD a pak znovu otestujte jestli je záznam zamčený. Tato sekvence se musí opakovat, dokud nebude záznam odemčený.

Jakmile jste dokončili upravování záznamu, je potřeba,aby záznam byl uzavřen (a tím odemčen pro jiné uživatele) pomocí příkazu UNLOAD RECORD. Pokud záznam není uzavřen, bude i nadále uzamčen pro jiné uživatele, dokud nebude označen jiný platný záznam. Změna platného záznamu pro tabulku automaticky odemkne předchozí platný záznam pro ostatní uživatele. Pokud již nechcete upravovat platný záznam je doporučeno provést UNLOAD RECORD. Tato diskuze se týká pouze existujících záznamů. Pokud je vytvořen nový záznam, může být uložen ať už je mód tabulky jakýkoli. Aby jste zjistili který uživatel a/nebo proces uzamkl záznam, použijte příkaz LOCKED ATTRIBUTES.

Smyčky k načtení odemčených záznamů

Následující příklad ukazuje jednoduchou smyčku k načtení odemčeného záznamu:

```
READ WRITE ([Zákazníci]) ` Nastavit mód tabulky na číst/psát
Repeat ` Opakovat smyčku dokud nebude záznam odemčený
   LOAD RECORD ([Zákazníci]) ` Načíst záznam a uzamknout jej
Until (Not (Locked([Zákazníci])))
   ` Udělat něco se záznamem
READ ONLY ([Zákazníci]) ` Nastavit zpět mód tabulky na pouze číst
```

Smyčka bude pokračovat dokud nebude záznam odemčený.

Smyčky jako tahle se používají pouze jestliže je nepravděpodobné, že by záznam byl uzamčen někým jiným a je možné počkat dokud nebude smyčka splněna. Je nepravděpodobné, že tuto smyčku použijete, pokud není možné záznam upravit metodou.

Následující příklad používá předchozí smyčku k otevření nezamčeného záznamu a k jeho upravení:

```
READ WRITE([Inventář])
Repeat ` Opakovat smyčku dokud nebude záznam odemčený
   LOAD RECORD([Inventář]) ` Načíst záznam a uzamknout jej
Until (Not (Locked([Inventář])))
[Inventář]Part Qty := [Inventář]Part Qty _ 1 ` Upravit záznam
SAVE RECORD ([Inventář]) ` Uložit záznam
UNLOAD RECORD ([Inventář]) ` Umožnit ostatním uživatelům jej upravit
READ ONLY([Inventář])
```

Příkaz MODIFY RECORD automaticky upozorní uživatele jestli je záznam ,zamčený zabrání případným změnám záznamu. Následující příklad zruší tuto automatickou kontrolu tím, že nejdříve otestuje jestli není záznam zamčený pomocí funkce Locked. Pokud je záznam zamčený, může uživatel akci zrušit.

Následující příklad otestuje jestli je platný záznam pro tabulku [Příkazy] zamčený. Pokud je zamčený, pozdrží se proces o jednu sekundu. Tento způsob může být použit jak ve víceuživatelské tak v jednouživatelské databázi:

```
Repeat
   READ ONLY([Příkazy]) ` Nyní nepotřebujete mód číst/psát
   QUERY([Příkazy])
      ` Pokud bylo hledání dokončeno a byly nalezeny nějaké záznamy
   If ((OK=1) & (Records in selection([Příkazy])>0))
      READ WRITE([Příkazy]) ` Nastavit tabulku na číst/psát
```

```
LOAD RECORD([Příkazy])
While (Locked([Příkazy]) & (OK=1)) `Pokud je záznam zamčený,
      ` opakovat dokud nebude odemčený
      ` Kým je záznam zamčený?
   LOCKED ATTRIBUTES([Příkazy];$Process;$User; $Machine;$Name)
   If ($Process=-1) ` Byl záznam vymazán?
      ALERT("Záznam byl mezitím vymazán.")
      OK:=0
   Else
      If ($User="") ` Jste v jednouživatelské verzi
         $User:="vy"
      End if
      CONFIRM("Záznam používá "+$User+" v procesu "+$Name+".")
      If (OK=1) ` Pokud chcete chvíli počkat
         DELAY PROCESS( Current process;120) ` Počkat několik sekund
         LOAD RECORD([Příkazy])` Znovu zkusit záznam načíst
      End if
   End if
End while
If (OK=1) ` Záznam je odemčen
   MODIFY RECORD([Příkazy]) ` Můžete záznam upravit
   UNLOAD RECORD([Příkazy])
End if
READ ONLY([Příkazy]) ` Přepnout zpět na pouze číst
OK:=1
End if
Until (OK=0)
```

Použití příkazů v prostředí Více uživatelů a Více procesů

Některé příkazy v jazyce provádějí specifické akce pokud jsou použity na zamčené záznamy. Pokud nejsou použity na zamčené záznamy, chovají se normálně.

Zde je seznam těchto příkazů a jejich akcí při použití na zamčené záznamy.

- MODIFY RECORD: Zobrazí okno, že je záznam již používán. Záznam není zobrazen a uživatel nemůže záznam změnit. V prostředí uživatele je záznam ukázán pouze pro čtení.
- MODIFY SELECTION: Chová se normálně, dokud uživatel nepoklepe na záznam. MODIFY SELECTION zobrazí okno že je záznam již používán a zobrazí záznam pouze pro čtení.
- APPLY TO SELECTION: Načte zamčený záznam, ale neupraví jej. Tento příkaz může být použit pro načtení informací z tabulky bez dalších starostí. Pokud příkaz narazí na zamčený záznam, přesune jej do systémové sady LockedSet.
- DELETE SELECTION: Nevymaže jakýkoli zamčený záznam a přeskočí jej. Pokud příkaz narazí na zamčený záznam, přesune jej do systémové sady LockedSet.
- DELETE RECORD: Tento příkaz bude záznam ignorovat, pokud je zamčený. Nevznikne žádná chyba. Záznam musíte nejdříve testovat, jestli je zamčený a teprve pak jej vymazat.
- SAVE RECORD: Tento příkaz je ignorován, pokud je záznam zamčený. Nevznikne žádná chyba. Záznam musíte nejdříve testovat, jestli je zamčený a teprve pak jej uložit.
- ARRAY TO SELECTION: Neuloží žádný zamčený záznam. Pokud příkaz narazí na zamčený záznam, přesune jej do systémové sady LockedSet.
- GOTO RECORD: Záznamy ve víceprocesové/víceuživatelské databázi mohou být vymazány a předány jiným uživatelem. Proto se mohou čísla záznamů změnit. Použijte upozornění, pokud se ve víceuživatelské databázi odkazujete na záznamy pomocí čísel záznamů.

• Sady: Specielní opatrnost věnujte práci se sadami. Informace na kterých je založena sada mohou být změněny jinými uživateli.

Dále si přečtěte

LOAD RECORD, Locked, LOCKED ATTRIBUTES, Metody, READ ONLY, Read only state, READ WRITE, UNLOAD RECORD, Proměnné.

Příkazy a odkazy pro Zamykání záznamů

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

READ WRITE (ČÍST PSÁT)

Příkazy a odkazy pro Zamykání záznamů

Verze 3

READ WRITE ({tabulka | *})

Parametr	Typ		Popis
tabulka | *	Tabulka	→	Tabulka pro kterou nastavit mód číst/psát, nebo pokud * tak nastavit pro všechny tabulky. Pokud vynecháno, použít výchozí tabulku.

Popis

Příkaz READ WRITE změní pro proces ve kterém je spuštěn nastavení tabulky na číst/psát. Pokud je předáný parametr *, jsou nastaveny všechny tabulky na číst/psát.

Pokud provedete tento příkaz a pak nečtete nějaký záznam, bude tento záznam odemčený, jestliže již nebyl načten jiným uživatelem.Tento příkaz nemění stav záznamu který je právě načten, ale pouze záznamy které se načtou po jeho spuštění.

Výchozí mód všech tabulek je číst/psát.

Použijte příkaz READ WRITE v případě, že chcete záznam změnit a uložit. Také můžete použít tento příkaz pokud chcete zamknout záznam pro jiné uživatele, i když neděláte žádné změny.Nastavení tabulky na číst/psát, zabrání uživatelům upravovat tuto tabulku. Nicméně stále mohou přidávat další záznamy.

Poznámka: Tento příkaz nepůsobí zpětně. Záznam je načten podle aktuálního nastavení tabulky. K načtení záznamu v módu číst/psát z tabulky s módem pouze číst, musíte nejdříve změnit mód tabulky na číst/psát.

Dále si přečtěte

READ ONLY, READ ONLY STATE, Zamykání záznamů.

Příkazy a odkazy pro Zamykání záznamů

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

READ ONLY (POUZE ČÍST)

Příkazy a odkazy pro Zamykání záznamů

Verze 3

READ ONLY ({tabulka | *})

Parametr	Typ		Popis
tabulka | *	Tabulka	→	Tabulka pro kterou nastavit mód číst/psát, nebo pokud * tak nastavit pro všechny tabulky. Pokud vynecháno, použít výchozí tabulku.

Popis

Příkaz READ ONLY změní pro proces ve kterém je spuštěn nastavení tabulky na pouze číst. Všechny podzáznamy které budou načteny do záznamů této tabulky, budou zamčené a nebudete je moci měnit. Pokud je předáný parametr *, jsou nastaveny všechny tabulky na pouze číst.

Použijte příkaz READ WRITE v případě, že nechcete záznam(y) změnit.

Poznámka: Tento příkaz nepůsobí zpětně. Záznam je načten podle aktuálního nastavení tabulky. K načtení záznamu v módu číst/psát z tabulky s módem pouze číst, musíte nejdříve změnit mód tabulky na číst/psát.

Dále si přečtěte

Read only state, READ WRITE, Zamykání záznamů.

Příkazy a odkazy pro Zamykání záznamů

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Read only state (Stav pouze číst)

Příkazy a odkazy pro Zamykání záznamů

Verze 3

Read only state ({tabulka}) → Logické

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro kterou nastavit mód číst/psát, Pokud vynecháno, použít výchozí tabulku.
Výsledek funkce	Logické	→	Přístup k tabulce pouze číst (True) nebo Přístup k tabulce číst/psát (False)

Popis

Tato funkce je použita k testování jestli je nebo není tabulka pro proces ve kterém byl příkaz proveden v módu pouze číst. Pouze číst vrací True a pokud je nastavena na číst/psát tak funkce vrátí False.

Příklad

Následující příklad testuje stav tabulky [Faktury]. Pokud je stav tabulky pouze číst, je nastavena na číst/psát a je načten platný záznam.

```
If (Read only state ([Faktury]))
   READ WRITE([Faktury])
   LOAD RECORD([Faktury])
End if
```

Dále si přečtěte

READ ONLY, READ WRITE, Zamykání záznamů.

Příkazy a odkazy pro Zamykání záznamů

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

LOAD RECORD (ZAVÉST ZÁZNAM)

Příkazy a odkazy pro Zamykání záznamů

Verze 3

LOAD RECORD ({tabulka})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro kterou nastavit mód číst/psát, Pokud vynecháno, použít výchozí tabulku.

Popis

LOAD RECORD načte platný záznam tabulky. Pokud není žádný platný záznam, příkaz neprovede nic.

Pak můžete použít funkci Locked, jestli můžete záznam měnit:

• Pokud je tabulka v módu pouze číst, vrátí Locked True a záznam nemůžete měnit.
• Pokud je tabulka v módu číst/psát, ale záznam je zamčený, načte se záznam pouze pro čtení a vy jej nemůžete měnit.
• Pokud je tabulka v módu číst/psát a záznam není zamčený, můžete jej změnit. Funkce Locked bude vracet True pro všechny ostatní procesy a uživatele.

Poznámka: Pokud je příkaz LOADRECORD proveden po READ ONLY, záznam je automaticky zrušen a načten bez použití příkazu UNLOAD RECORD.

Obvykle nebudete potřebovat používat příkaz LOAD RECORD, protože příkazy jako QUERY, NEXT RECORD, PREVIOUS RECORD, atd., automaticky načtou platný záznam.

Pokud potřebujete upravit existující záznam v prostředí víceuživatelů nebo víceprocesů, potřebujete přístup k tabulce nastavený na číst/psát. Pokud je záznam zamčený a nenačtený, LOAD RECORD vám umožní zkusit záznam načíst později znovu. Při použití LOAD RECORD ve smyčce, můžete čekat dokud se záznam neuvolní do módu číst/psát.

Dále si přečtěte

Locked, Zamykání záznamů, UNLOAD RECORD.

Příkazy a odkazy pro Zamykání záznamů

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

UNLOAD RECORD (VYVÉST ZÁZNAM)

Verze 3

UNLOAD RECORD ({tabulka})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro kterou chceme vyvést záznam z paměti, Pokud vynecháno, použít výchozí tabulku.

Popis

Příkaz UNLOAD RECORD vyvede platný záznam tabulky z paměti.

Pokud je záznam odemčený pro místního uživatele (zamčený pro ostatní uživatele), UNLOAD RECORD odemkne záznam pro ostatní uživatele.

Ačkoli UNLOAD RECORD odečte záznam z paměti, tento záznam zůstane platným záznamem. Jakmile se změní platný záznam, je předchozí automaticky odstraněn z paměti a odemčen pro ostatní uživatele. Použijte tento příkaz pokud jste skončili se změnami v záznamu a chcete jej odemknout pro ostatní uživatele, ale chcete aby tento záznam zůstal platným záznamem tabulky.

Pokud obrázek obsahuje velká data, jako obrázková pole nebo externí dokumenty (třeba dokumenty 4D Write nebo 4D Draw), nemusíte chtít záznam nechávat v paměti pokud jej již nechcete měnit. V tomto případě použijte UNLOAD RECORD aby jste ponechali platný záznam ale odstranili jej z paměti. Uvolníte paměť, kterou záznam zabíral, ale již nemáte přístup k hodnotám jeho polí. Pokud budete později potřebovat přístup k polím, použijte příkaz LOAD RECORD.

Dále si přečtěte

LOAD RECORD, Zamykání záznamů.

Příkazy a odkazy pro Zamykání záznamů

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Locked (Zamčeno)

Příkazy a odkazy pro Zamykání záznamů

Verze 3

Locked ({tabulka}) → Logické

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro kterou zjistit zamčení, Pokud vynecháno, použít výchozí tabulku.
Výsledek funkce	Logické	←	Záznam je uzamčen (True) nebo Záznam je odemčen (False)

Popis

Locked testuje jestli je nebo není platný záznam pro tabulku zamčený. Použijte tuto funkci k testování jestli je záznam zamčený; podle toho proveďte patřičnou akci, jako třeba dejte uživateli na vybranou jestli počká dokud se záznam neodemkne.

Pokud Locked vrátí True, je záznam zamčen jiným uživatelem nebo procesem a nemůže být uložen. V tomto případě použijte příkaz LOAD k opětovnému načtení záznamu dokud Locked nevrátí False.

Pokud Locked vrátí False, znamená to že je záznam odemčený pro mísntího uživatele a zamčený pro všechny ostatní uživatele. Tabulka musí být v módu číst/psát aby bylo možno záznam měnit.

Pokud se pokusíte načíst záznam, který byl již vymazán, bude Locked stále vracet True. Aby jste zabránili čekání na záznam který již neexistuje, použijte příkaz LOCKED ATTRIBUTES. Pokud byl záznam vymazán, příkaz LOCKED ATTRIBUTES vrátí 1 do čísla procesu.

Během pokusů LOAD RECORD načíst záznam, bude funkce Locked testovat dostupnost záznamu. Pokud je záznam zamčený, je možné akci zrušit.

Dále si přečtěte

LOAD RECORD, LOCKED ATTRIBUTES, Zamykání záznamů.

Příkazy a odkazy pro Zamykání záznamů

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

LOCKED ATTRIBUTES (VLASTNOSTI UZAMČENÍ)

Verze 3

LOCKED ATTRIBUTES ({tabulka;} proces; uživatel; stroj; NázevProcesu)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka, pro kterou testujeme uzamčený platný záznam. Pokud vynecháno, použít výchozí tabulku.
proces	Číslo	←	Číslo odkazu na proces
uživatel	Řetězec	←	Jméno uživatele ve víceuživatelské verzi
stroj	Řetězec	←	Název počítače ve víceuživatelské verzi
NázevProcesu	Řetězec	←	Název procesu

Popis

LOCKED ATTRIBUTES vrací informace o uživateli a procesu který záznam uzamkl. Číslo procesu, jméno uživatele, název stroje a název procesu jsou vráceny do proměnných proces, uživatel, stroj a NázevProcesu. Tyto informace můžete použít ve vlastním okně upozornění k informování uživatele o tom, kdo záznam uzamkl.

Pokud záznam není uzamčen, do parametru proces se dosadí 0 a proměnné uživatel, stroj a NázevProcesu budou prázdné řetězce. Pokud byl záznam, který se pokoušíte zjistit, vymazán, v proměnné proces bude -1 a ostatní budou prázdné řetězce.

V jednouživatelské databázi se doplní hodnoty do proces a NázevProcesu pouze v případě, že je záznam uzamčen. Hodnoty v uživatel a stroj budou prázdné řetězce.

V architektuře klient/server bude číslo dosazené do proměnné proces číslo procesu na serveru.

Do proměnné uživatel se dosadí jméno uživatele z editoru hesel 4th Dimension. Pokud není žádný systém hesel, je vráceno "Manager".

V proměnné stroj bude vlastník počítače ze systému. Pokud toto jméno změníte, projeví se to až po restartu.

Dále si přečtěte

Locked, Zamykání záznamů.

Příkazy a odkazy pro Zamykání záznamů

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DISPLAY RECORD (ZOBRAZIT ZÁZNAM)

Příkazy a odkazy pro Záznamy

Verze 3

DISPLAY RECORD ({tabulka})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ze které zobrazit záznam, Pokud vynecháno, použít výchozí tabulku.

Popis

Příkaz DISPLAY RECORD zobrazí platný záznam tabulky v platném vstupním formuláři. Záznam je zobrazen pouze dokud nějaká událost nepřekreslí okno. Jako příklad můžete provést ADD RECORD, vrácející se do vstupního formuláře nebo do záhlaví nabídek. DISPLAY RECORD neprovede nic pokud nemáte vybraný řádný platný záznam.

DISPLAY RECORD se používá také k zobrazení vlastních posuvných zpráv. Může být také využit pro vytvoření vlastních běžících obrázkových předvedení.

Pokud existuje metoda formuláře, je aktivována událost Při zavedení (On Load).

UPOZORNĚNÍ: Nevolejte příkaz DISPLAY RECORD v procesu Web spojení, protože okno bude zobrazeno na stroji 4th Dimension a ne na Web prohlížeči.

Příklad

Následující příklad postupně zobrazí sérii záznamů:

```
ALL RECORDS([Demo]) ` Vybrat všechny záznamy
INPUT FORM ([Demo]; "Display") ` Nastavit formulář k zobrazení
For ($vlZaznam;1;Records in selection([Demo])) ` Smyčka přes všechny záznamy
   DISPLAY RECORD([Demo]) ` Zobrazit záznam
   DELAY PROCESS (Current process; 180) ` Počkat 3 sekundy
   NEXT RECORD([Demo]) ` Přesunout na další záznam
End for
```

Dále si přečtěte

MESSAGE.

Příkazy a odkazy pro Záznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CREATE RECORD (VYTVOŘIT ZÁZNAM)

Příkazy a odkazy pro Záznamy

Verze 3

CREATE RECORD ({tabulka})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ze které zobrazit záznam, Pokud vynecháno, použít výchozí tabulku.

Popis

CREATE RECORD vytvoří nový záznam pro tabulku, ale nezobrazí jej. K vytvoření a zobrazení záznamu použijte ADD RECORD.

Příkaz CREATE RECORD se používá v případě že data do nového záznamu budete přiřazovat pomocí Jazyka. Nový záznam se stane platným záznamem a platným výběrem (výběr o jednom záznamu).

Tento záznam exituje pouze v paměti, dokud neprovedete SAVE RECORD. Pokud se změní platný záznam (například během dotazu) před tím, než nový uložíte, bude nový záznam ztracen.

Příklad

Následující příklad archivuje záznamy, které jsou starší než 30 dní. Provede to vytvořením nového záznamu v tabulce Archivu. Jakmile je archivace dokončena, jsou uložené záznamy vymazány z tabulky [Account]:

```
   ` Najít záznamy starší než 30 dní
QUERY ([Accounts]; [Accounts]Entered < (Current date - 30))
For ($vlZaznam;1; Records in selection([Accounts])) ` Jedna smyčka pro každý záznam
   CREATE RECORD ([Archive]) ` Vytvořit nový záznam archivu
   [Archive]Number:=[Account]Number ` Kopírovat pole do záznamu archivu
   [Archive]Entered:=[Account]Entered
   [Archive]Amount:=[Account]Amount
   SAVE RECORD([Archive]) ` Uložit záznam
   NEXT RECORD([Accounts]) ` Přesunout se na další záznam
End for
DELETE SELECTION([Accounts]) ` Vymazat záznamy
```

Dále si přečtěte

SAVE RECORD.

Příkazy a odkazy pro Záznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DUPLICATE RECORD (DUPLIKOVAT ZÁZNAM)

Verze 3

DUPLICATE RECORD ({tabulka})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ze které duplikovat platný záznam, Pokud vynecháno, použít výchozí tabulku.

Popis

Příkaz DUPLICATE RECORD vytvoří v tabulce nový záznam, který je kopií platného záznamu. Nový záznam bude platný záznam. Pokud neexistuje žádný platný záznam, příkaz neprovede nic. K uložení tohoto záznamu musíte použít příkaz SAVE RECORD.

Příkaz DUPLICATE RECORD může být proveden během vstupu dat. Toto vám umožní vytvořit "klon" upravovaného záznamu. Nezapomeňte že musíte nejdříve provést SAVE RECORD k uložení změn v záznamu.

Dále si přečtěte

SAVE RECORD.

Příkazy a odkazy pro Záznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Is new record (Je novým záznamem)

Příkazy a odkazy pro Záznamy

Verze 6.5

Is new record ({tabulka}) → Logické

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka záznamu který se má prověřit, Pokud vynecháno, použít výchozí tabulku.
Výsledek funkce	Logické	←	True pokud byl záznam vytvořen jinak False

Popis

Příkaz Is new record vrací True pokud platný záznam pro tabulku byl právě vytvořen a ještě nebyl uložen.

Poznámka kompatibility: To samé můžete zjistit s použitím příkazu Record number a testováním na číslo -3. Nicméně vám silně doporučujeme aby jste používali příkaz Is new record, protože je kompatibilní s budoucími verzemi 4th Dimension.

Příkad

Následující dvě instrukce jsou stejné. Druhý způsob je doporučený, protože je kompatibilní s budoucími verzemi 4D:

```
If (Record number([Tabulka])=-3) `Nedoporučeno
   ` ...
End if

If (Is new record([Tabulka])) `Silně doporučeno
   ` ...
End if
```

Dále si přečtěte

Modified record, Record number.

Příkazy a odkazy pro Záznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Modified record (Záznam upraven)

Příkazy a odkazy pro Záznamy

Verze 3

Modified record ({tabulka}) → Logické

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka záznamu který se má prověřit, Pokud vynecháno, použít výchozí tabulku.
Výsledek funkce	Logické	←	True pokud byl záznam upraven jinak False

Popis

Modified record vrací True pokud byl platný záznam upraven, ale neuložen; jinak vrací False. Tato možnost dovoluje návrháři snadno otestovat, jestli byl záznam změněn a je třeba jej uložit. Je to zejména použitelné ve vstupním formuláři k testování jestli uložit nebo neuložit záznam před přechodem na jiný. Tato funkce vždy vrací True na novém záznamu.

Příklad

Následující příklad ukazuje typické použití funkce Modified record:

```
If (Modified record ([Zákazníci]))
   SAVE RECORD ([Zákazníci])
End if
```

Dále si přečtěte

Modified, Old, SAVE RECORD.

Příkazy a odkazy pro Záznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Is record loaded (Je záznam zaveden)

Příkazy a odkazy pro Záznamy

Verze 6.5

Is record loaded ({tabulka}) → Logické

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka záznamu který se má prověřit, Pokud vynecháno, použít výchozí tabulku.
Výsledek funkce	Logické	←	True pokud je záznam nečtený, jinak False

Popis

Příkaz Is record loaded vrací True pokud je platný záznam tabulky načten v platném procesu.

Příklad

Místo použití automatických akcí "Další záznam" nebo "Předchozí záznam", můžete napsat metodu objektu pro tato tlačítka k určení jejich akce. Tlačítko "Další" zobrazí začátek výběru pokud je uživatel na konci výběru a tlačítko "předchozí" zobrazí konec výběru pokud je uživatel na začátku výběru.

```
   ` Metoda obejktu pro tlačítko "Předchozí" (bez automatické akce)
If (Form event=On Clicked)
   PREVIOUS RECORD([Group])
   If (Not(Is record loaded([Group])))
      GOTO SELECTED RECORD([Group];Records in selection([Group]))
         `Jít na poslední záznam ve výběru
   End if
End if

   ` Metoda objektu tlačítka "Další" (bez automatické akce)
If (Form event=On Clicked)
   NEXT RECORD([Group])
   If (Not(Is record loaded([Group])))
      GOTO SELECTED RECORD([Groups];1)
         `Jít na první záznam ve výběru
   End if
End if
```

Příkazy a odkazy pro Záznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SAVE RECORD (ULOŽIT ZÁZNAM)

Příkazy a odkazy pro Záznamy

Verze 3

SAVE RECORD ({tabulka})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka záznamu který se má uložit, Pokud vynecháno, použít výchozí tabulku

Popis

Příkaz SAVE RECORD uloží platný záznam tabulky v platném procesu. Pokud není žádný platný záznam, je příkaz SAVE RECORD ignorován.

Tento příkaz používáte k uložení záznamu který jste vytvořili nebo upravili pomocí kódu. Záznam který byl upraven a potvrzen uživatelem ve formuláři, nepotřebuje ukládat pomocí příkazu SAVE RECORD. Záznam který byl upraven uživatelem a pak zrušen, může být stále uložen pomocí SAVE RECORD.

Zde jsou některé příklady kdy je SAVE RECORD vyžadován:
- K uložení nového záznamu vytvořeného pomocí CREATE RECORD a DUPLICATE RECORD
- K uložení dat z RECEIVE RECORD
- K uložení záznamu upraveného metodou
- K uložení záznamu který obsahuje nový nebo upravený podzáznam vytvořený pomocí příkazů ADD SUBRECORD, CREATE SUBRECORD nebo MODIFY SUBRECORD
- Během vstupu dat k uložení zobrazeného záznamu před použitím příkazu který změní platný záznam
- Během vstupu dat k uložení platného záznamu

Neprovádějte příkaz SAVE RECORD v metodě formuláře během události Při potvrzení vstupu když byl záznam přijatý. Pokud to uděláte, záznam bude uložený dvakrát.

Příklady

Následující příklad je část metody, která čte záznamy z dokumentu. Tato část načte záznam a pokud je načten správně, je uložen:

```
RECEIVE RECORD ([Zákazníci]) ` Načíst záznam z disku
If (OK= 1) `Pokud je správně načten…
   SAVE RECORD ([Zákazníci]) ` uložit
End if
```

Dále si přečtěte

CREATE RECORD, Locked, Triggery.

DELETE RECORD (VYMAZAT ZÁZNAM)

Příkazy a odkazy pro Záznamy

Verze 3

DELETE RECORD ({tabulka})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka záznamu který se má vymazat, Pokud vynecháno, použít výchozí tabulku

Popis

Příkaz DELETE RECORD vymaže platný záznam tabulky v procesu. Pokud není žádný patný záznam, příkaz neprovede nic. Ve formuláři můžete vytvořit tlačítko vymazat,bez použití tohoto příkazu. Po vymazání záznamu je platný výběr pro tabulku prázdný.

Vymazání záznamu je trvalá operace a nelze ji vrátit.

Jakmile je záznam vymazaný, jeho číslo záznamu bude znovu použito při vytvoření nového záznamu. Pokud z databáze mažete záznamy, tak vám nedoporučujeme používat čísla záznamů pro identifikaci záznamů.

Příklad

Následující příklad vymaže záznam zaměstnance. Metoda se zeptá uživatele na ID zaměstnance,který se má vymazat,nalezne jej a pak jej vymaže:

```
vFind := Request ("ID Zaměstnance k vymazání:") ` Získat ID zaměstnance
If (OK = 1)
   QUERY ([Zaměstnanci]; [Zaměstnanci]ID = vFind) ` Nalézt zaměstnance
   DELETE RECORD ([Zaměstnanci]) ` Vymazat zaměstnance
End if
```

Dále si přečtěte

Locked, Triggery.

Příkazy a odkazy pro Záznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Records in table (Záznamů v tabulce)

Příkazy a odkazy pro Záznamy

Verze 3

Records in table ({tabulka}) → Číslo

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro kterou vrátit počet záznamů, Pokud vynecháno, použít výchozí tabulku
Výsledek funkce	Číslo	←	Celkový součet záznamů v tabulce

Popis

Records in table vrací celkový počet záznamů v tabulce. Příkaz Records in selection vrací pouze počet záznamů v platném výběru. Pokud je Records in table použit v transakci, jsou záznamy vytvořené v transakci zahrnuté do součtu.

Příklad

Následující příklad zobrazí upozornění které ukáže počet záznamů v tabulce:

```
ALERT ("V tabulce je " +String(Records in table([Lidé]))+" záznamů.")
```

Dále si přečtěte

Records in selection.

Příkazy a odkazy pro Záznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Record number (Číslo záznamu)

Příkazy a odkazy pro Záznamy

Verze 3

Record number ({tabulka}) → Číslo

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka , pro kterou chceme číslo (pořadí) platného záznamu v tabulce, nebo pokud vynecháno, použít výchozí tabulku
Výsledek funkce	Číslo	←	Číslo platného záznamu

Popis

Record number vrací číslo záznamu pro platný záznam tabulky. Pokud není žádný platný záznam, jako třeba že je ukazatel na záznam před nebo za platným výběrem, vrátí Record number -1. Jestliže je záznam nový a ještě nebyl uložen, Record number vrací -3.

Čísla záznamů se mohou měnit. Čísla vymazaných záznamů jsou znovu použita. Mohou se také změnit pokud provedete kompaktaci nebo obnovení podle tagů pomocí nástroje 4D Tools. Při vytváření záznamů během transakce jsou záznamům přiřazena dočasná čísla. Jakmile je transakce potvrzena, je záznamu přiřazeno stálé číslo.

Příklad

Následující příklad uloží číslo platného záznamu a pak vyhledá záznamy se stejnými daty:

```
$RecNum:=Record number([Lidé]) ` Získat číslo záznamu
QUERY ([Lidé]; [Lidé]Příjmení = [Lidé] Příjmení) ` Existuje již stejné příjmení?
   ` Zobrazit upozornění s počtem lidí se stejným příjmením
ALERT ("Existuje "+String (Records in selection([Lidé])+" lidí se stejným příjmením.")
GOTO RECORD ([Lidé]; $RecNum) ` Jít zpět na stejný záznam
```

Dále si přečtěte

O číslech záznamů, GOTO RECORD, Is new record, Selected record number, Sequence number.

Příkazy a odkazy pro Záznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GOTO RECORD (JÍT NA ZÁZNAM)

Verze 3

GOTO RECORD ({tabulka;} záznam)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka se záznamem na který jít, Pokud vynecháno, použít výchozí tabulku
záznam	Číslo	→	Číslo vrácené Record number

Popis

GOTO RECORD označí specifický záznam jako platný záznam tabulky. Parametr záznam je číslo záznamu vrácené funkcí Record number. Po provedení tohoto příkazu je tento záznam jediným ve výběru záznamů.

Pokud zadané číslo bude menší než nejnižší číslo záznamu, nebo bude větší než nejvyšší číslo záznamu, 4D First generuje chybu a zobrací okno s upozorněním, že číslo záznamu je mimo rozsah. Pokud je číslo záznamu čílso vymazaného záznamu, bude výběr prázdný.

Poznámka: S tímto příkazem nemůžete používat dočasná čísla záznamů vytvářená během transakce.

Příklad

Přečtěte si příklad u příkazu Record number.

Dále si přečtěte

O číslech záznamů, Record number.

Příkazy a odkazy pro Záznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Sequence number (Pořadové číslo)

Příkazy a odkazy pro Záznamy

Verze 3

Sequence number ({tabulka}) → Číslo

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka, pro kterou chceme následující pořadové číslo, Pokud vynecháno, použít výchozí tabulku
Výsledek funkce	Číslo	←	Následující pořadové číslo

Popis

Sequence number vrací další pořadové číslo pro tabulku. Pořadové číslo je jedinečné pro každou tabulku. Je to neopakovatelné číslo které vzroste pro každý nový záznam v tabulce. Začíná od čísla 1.

Příkaz Sequence number můžete použít místo symbolu #N jestliže:

- pokud vytváříte záznamy metodami
- Pořadové číslo musí začít jinde než od 1
- Pořadové číslo musí vzrůst o více než 1
- Pořadové číslo je část kódu, například část kódu čísla

K uložení přadového čísla prostřednictvím metody, vytvořte pole Long Integer v tabulce a uložte do něj pořadové číslo.

Pořadové číslo je stejné číslo jaké bude přiřazeno pokud do kolonky Výchozí metoda ve vlastnostech objektu předáte symbol #N. Jestli chcete vědět více informací o přiřazení výchozí hodnoty, přečtěte si *Příručku návrháře 4th Dimension.*

Pokud musí pořadové číslo začít jiným číslem než 1, přidejte rozdíl za Sequence number. Pokud musí začít třeba číslem 1000, můžete použít následující řádek:

```
[Tabulka1]Poř_Číslo:=Sequence number ([Tabulka1]) + 999
```

Příklad

Následující příklad je částí metody formuláře. Testuje zda je tento záznam nový; to znamená že pole Číslo faktury je prázdné. Pokud je to nový záznam, metoda přiřadí číslo faktury. Číslo faktury je složeno ze dvou částí: pořadové číslo a ID operátoru, které bylo předánno při otevření databáze. Pořadové číslo je formátováno jako 5-ti místný řetězec.

```
   ` Pokud je tento záznam nový, vytvořit nové pořadové číslo
If ([Faktury]ČísloFaktury = "")
      ` Toto pořadové číslo je řetězec končící ID operátora.
   [Faktury]ČísloFaktury:=String(Sequence number;"00000")+[Faktury]OpID
End if
```

Dále si přečtěte

O číslech záznamů, Record number, Selected record number.

Příkazy a odkazy pro Záznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

O číslech záznamů

Příkazy a odkazy pro Záznamy

Verze 3

K záznamům jsou přiřazena tři čísla:

- Číslo záznamu
- Číslo záznamu ve výběru
- Pořadové číslo

Číslo záznamu

Číslo záznamu je fyzické číslo záznamu. Toto číslo je k záznamu přiřazeno ve chvíli kdy je vytvořen a patří mu dokud není záznam smazán nebo dokud nezměníte trvalé třídění pomocí 4D Tools. Čísla záznamů začínají od nuly. Nejsou jedinečná, protože číslo vymazaného záznamu je znovu použito pro nový záznam. Také se čísla mění pokud třídíte data pomocí 4D Tools nebo jsou data kompaktována. Novým záznamům vytvořeným během transakce jsou přiřazena dočasná čísla. Konečná čísla jim jsou přiřazena až po potvrzení transakce.

Číslo záznamu ve výběru

Číslo záznamu ve výběru udává pozici záznamu v platném výběru a proto záleží na platném výběru. Pokud je výběr změněn nebo přetříděn, změní se i toto číslo. Číslování záznamů ve výběru začíná od 1.

Pořadové číslo

Pořadové číslo je jedinečné neopakující se číslo, které může být přiřazeno záznamu nebo poli. Není automaticky ukládáno s každým záznamem. Začíná od 1 a roste s každým přidaným záznamem. Narozdíl od čísla záznamu není pořadové číslo použito znovu když je záznam vymazán nebo jsou data přetříděna nebo kompaktována pomocí 4D Tools. Pořadová čísla jsou způsob jak mít jedinečné ID každého záznamu. Jestliže pořadové číslo vzrůstá během transakce, tak se vrátí na původní hodnotu po zrušení transakce.

Příklady čísel záznamů

Následující tabulky ukazují čísla která jsou k záznamům přiřazena.. Každý řádek v tabulce ukazuje informace o záznamu. Pořadí řádků je pořadí ve které by byly záznamy zobrazeny ve výstupním formuláři.

- **Sloupec data**: Data z pole každého záznamu. Pro náš příklad jsou použita jména osob.
- **Sloupec Číslo záznamu**: Fyzické číslo záznamu. Je to číslo které vrátí funkce Record number.
- **Sloupec Číslo záznamu ve výběru**: Pozice záznamu v platném výběru. Je to číslo které vrací funkce Selected record number.
- **Sloupec Pořadové číslo**: Jedinečné pořadové číslo záznamu. Je to číslo vracené funkcí Sequence number při vytvoření záznamu. Toto číslo je uloženo v poli.

Po předání záznamů

První tabulka ukazuje záznamy po předání.

- Výchozí pořadí záznamů je v čísle záznamu
- Číslo záznamu začíná od 0
- Číslo záznamu ve výběru a pořadové číslo začínají od 1

Data	*Číslo záznamu*	*Číslo záznamu ve výběru*	*Pořadové číslo*
Tereza	*0*	*1*	*1*

Tomáš	*1*	*2*	*2*
Simona	*2*	*3*	*3*
Štěpán	*3*	*4*	*4*
Lenka	*4*	*5*	*5*

Poznámka: Záznamy zůstanou ve výchozím pořadí pokud uživatel změní platný výběr bez jeho přetřídění; například po vybrání položky nabídky Všechny záznamy v prostředí uživatele nebo po provedení příkazu ALL RECORDS.

Po přetřídění záznamů

Následující tabulka ukazuje stejné záznamy setříděné podle jména.

• Stejné číslo záznamu je přiřazeno ke každému záznamu
• Číslo záznamu ve výběru odráží pozici záznamu ve tříděném výběru
• Pořadové číslo se nemění od chvíle kdy byl záznam vytvořen a číslo uloženo v poli

Data	***Číslo záznamu***	***Číslo záznamu ve výběru***	***Pořadové číslo***
Lenka	*4*	*1*	*5*
Simona	*2*	*2*	*3*
Štěpán	*3*	*3*	*4*
Tereza	*0*	*4*	*1*
Tomáš	*1*	*5*	*2*

Po vymazání záznamu

Následující tabulka ukazuje záznamy po vymazání záznamu Štěpán.

• Změní se pouze číslo záznamu ve výběru. Toto číslo odráží pozici záznamu ve výběru.

Data	***Číslo záznamu***	***Číslo záznamu ve výběru***	***Pořadové číslo***
Lenka	*4*	*1*	*5*
Simona	*2*	*2*	*3*
Tereza	*0*	*4*	*1*
Tomáš	*1*	*5*	*2*

Po předání nového záznamu

Následující tabulka ukazuje záznamy po přidání nového záznamu Lukáš.

• Nový záznam je přidán na konec platného výběru
• Číslo záznamu po Štěpánovi je použito znovu
• Pořadové číslo bude vzrůstat

Data	***Číslo záznamu***	***Číslo záznamu ve výběru***	***Pořadové číslo***
Tereza	*0*	*1*	*1*
Tomáš	*1*	*2*	*2*
Simona	*2*	*3*	*3*
Lenka	*4*	*4*	*5*
Lukáš	*3*	*5*	*6*

Po změně výběru a setřídění

Následující tabulka ukazuje výběr omezený na tři záznamy a pak setříděný.

• Změní se pouze číslo záznamu ve výběru

Data	***Číslo záznamu***	***Číslo záznamu ve výběru***	***Pořadové číslo***

Lenka	*4*	*1*	*5*
Simona	*2*	*2*	*3*
Tomáš	*1*	*3*	*2*

Dále si přečtěte

Record number, Selected record number, Sequence number.

Příkazy a odkazy pro Záznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

PUSH RECORD (VSUNOUT ZÁZNAM)

Verze 3

PUSH RECORD ({tabulka})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka, pro kterou chceme záznam vsunout do paměti, Pokud vynecháno, použít výchozí tabulku

Popis

PUSH RECORD vsune záznam do paměti (i s podzáznamy, pokud nějaké má) do stack záznamu tabulky. PUSH RECORD může být použito před uložením záznamu.

Pokud do paměti vsunete záznam, který byl odemčený, tento záznam zůstane pro ostatní uživatele a procesy zamčený, dokud jej z paměti nevezmete zpět a nevyvedete je pomocí UNLOAD RECORD.

Příklad

Následující příklad vsune do paměti záznam tabulky [Zákaznící]:

```
PUSH RECORD ([Zákazníci]) ` Vsunout záznam zákazníka do paměti
```

Dále si přečtěte

POP RECORD, Používání paměti Stack pro záznam.

Příkazy a odkazy pro Záznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

POP RECORD (VYSUNOUT ZÁZNAM)

Verze 3

POP RECORD ({tabulka})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka, pro kterou chceme záznam vysunout z paměti, Pokud vynecháno, použít výchozí tabulku

Popis

POP RECORD vysune záznam z paměti a udělá z něj platný záznam.

Pokud vsunete záznam, změníte výběr aby neobsahoval vsunutý záznam a pak vysunete záznam, platný záznam není v platném výběru. K označení vysunutého záznamu jako platný výběr, použijte příkaz ONE RECORD SELECT. Jestliže použijete příkazy které pohybují záznamy před uložením záznamu, ztratíte tuto kopii z paměti.

Příklad

Následující příklad vysune záznam zákazníka z paměti:

```
POP RECORD ([Zákazníci]) ` Vysune záznam zákazníka z paměti
```

Dále si přečtěte

PUSH RECORD, Používání paměti Stack pro záznam.

Příkazy a odkazy pro Záznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Používání paměti Stack pro záznam

Verze 3

Příkazy PUSH RECORD a POP RECORD vám umožní vložit záznam do paměti pak jej opět z paměti vyjmout.

Každý proces má svojí paměť stack pro záznam. 4th Dimension tuto paměť pro vás udržuje. Každá stack paměť pro záznam je LIFO paměť. Velikost této paměti je omezena pamětí vašeho počítače.

Příkazy PUSH RECORD a POP RECORD používejte opatrně. Každý záznam uložený do paměti zabere její určitou část. Předání příliš velkého počtu záznamů může způsobit nedostatek paměti.

PUSH RECORD a POP RECORD jsou použitelné pokud chcete prozkoumat záznam ve stejném souboru během vstupu dat. K tomu vsunete záznam do paměti, nleznete a prozkoumáte záznamy v souboru (například zkopírujete pole do proměnných) a nakonec vyjmete záznam z paměti a obnovíte jej.

Poznámka k uživatelům verze 3: Když při vkládání záznamu potřebujete ověřit jedinečnost polí, použijte nový příkaz SET QUERY DESTINATION. To vám ušetří provádění příkazů PUSH RECORD a POP RECORD před a po provedení dotazu. SET QUERY DESTINATION vám umožní provést dotaz bez změny platného výběru a záznamu.

Dále si přečtěte

POP RECORD, PUSH RECORD, SET QUERY DESTINATION.

Příkazy a odkazy pro Záznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Vztahy

Příkazy a odkazy pro Vztahy

Verze 6.0

Příkazy v této části, zejména RELATE ONE a RELATE MANY, vytvářejí a řídí automatické a neautomatické vztahy mezi tabulkami. Před použitím než použijete některý z příkazů v této části si přečtěte v *Příručce Návrháře 4th Dimension* informace o vytváření vztahů mezi tabulkami.

Vytvoření automatických vztahů pomocí příkazů

Dvě tabulky mohou být spojeny automatickým vztahem. Jakmile je automatický vztah vytvořen, načítá nebo označuje vztažené záznamy mezi tabulkami. Vytvoření vztahu ovlivňuje mnoho operací.

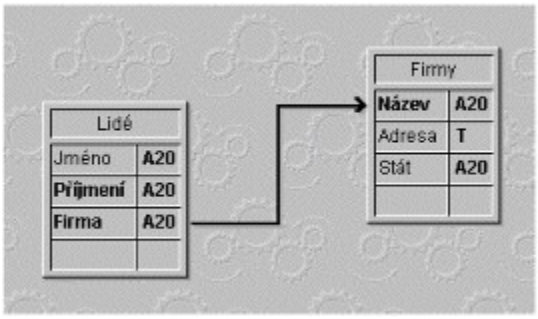
Jsou to následující:

- Zadávání dat
- Listování záznamy ve výstupním formuláři
- Vytváření zpráv
- Operace s výběrem záznamů, jako je dotaz, třídní a použití výrazu

Aby se optimalizoval výkon, když 4th Dimension vytvoří automatické vztahy, je pro tabulku vytvořen vždy pouze jeden platný záznam. Pro všechny následující operace jsou načteny vztažené záznamy podle následujících pravidel:

- Jestliže vztah vybere pouze jeden záznam ze vztažené tabulky, je tento záznam načten do paměti.
- Jestliže vztah vybere více záznamů ze vztažené tabulky, je pro tu tabulku vytvořen nový platný výběr a první záznam je načten do paměti.

Například s použitím struktury zobrazené zde: jestliže je otevřen a zobrazen pro vstup dat záznam z tabulky [Lidé], je označen a načten vztažený záznam v tabulce [Firmy]. Stejně tak pokud je otevřen záznam v tabulce [Firmy], jsou vybrány vztažené záznamy v tabulce [Lidé].



V této struktuře je tabulka [Lidé] **tabulka skupiny** a tabulka [Firmy] je **tabulka jedince**. Nezapomeňte tento koncept. Přemýšlejte o něm takto "více lidí pracuje v jedné firmě" a "každá firma má více lidí".

Stejně tak je pole Firma v tabulce [Lidé] **pole skupiny** a pole Název v tabulce [Firmy] je **pole jedince**.

Vždy není možné mít vztažená pole jedinečná. Například název firmy může být stejný pro více firem. Tuto nejedinečnou situaci můžete vyřešit přidáním dalšího jedinečné pole, podle kterého budete mít vztah vytvořen. Třeba pole ID firmy.

Následující tabulka ukazuje záznamy které používají automatické vztahy během akce příkazu. Všechny příkazy automaticky vytvářejí vztah Skupiny k Jedinci. Ale pouze příkazy u kterých je Ano v druhém sloupci automaticky vytvářejí vztah Jedince ke Skupině.

Příkaz	*vztah Jedince ke Skupině*
ADD RECORD	*Ano*
ADD SUBRECORD	*Ne*
APPLY TO SELECTION	*Ne*
DISPLAY SELECTION	*Ne*
EXPORT DIF	*Ne*
EXPORT SYLK	*Ne*
EXPORT TEXT	*Ne*
MODIFY RECORD	*Ano*
MODIFY SUBRECORD	*Ne*
MODIFY SELECTION	*Ano (ve vstupu dat)*
ORDER BY	*Ne*
ORDER BY FORMULA	*Ne*
QUERY BY FORMULA	*Ano*
QUERY SELECTION	*Ano*
QUERY	*Ano*
PRINT LABEL	*Ne*
PRINT SELECTION	*Ano*
REPORT	*Ne*
SELECTION TO ARRAY	*Ne*
SELECTION RANGE TO ARRAY	*Ne*

Použití příkazů k vytvoření vztahu tabulky

Automatický vztah neznamená že vztažený záznam nebo záznamy budou prostě načteny protože příkaz načetl záznam. V některých případech, pokud chcete přístup ke vztaženým datům, musíte po otevření záznamu příkazem ještě označit vztažené záznamy pomocí příkazu RELATE ONE nebo RELATE MANY.

Některé z příkazů zobrazené v předchozí tabulce (jaké příkazy dotazů) po dokončení akce načtou platný záznam. V tomto případě načtený záznam neoznačí automaticky vztažené záznamy. Pokud zde potřebujete vztažená data, musíte použít příkaz RELATE ONE nebo RELATE MANY.

Dále si přečtěte

AUTOMATIC RELATIONS, CREATE RELATED ONE, OLD RELATED MANY, OLD RELATED ONE, RELATE MANY, RELATE MANY SELECTION, RELATE ONE, RELATE ONE SELECTION, SAVE OLD RELATED ONE, SAVE RELATED ONE.

Příkazy a odkazy pro Vztahy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

AUTOMATIC RELATIONS (AUTOMATICKÉ VZTAHY)

Příkazy a odkazy pro Vztahy

Verze 3

AUTOMATIC RELATIONS (jedinci{; skupiny})

Parametr	Typ		Popis
Jedinci	Logické	→	Všechny M:1 vztahy automaticky
Skupiny	Logické	→	Všechny 1:M vztahy automaticky

Popis

AUTOMATIC RELATIONS dočasně mění všechny ruční vztahy na automatické. Vztah zůstane automatický dokud znovu neprovedete příkaz AUTOMATIC RELATIONS.

Pokud je Jedinci True, pak jsou všechny vztahy Skupiny k Jedinci automatické. Jestliže Jedinci False, všechny předtím upravené vztahy Skupiny k Jedinci se vrátí na ruční.

Stejně je to s parametrem Skupiny, pouze s tím rozdílem, že se jedná o vztahy Jedince ke Skupině.

Vztahy které byli označeny jako automatické v prostředí návrháře se měnit nebudou.

Pokud byli nastaveny všechny vztahy jako ruční, tento příkaz je změní na automatické před operací která vyžaduje automatické vztahy (jako hledání a třídění). Po dokončení akce mohou být vztahy nastaveny zpět.

Příklad

Následující příklad změní všechny vztahy Skupiny k Jedinci na automatické a vrátí zpět všechny vztahy Skupiny k Jednici:

```
AUTOMATIC RELATIONS (True;False)
```

Dále si přečtěte

Vztahy, SELECTION RANGE TO ARRAY, SELECTION TO ARRAY.

Příkazy a odkazy pro Vztahy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

RELATE ONE (VZTÁHNOUT K JEDINCI)

Příkazy a odkazy pro Vztahy

Verze 3

RELATE ONE (TabulkaSkupin | Pole{; VýbPole})

Parametr	Typ		Popis
TabulkaSkupin | Pole	Tabulka | Pole	→	Tabulka, pro kterou chceme nastavit všechny vztahy automatické nebo pole s neautomatickým vztahem k tabulce jedinců
VýbPole	Pole	→	Pole tabulky jedinců pro informace při výběru seznamu

Popis

Příkaz RELATE ONE má dvě použití.

První způsob, RELATE ONE(TabulkaSkupin), vytvoří všechny automatické vztahy Skupiny k Jedinci v platném procesu. To znamená že pro každé pole v tabulce skupiny která má automatický vztah Skupiny k Jedinci, označí příkaz vztažený záznam v každé vztažené tabulce. Změníte tím platné záznamy pro vztažené tabulky.

Druhý způsob, RELATE ONE(Pole{;VýbPole}), vyhledá pole vztažená k poli skupiny. Vztah nemusí být automatický. Pokud exituje, RELATE ONE načte záznamy do paměti a udělá z nich platný záznam a platný výběr pro jejich tabulky.

Volitelný parametr VýbPole může být zadán pouze když pole skupiny je Alfy pole. VýbPole musí být ze vztažené tabulky a musí být typu Alfa, číselné, datumové, časové nebo Logické. Nemůže být textové, BLOB nebo podtabulka.

Jestliže je definováno VýbPole a je nalezen více než jeden záznam ve vztažené tabulce, RELATE ONE zobrazí seznam záznamů, které vyhovují hodnotě v poli skupiny. V tomto seznamu bude levý sloupec zobrazovat hodnotu pole skupiny a pravý sloupec bude zobrazovat hodnotu výběrového pole.

Pokud bude hodnota pole skupiny končit znakem @, může být nalezen více než jeden záznam. Pokud je nalezena pouze jedna hodnota, seznam se nebude zobrazovat. Definování pole VýbPole je to samé jako definování Zástupného výběru při vytváření vztahu v prostředí návrháře. Jestli chcete vědět více informací o definování zástupného výběru, přečtěte si *Příručku návrháře 4th Dimension.*

RELATE ONE pracuje i se vztahy k podtabulkám, ale musíte mít vztah k rodičovské tabulce a k vztaženému poli podtabulky ve stavu kdy je možno vztah pořádně vytvořit. Při používání vztahu k podzáznamu, musíte použít RELATE ONE pro načtení záznamu do paměti a pak použít ještě jednou RELATE ONE pro podtabulku.

Příklad

Řekněme, že máte tabulku [Faktury] vztaženou neautomatickým vztahem k tabulce [Zákazníci]. Vztah je z pole [Faktury]IDZákazníka k [Zákazníci]ID a další vztah je z pole [Faktury]Dodavatel k poli [Zákazníci]ID.

Jelikož jsou oba vztahy k jedné tabulce, [Zákazníci], nemůžete najednou získat informace o prodejích i dodávkách. Proto zobrazení obou adres ve formuláři musí být provedeno pomocí proměnných a příkazu RELATE ONE. Jakmile je zobrazen [Zákazník], mohou být zobrazena data pouze z jednoho vztahu.

Následující dvě metody jsou metody objektu, jsou to metody objektu polí [Faktury]IDZákazníka a [Faktury]Dodavatel. Spustí se při předání dat do pole.

Zde je metoda objektu pro [Faktury]IDZákazníka:

```
RELATE ONE ([Faktury]IDZákazníka)
```

```
vAdresa1:=[Zákazníci]Adresa
vMěsto1:= [Zákazníci]Město
vStát1:= [Zákazníci]Stát
vPSČ1:=[Zákazníci]PSČ
```

Zde je metoda objektu pro pole [Faktury]Dodavatel:

```
RELATE ONE ([Faktury]Dodavatel)
vAdresa2:=[Zákazníci]Adresa
vMěsto2:= [Zákazníci]Město
vStát2:= [Zákazníci]Stát
vPSČ2:=[Zákazníci]PSČ
```

Dále si přečtěte

OLD RELATED ONE, RELATE MANY.

Příkazy a odkazy pro Vztahy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

RELATE MANY (VZTÁHNOUT K SKUPINĚ)

Příkazy a odkazy pro Vztahy

Verze 3

RELATE MANY (TabulkaJed | Pole)

Parametr	Typ		Popis
TabulkaJed | Pole	Tabulka | Pole	→	Tabulka k aktivaci vztahů jedinec k skupině 1:M, nebo pole jedinců (rodičů)

Popis

Příkaz RELATE MANY má dvě použití.

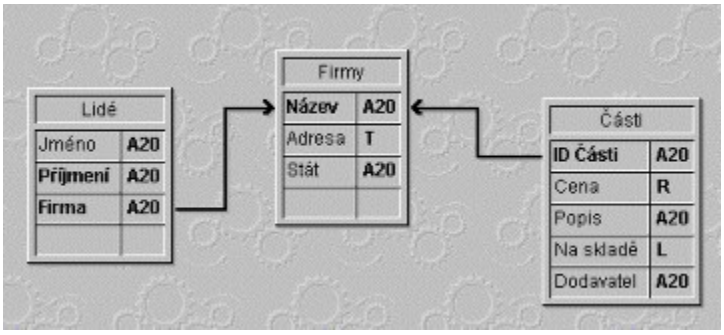
První způsob, RELATE MANY(tabulkaJed), vytvoří všechny vztahy Jedince ke Skupině pro TabulkaJed. Změní platný výběr pro každou tabulku která má vztah Jednice ke Skupině k TabulkaJed. Výběr v tabulce Skupiny záleží na hodnotě každého vztaženého pole v tabulce Jedince. Po každém použití tohoto příkazu se platný výběr v tabulce Skupiny vytvoří znovu.

Druhý způsob, RELATE MANY(poleJedince), vytvoří vztah Jedince ke Skupině pro pole Jedince. Změní platný výběr pouze u těch tabulek Skupiny, které mají vztah k poliJedince. To znamená, že vybrané záznamy se stanou platným výběrem pro tabulku.

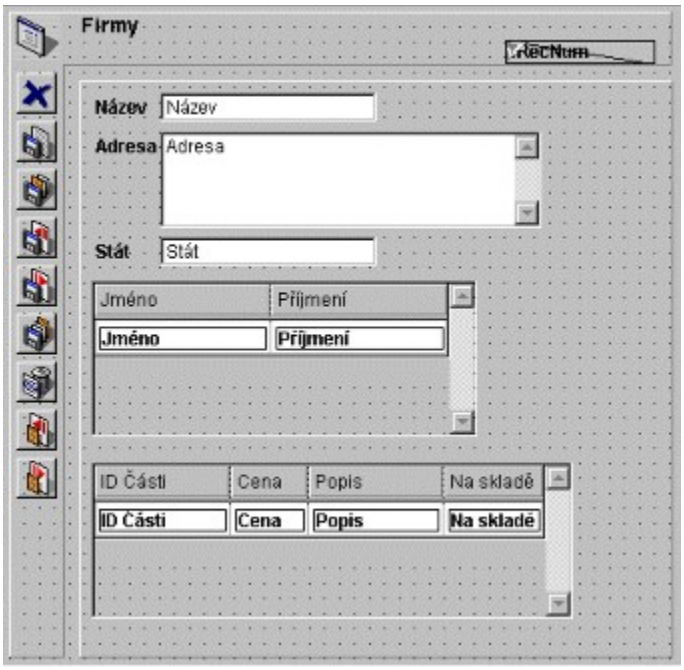
Poznámka: Pokud je během provedení příkazu RELATE MANY výběr v tabulce Jedince prázdný, příkaz neprovede nic.

Příklad

V následujícím příkladu jsou spojeny tři tabulky automatickým vztahem. Tabulky [Lidé] a [Části] mají automatický vztah Skupiny k Jedinci k tabulce [Firmy].



Tento formulář tabulky [Firmy] zobrazuje záznamy ze vztažených tabulek [Lidé] a [Části].



Jakmile jsou formuláře pro Lidé a Části zobrazeny, vztažené záznamy tabulek jsou načteny a stanou se z nich platné výběry tabulek.

Na druhou stránku nejsou vztažené záznamy načteny, pokud je záznam v tabulce [Firmy] označen programově.

Poznámka: Pokud je příkaz RELATE MANY proveden na prázdný výběr, není tento příkaz proveden a výběr v tabulce SKUPINY se nezmění.

Například následující metoda bude listovat všemi záznamy v tabulce [Firmy]. Pro každý záznam zobrazí upozornění které zobrazí počet lidí ve firmě (počet vztažených záznamů v tabulce [Lidé]) a počet částí které dodávají (počet vztažených záznamů v tabulce [Části]). V tomto příkladu se upozornění zobrazí ve více řádkách.

Nezapomeňte, že RELATE MANY je někdy potřeba i v případě automatických vztahů:

```
ALL RECORDS ([Firmy]) ` Vybrat všechny záznamy v tabulce
ORDER BY ([Firmy]; [Firmy]Jméno) ` Setřídit podle abecedy
For ($i; 1; Records in table ([Firmy])) ` Jedna smyčka na záznam
   RELATE MANY ([Firmy]Jméno) ` Vybrat vztažené záznamy
   ALERT ("Firma: "+[Firmy]Název+Char (13)+"Lidí ve firmě: "+
            String (Records in selection ([Lidé]))+Char(13)+
      "Počet částí které dodávají: "+ String (Records in selection ([Části])))
   NEXT RECORD ([Companies]) ` Jít na další záznam
End for
```

Dále si přečtěte

OLD RELATED MANY, RELATE ONE.

Příkazy a odkazy pro Vztahy

4th Dimension 6.5, Seznam témat příkazů

CREATE RELATED ONE (VYTVOŘIT VZTAŽENÉHO JEDINCE)

Příkazy a odkazy pro Vztahy

Verze 3

CREATE RELATED ONE (pole)

Parametr	Typ		Popis
pole	Pole	→	Pole skupiny

Popis

CREATE RELATED ONE má dvě funkce. Jestliže vztažený záznam pro pole neexistuje (to je když není nalezen pro hodnotu v poli skupiny), CREATE RELATED ONE vytvoří nový vztažený záznam.

K uložení hodnoty do správného pole, přiřaďte hodnoty z pole Skupiny do pole Jedince. Proveďte SAVE RELATED ONE k uložení nového záznamu.

Pokud vztažený záznam existuje, CREATE RELATED ONE funguje jako RELATE ONE a načte jej do paměti.

Dále si přečtěte

SAVE RELATED ONE.

Příkazy a odkazy pro Vztahy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SAVE RELATED ONE (ULOŽIT VZTAŽENÉHO JEDINCE)

Příkazy a odkazy pro Vztahy

Verze 3

SAVE RELATED ONE (pole)

Parametr	Typ		Popis
pole	Pole	→	Pole Skupiny

Popis

SAVE RELATED ONE uloží záznam vztažený k poli. Proveďte příkaz SAVE RELATED ONE k uložení záznamu vytvořeného pomocí CREATE RELATED ONE nebo otevřeného příkazem RELATE ONE.

SAVE RELATED ONE neúčinkuje na podzáznamy, protože uložení rodičovského záznamu automaticky uloží jeho podzáznamy.

S tímto příkazem nelze ukládat uzamčené záznamy. Při používání tohoto příkazu si musíte být jisti, že je záznam odemčený. Pokud je záznam zamčený, je příkaz ignorován, záznam není uložen a je generována chyba.

Dále si přečtěte

CREATE RELATED ONE, Locked, RELATE ONE, Triggery.

Příkazy a odkazy pro Vztahy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

OLD RELATED ONE (VZTÁHNOUT K PŮVODNÍMU JEDINCI)

Příkazy a odkazy pro Vztahy

Verze 3

OLD RELATED ONE (pole)

Parametr	Typ		Popis
pole	Pole	→	Pole skupiny

Popis

OLD RELATED ONE dělá to samé jako RELATE ONE, ale používá starou hodnotu pole. Hodnotu, kterou vrací příkaz Old. Pro více informací si přečtěte popis k příkazu Old.

Poznámka: OLD RELATED ONE používá starou hodnotu pole Skupiny, kterou vrací funkce Old. Podprobnější informace najdete u příkazu Old.

OLD RELATED ONE načte záznam který byl předchozí vztažený k platnému záznamu. Pole v tom záznamu pak mohou být zpřístupěna. Pokud chcete tento starý vztažený záznam změnit a uložit, musíte k tomu použít příkaz SAVE OLD RELATED ONE. Nezapomeňte že u nově vytvořených záznamů nejsou žádné staré vztažené záznamy.

Dále si přečtěte

Old, OLD RELATED MANY, RELATE MANY, SAVE OLD RELATED ONE.

Příkazy a odkazy pro Vztahy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SAVE OLD RELATED ONE (ULOŽIT PŮVODNÍHO VZTAŽENÉHO JEDINCE)

Verze 3

SAVE OLD RELATED ONE (pole)

Parametr	Typ		Popis
pole	Pole	→	Pole skupiny

Popis

SAVE OLD RELATED ONE pracuje stejně jako SAVE RELATED ONE, ale ukládá starý vztažený záznam a používá k tomu starý vztah k poli. Před tím než použijete SAVE OLD RELATED ONE, musíte záznam načíst pomocí OLD RELATED ONE. Použijte tento příkaz pokud chcete uložit změny v záznamu otevřeném příkazem OLD RELATED ONE.

SAVE OLD RELATED ONE neuloží zamčený záznam. Při používání tohoto příkazu si musíte být jisti, že je záznam odemčený. Pokud je záznam zamčený, je příkaz ignorován, záznam není uložen a je generována chyba.

Dále si přečtěte

Locked, OLD RELATED ONE, Triggery.

Příkazy a odkazy pro Vztahy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

OLD RELATED MANY (VZTÁHNOUT K PŮVODNÍ SKUPINĚ)

Verze 3

OLD RELATED MANY (pole)

Parametr	Typ		Popis
pole	Pole	→	Pole jedince

Popis

OLD RELATED MANY pracuje stejným způsobem jako RELATE MANY, s tím rozdílem že OLD RELATED MANY používá k vytvoření vztahu starou hodnotu v poli jedince.

Poznámka: OLD RELATED MANY používá starou hodnotu pole skupiny kterou vrací funkce Old. Jestli chcete vědět více informací, přečtěte si popis k příkazu Old.

OLD RELATED MANY mění výběr vztažené tabulky a označí první záznam výběru jako platný záznam.

Dále si přečtěte

OLD RELATED ONE, RELATE MANY.

Příkazy a odkazy pro Vztahy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

RELATE ONE SELECTION (VZTÁHNOUT VÝBĚR K JEDINCŮM)

Verze 6.0 (upraveno)

RELATE ONE SELECTION (TabSkupin; TabJedinců)

Parametr	Typ		Popis
TabSkupin	tabulka	→	Název tabulky skupin/potomků (zde vztah začíná)
TabJedinců	tabulka	→	Název tabulky jedinců/rodičů (sem se vztah odkazuje)

Popis

Příkaz RELATE ONE SELECTION vytvoří nový výběr záznamů pro tabulku TabJedinců, založenou na záznamech v tabulce TabSkupin.

Tento příkaz může být použit pouze v případě že existuje vztah z TabSkupin do TabJedinců.RELATE ONE SELECTION může fungovat i přes několik úrovní vztahů. Mezi tabulkou TabSkupin a TabJedinců může být několik jiných tabulek a to se vztahem buď automatickým nebo ručním.

Upozornění: Nepoužívejte tento příkaz uvnitř transakce.

Příklad

Následující příklad nalezne všechny klienty, jejichž datum faktury je dnešní.
Zde je jeden způsob jak vytvořit vztah mezi tabulkou [Zákazníci] a tabulkou [Faktury]:

```
CREATE EMPTY SET([Zákazníci];"Payment Due")
QUERY([Faktury];[Faktury]Datum = Current date)
While(Not(End selection([Faktury])))
   RELATE ONE ([Faktury]CustID)
   ADD TO SET([Zákazníci];"Payment Due")
   NEXT RECORD([Faktury])
End while
```

Následující způsob používá RELATE ONE SELECTION k dosažení stejného výsledku:

```
QUERY([Faktury];[Faktury]Datum = Current date)
RELATE ONE SELECTION([Faktury];[ Zákazníci])
```

Dále si přečtěte

QUERY, RELATE MANY SELECTION, RELATE ONE, Sady.

Příkazy a odkazy pro Vztahy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

RELATE MANY SELECTION (VZTÁHNOUT VÝBĚR K SKUPINÁM)

Verze 6.0 (upraveno)

RELATE MANY SELECTION (pole)

Parametr	Typ		Popis
pole	Pole	→	Pole z tabulky skupin/potomků, k vytvoření výběru

Popis

RELATE MANY SELECTION vytvoří výběr v tabulce Skupiny podle výběru v tabulce Jedince.

Poznámka: RELATE MANY SELECTION mění platný záznam v tabulce Jedince.

Upozornění: Nepoužívejte tento příkaz uvnitř transakce.

Příklad

Tento příklad vybere faktury zákazníků kteří mají úvěr vyšší než 10000,- Kč. Vztah mezi tabulkami je přes pole [Faktury]IDZákazníka a [Zákazníci]IDZákazníka.

```
    ` Vybrat zákazníky
QUERY ([Zákazníci];[ Zákazníci]Úvěr>=1000)
    ` Nalézt faktury vztažené k těmto zákazníkům.
RELATE MANY SELECTION ([Faktury]IDZákazníka)
```

Dále si přečtěte

QUERY, RELATE ONE, RELATE ONE SELECTION.

Příkazy a odkazy pro Vztahy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Zdroje

Příkazy a odkazy pro Zdroje

Verze 6.0

Zdroje jsou data jakéhokoli formátu uložená v souboru **.RSR** na Windows a v **zdrojové části** programu na Macintoshi. Zdroje obvykle obsahují data jako jsou řetězce, obrázky, ikony, atd. Můžete samozřejmě vytvořit i vaše vlastní zdroje a uložit do nich co potřebujete.

Datová část a Zdrojová část

Na Macintoshi má každý soubor **datovou část** a **zdrojovou část**. Datová část je shodná se soubory na Windows a UNIX. Zdrojová část na Macintoshi obsahuje zdroje Macintosh a nemá žádný ekvivalent na Windows nebo UNIX.

Zdroje Windows jsou promíchány a přidány do ostatních souborů. Například v aplikaci ve Windows může soubor .EXE obsahovat jak data zdrojů tak kód. Aby jsme u 4th Dimension docílili nazávislosti na platformě, 4th Dimension pracuje s Macintosh zdroji jak na Windows tak na Macintoshi.

4D Transporter

Vzhledem k tomu že na Windows neexistují zdrojové části programu, nástroj **4D Transporter** (dodáváno s Macintosh verzí 4th Dimension) vám umožní převody databází mezi Macintosh a Windows.

Pokud přenášíte databázi z Windows na Macintosh, soubory .4DB a .RSR jsou **spojeny** do jednoho Macintosh souboru. Soubor .4DB se stane datovou částí struktury a .RSR soubor se stane zdrojovou částí souboru struktury. Můžete samozřejmě převést databázi z Macintoshe na Windows. Pak se soubor struktury **rozdělí** na dva soubory .4DB a .RSR. Datová část struktuty bude v souboru .4DB a zdrojová čast struktury bude v souboru .RSR.

Toto je jediný důvod proč existuje nástroj 4D Transporter - dělení a skládání datových a zdrojových částí souborů. Nebude nijak měnit ani překládat data uložená v databázi. Jestli chcete vědět více informací a převádění 4D databází mezi platformami, přečtěte si *4D Transporter manuál.*

Soubor zdrojů

Nezáleží na tom na jaké platformě pracujete, ale soubor struktury databáze není jediný soubor který obsahuje zdroje. Aplikace 4th Dimension samotná obsahuje zdroje. Na Macintoshi jsou tyto zdroje uloženy ve zdrojové části programu. Na Windows jsou uloženy v souboru 4D.RSR a spustitelná část je v souboru 4D.EXE.

4D Plug-in mohou také obsahovat zdroje. Například 4D Write obsahuje zdroje. Na Macintoshi jsou tyto zdroje uloženy v 4D Write 6.0. Na Windows jsou v souboru 4DWRITE.RSR.

Datový soubor databáze 4D také obvykle obsahuje nějaké zdroje. Například pokud uzamknete data pro použití právě s jednou strukturou (pomocí **Customizer Plus**), tato akce vloží WEDD ("wedding" - pevně spojit) zdroj do struktury a dat. Na Macintoshi jsou zdroje ukládány do zdrojové části souboru a na Windows jsou uloženy do souboru .4DR, souboru zdrojů pro data.

Poznámka: Customizer Plus je dodáván s 4th Dimension jak pro Windows tak pro Macintosh.

Na Windows, kromě souboru dat _.4DR, obvykle budete používat standardní Macintosh zdroje jako soubory s koncovou .RSR. Všiměte si, že příkaz Create resource file používá jako výchozí příponu .RES.

Vytváření vlastních zdrojových souborů

Kromě standardních zdrojových souborů 4th Dimension si můžete vytvořit a používat vlastní zdrojové soubory pomocí příkazů Create resource file a Open resource file. Tyto dva příkazy vrací **číslo odkazu na zdrojový soubor** který určuje jedinečný soubor zdrojů. Číslo odkazu na zdrojový soubor je podobné jako číslo odkazu na dokument, které vrací příkazy pro otevření souboru jako Open document. Všechny příkazy 4D pro zdroje mohou pracovat s číslem odkazu na zdrojový soubor. Po skončení práce se souborem zdrojů jej nezapomeňte uzavřít pomocí příkazu CLOSE RESOURCE FILE.

Řetěz zdrojových souborů

Když pracujete s databází, můžete pracovat se **všemi otevřenými soubory zdrojů** nabo se **specifickým souborem zdrojů**.

Najednou může být otevřeno více souborů zdrojů. Je to ten samý případ jako u databáze 4D. Jsou otevřeny následující soubory:

- Na Macintoshi, je otevřen Systémový soubor zdrojů
- Na Windows, soubor ASIPORT.RSR (obsahuje některé ze systémových souborů zdrojů Macintoshe)
- Zdrojový soubor 4th Dimension
- Zdrojový soubor struktury databáze
- Volitelně může být otevřen zdrojový soubor datového souboru
- Nakonec můžete ještě otevřít váš vlastní zdrojový soubor pomocí Open resource file

Tento seznam otevřených zdrojových souborů se nazývá **řetěz zdrojových souborů**. Požadovaný zdroj můžete hledat dvěma způsoby:

- Pokud do příkazu předáte odkaz na zdrojový soubor, bude se prohledávat pouze tento soubor.
- Pokud do příkazu nepředáte žádný odkaz, bude se zdroj hledat ve všech otevřených souborech zdrojů počínaje posledním otevřeným souborem a končí u souboru kerý byl otevřený jako první. Prohlížení řetězu zdrojových souborů probíhá v opačném pořadí než v jakém jsou otevírány - poslední otevřený je prohlížen jako první.

Zde je příklad:

```
$vhResFile:=Create resource file("Just_a_file")
If (OK=1)
   ARRAY STRING(63;asSomeStrings;0)
   STRING LIST TO ARRAY(8;asSomeStrings;$vhResFile)
   ALERT("Velikost array je "+String(Size of array(asSomeStrings))+" prvků.")
   STRING LIST TO ARRAY(8;asSomeStrings)
   ALERT("Velikost array je "+String(Size of array(asSomeStrings))+" prvků.")
   CLOSE RESOURCE FILE($vhResFile)
End if
```

Při spuštění této metody bude první upozornění vypadat "Velikost array je 0 prvků." a druhé upozornění bude "Velikost array je 634 prvků."

První provedení:

```
STRING LIST TO ARRAY(8;asSomeStrings;$vhResFile)
```

bude hledat pouze zdroj "STR#" ID=8 v právě vytvořeném souboru zdrojů. Vzhledem k tomu že je soubor prázdný, nebude nalezeno nic.

Druhé provedení:

STRING LIST TO ARRAY(8;asSomeStrings)

bude hledat ve všech otevřených souborech zdrojů "STR#" ID=8 a to včetně nově otevřeného souboru. Protože nově otevřený soubor zdrojů neobsahuje tento zdroj, STRING LIST TO ARRAY se přesune na soubor zdrojů databáze. Ani tam nic nenalezne a začne hledat v souboru zdrojů 4th Dimension. Nalezne zdroj v tomto souboru a vyplní s ním array.

Závěr: Při práci se specifickým souborem zdrojů se ujistěte, že jste do příkazu 4D vložili správné číslo odkazu na zdrojový soubor. Jinak nebude 4th Dimension vědět, se kterým souborem chcete pracovat.

Typ zdrojů

Zdrojový soubor je uvnitř strukturován. Kvůli datům v každém zdroji obsahuje záhlaví a mapu která celkově popisuje jeho obsah.

Zdroje jsou děleny podle **typů**. Typy jsou vždy udávány 4-znakovým řetězcem. Typy zdrojů jsou citlivé na velikost písmen a proto zdroje "Hi_!", "hi_!" a "HI_!" jsou odlišné.

Důležité: Typy zdrojů zapsané malými písmeny jsou zadané pro Operační systém. Vyhněte se tomu nazývyt své vlastní zdroje malými písmeny.

Následující seznam ukazuje nejčastěji používané typy zdrojů:

• Zdroj typu "STR#" je zdroj obsahující seznam Pascalovských řetězců. Tento zdroj se nazývá **zdroj seznamu řetězců**.
• Zdroj typu "STR " (nezapomeňte mezeru jako čtvrtý znak) je zdroj obsahující jednotlivé pascalovské řetězce. Tento zdroj se nazývá **zdroj řetězce**.
• Zdroj typu "TEXT" je zdroj obsahující textové řetězce bez zadané délky. Tento zdoj se nazývá **textový zdroj**.
• Zdroj typu "PICT" je zdroj obsahující QuickDraw Macintosh obrázky, které můžete otevřít jak na Windows tak na Macintoshi. Tento zdroj se nazývá **obrázkový zdroj**.
• Zdroj typu "cicn" je zdroj obsahující barevné ikony, které můžete použít k zobrazení s 4D jak na Windows tak na Macintoshi. Tento zdroj se nazývá **zdroj barevných ikon**. Ikony z tohoto zdroje mohou být pomocí příkazu SET LIST ITEM PROPERTIES například přiřazeny k položkám hierarchického seznamu.

Mimo standardních typů zdrojů si můžete samozřejmě vytvořit i své vlastní zdroje. Můžete si například vytvořit zdroj "MTYP" (pro "Můj typ").

Aby jste zjistili, jaké všechny typy zdrojů jsou obsaženy ve všech otevřených souborech, použijte k tomu příkaz RESOURCE TYPE LIST. K zjištění jednoho typu zdrojů ve všech otevřených souborech zdrojů nebo v jednom souboru zdrojů, použijte příkaz RESOURCE LIST. Tento příkaz vám vrátí ID a názvy (přečtěte si další část) všech všech zdrojů zadaného typu.

UPOZORNĚNÍ: Mnoho aplikací je závislých na zdrojích, které obsahují jejich data. Například při práci při přístupu ke zdroji "STR#", očekává aplikace ve zdroji seznam řetězců. NEUKLÁDEJTE jiná data do zdrojů, které mají zadaný typ. Můžete tím způsobit systémovou chybu 4D nebo jiné aplikace.

UPOZORNĚNÍ: Zdroj je vysoce strukturovaný soubor - NEMĚŇTE tyto soubory jinými příkazy než příkazy zdrojů 4D. Nezapomeňte, že vám nic nebrání zpřístupnit soubor zdroje pomocí čísla odkazu na soubor zdroje (je to čas 4D stejně jako u odkazu na dokument) a toto číslo můžete přiřadit třeba k příkazu SEND PACKET. Jestliže to uděláte, pravděpodobně poškodíte soubor zdrojů.

UPOZORNĚNÍ: Soubor zdroje může obsahovat až 2700 jednotlivých zdrojů. NEZAPOMEŇTE tento limit. Nic vám nebrání v tom, jej překročit, nicméně pokud to uděláte, můžete tím poškodit soubor zdrojů a tento soubor již nemusí být použitelný.

Název a ID zdroje

Zdroj má **název zdroje**. Tento název může být až 255 znaků dlouhý. Rozlišuje jednotlivou diakritiku, ale nerozlišuje malá a velká písmena. Tyto názvy se používají pro popsání zdrojů, ale přístup ke zdrojům vám zajišťuje ID zdroje. Názvy zdrojů nemusí být jedinečné; některé z nich mohou mít stejné názvy.

Zdroj má i **číslo ID zdroje** (zkráceně ID zdroje nebo ID). Toto ID je jedinečné v typu zdroje a souboru zdrojů. Například:

• Jeden soubor zdrojů může obsahovat zdroj "ABCD" ID=1 a zdroj "EFGH" ID=1
• Dva soubory zdrojů mohou obsahovat zdroje stejného typu a stejného ID.

Pokud používáte zdroj pomocí příkazů 4D, definujete jeho typ a ID. Pokud nedefinujete soubor ze kterého chcete zdroj číst, příkaz bude prohledávat všechny otevřené soubory zdrojů a vrátí vám první hodnotu která bude vyhovovat zadaným hodnotám. Zapamatujte si, že zdroje jsou prohledávány v opačném pořadí, než v jakém byli otevřeny.

Důležité: Použijte rozsah mezi 15000 a 32767 pro vaše vlastní zdroje. NEPOUŽÍVEJTE záporná čísla pro ID zdrojů; tyto jsou obráceny a použity Operčním systémem. NEPOUŽÍVEJTE zdroje v rozsahu 0...14999; tato čísla jsou používána 4th Dimension.

K zjištění ID a názvu zdroje zadaného typu, použijte příkaz RESORCE LIST.

K získání názvu jednotlivého zdroje, použijte příkaz Get resource name.

Ke změně názvu jednotlivého zdroje, použijte příkaz SET RESOURCE NAME.

Stejně jako všechny příkazy 4D volitelně akceptují číslo odkazu na zdrojový soubor, můžete snadno pracovat se zdroji které mají stejný typ a ID ve dvou rozdílných zdrojových souborech. Následující příklad překopíruje všechny "PICT" zdroje z jednoho souboru do druhého:

```
      ` Otevřít existující zdrojový soubor
   $vhResFileA:=Open resource file("")
   If (OK=1)
      ` Vytvořit nový zdrojový soubor
      $vhResFileB:=Create resource file("")
      If (OK=1)
         ` Získat názvy a ID všech zdrojů "PICT"
         ` umístěné ve zdrojovém souboru A
         RESOURCE LIST("PICT";$aiResID;$asResName;$vhResFileA)
         ` Pro každý zdroj:
         For($vlElem;1;Size of array($aiResID))
            $viResID:=$aiResID{$vlElem}
            ` Načíst zdroj ze zdrojového souboru A
            GET RESOURCE ("PICT";$viResID;vxResData;$vhResFileA)
            ` Pokud může být zdroj načten
            If (OK=1)
               ` Přidat a zapsat zdroj do souboru B
               SET RESOURCE ("PICT";$viResID;vxResData;$vhResFileB)
               ` Pokud může být zdroj přidán a zapsán
               If (OK=1)
                  ` Zkopírovat i název zdroje
                  SET RESOURCE NAME("PICT;$viResID; $asResName{$vlElem};$vhResFileB)
                  ` Stejně jako jeho vlastnosti
                  ` (Podívejte se na následující diskuzi o vlastnostech zdrojů)
                  $vlResAttr:=Get resource properties("PICT";$viResID; $vhResFileA)
                  SET RESOURCE PROPERTIES("PICT";$viResID; $vlResAttr;$vhResFileB)
               Else
```

```
                ALERT("Zdroj PICT ID="+String($viResID)+" nemůže být zapsán.")
            End if
        Else
            ALERT("Zdroj PICT ID="+String($viResID)+" nemůže být načten.")
        End if
    End for
    CLOSE RESOURCE FILE($vhResFileB)
  End if
  CLOSE RESOURCE FILE($vhResFileA)
End if
```

Vlastnosti zdroje

Kromě typu, ID a názvu má zdroj i další **vlastnosti** (také nazývané předvolby). Zdroj například může a nemusí být vymazatelný. Tato vlastnost řekne Operačnímu systému, kdy ano a kdy ne, má být zdroj odstraněn z paměti když je volná paměť vyžadovaná pro umístění jiného objektu. Jak je ukázáno v předchozím příkladu tak při vytváření nebo kopírování zdrojů, nemusí stačit přenést zdroj, ale i jeho název a vlastnosti. Podrobný popis vlastností zdrojů najdete u příkazu SET RECOURCE PROPERTIES.

Zacházení s obsahem zdrojů

K načtení zdroje jakéhokoli typu do paměti, použijte příkaz GET RESOURCE, který vrátí obsah zdroje do BLOBu.

K přidání nebo přepsání zdroje na disku, použijte příkaz SET RESOURCE, který nastaví obsah zdroje na obsah BLOBu který předáte jako parametr.

K vymazání existujícího zdroje, použijte příkaz DELETE RESOURCE.

K jednoduchému ovládání standardních typů zdrojů, 4D obsahuje následující vestavěné příkazy, které vám ušetří práci s BLOBem a čtením z něj:

- STRING LIST TO ARRAY naplní textový nebo řetězcový array řetězci obsaženými ve zdroji seznamu řetězců.
- ARRAY TO STRING LIST vytvoří nebo přepíše zdroj řetězce textovým nebo řetězcovým array.
- Get indexed string vrací patřičný řetězec ze zdroje seznamu řetězců.
- Get string resource vrací řetězec ze zdroje řetězců.
- SET STRING RESOURCE vytvoří nebo přepíše zdroj řetězců.
- Get text resource vrací text z textového zdroje.
- SET TEXT RESOURCE vytvoří nebo přepíše textový zdroj.
- GET PICTURE RESOURCE vrací obrázek z obrázkového zdroje.
- SET PICTURE RESOURCE vytvoří nebo přepíše obrázkový zdroj.
- GET ICON RESOURCE vrací zdroj barevné ikony jako obrázek.

Nezapomeňte že tyto příkazy jsou určeny pro práci se standardními typy zdrojů; nicméně vám to nebrání používat příkazy GET RESOURCE a SET RESOURCE s použitím BLOBů. Například tento řádek kódu:

```
ALERT(Get text resource(20000))
```

Je kratší způsob pro:

```
GET RESOURCE("TEXT";20000;vxData)
If (OK=1)
    $vlPosun:=0
```

```
    ALERT(BLOB to text (vxData;Text without length;$vlPosun;BLOB Size(vxData)))
  End if
```

Příkazy 4D a zdroje

Mimo příkazů popsaných v této kapitole existují ve 4D ještě další příkazy určené k práci se zdroji a zdrojovými soubory.

- Na Macintoshi mohou příkazy DOCUMENT TO BLOB a BLOB TO DOCUMENT načítat a zapisovat do jakéhokoli zdrojové části Macintosh souboru.
- S použitím příkazů SET LIST ITEM PROPERTIES a SET LIST PROPERTIES můžete přiřadit k seznamu nebo položce seznamu obrázek nebo barevnou ikonu ze zdroje nebo použít zdroje barevných ikon jako uzel seznamu.
- Příkaz PLAY zahraje zahraje "snd " zdroj jak na Macintoshi tak na Windows.
- Příkaz SET CURSOR změní kurzor myši s použitím zdroje "CURS"

Dále si přečtěte

Příkazy BLOB, Chyby manažeru zdrojů OS, Zdroje a 4D Insider: příklad.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

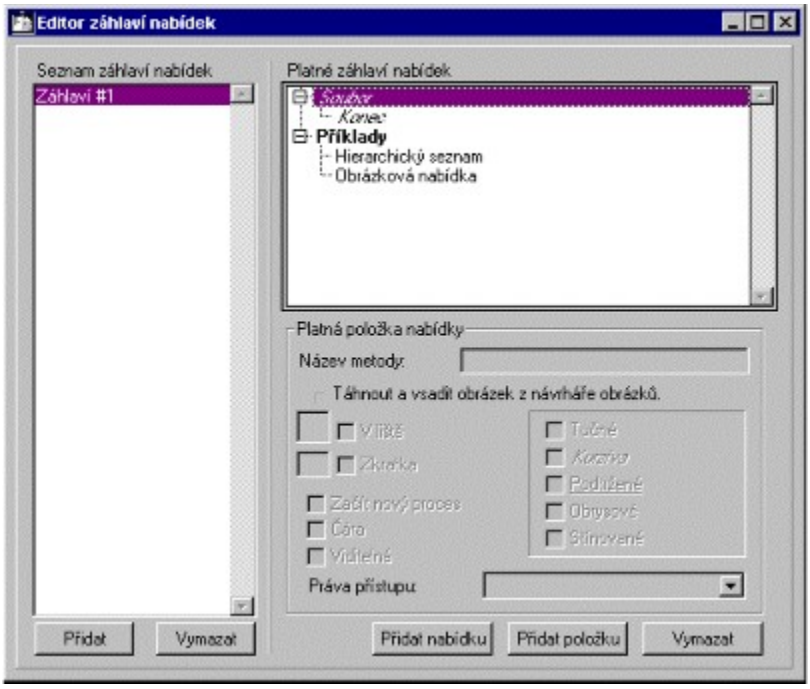
Zdroje a 4D Insider: Příklad

Příkazy a odkazy pro Zdroje

Verze 6.0

Zdroje jsou snadný způsob jak upravovat lokalizaci databáze 4D při vývoji aplikace s více jazyky pro mezinárodní trh.

Podívejme se na příklad. Následující obrázek ukazuje záhlaví nabídek v češtině:

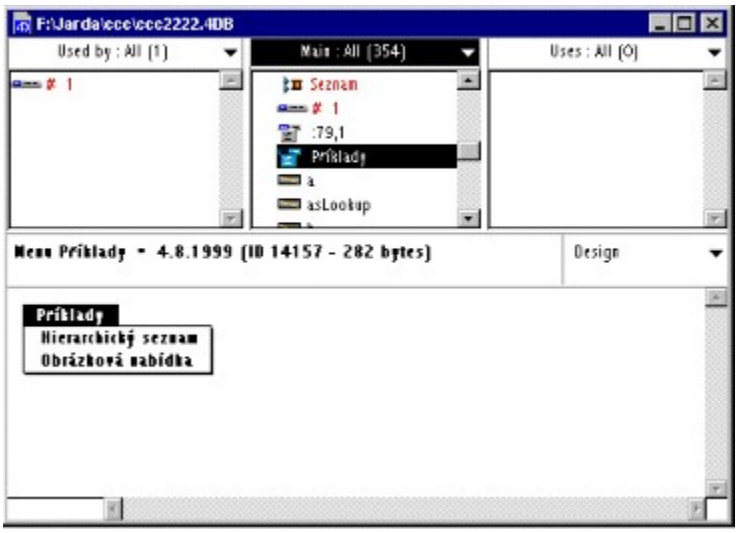


Titulek nabídky Soubor **se vždy odkazuje na zdroj, zatímco jeho položka nabídky** Konec **ne. Nabídka Příklady je složena z položek** Hierarchický seznam **a** Obrázková nabídka**. Tato nabídka a její položky se neodkazují na zdroje.**

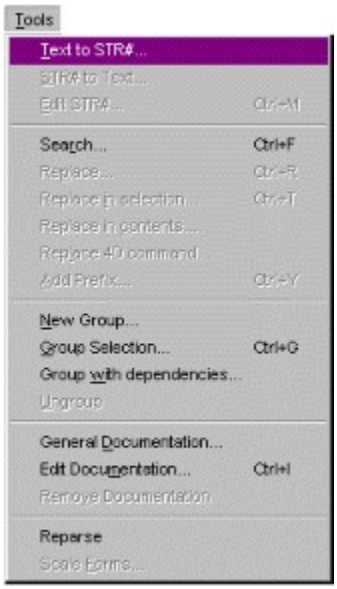
S použitím 4D Insider je možné převést znaky záhlaví nabídky do zdrojů "STR#". Podívejme jak to udělat.

Poznámka: 4D Insider je nástroj pracující přes všechny odkazy a řídící knihovny, který je dodáván se 4D Desktop.

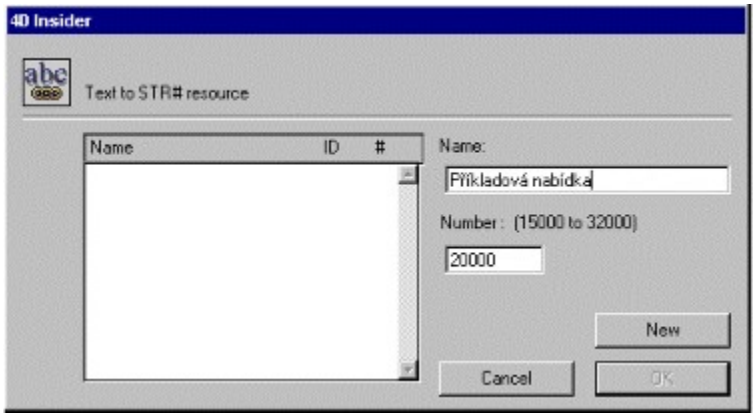
1. Otevřete databázi pomocí 4D Insider. Následující obrázek ukazuje nabídku databáze v okně 4D Insider.



2. V této chvíli můžete převést záhlaví nabídek do zdroje "STR#". K tomu vyberte položku **Text to STR#** z nabídky **Tools**.



Objeví se okno pro převod textu do STR# zdroje.



3. Zadejte název a ID zdroje, a klepněte na **New.** Například **Příkladová nabídka** bude název zdroje a **20000** bude ID zdroje. Zdroj bude vytvořen.

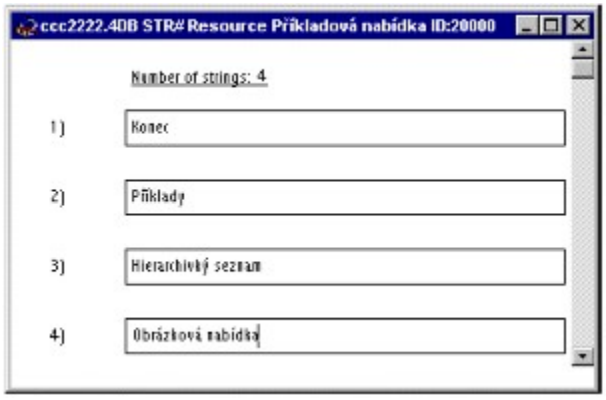
4. Vyberte **STR#** v rozevírací nabídce hlavního okna.

5. Poklepejte na položku STR# 20000 k zobrazení jejího obsahu.

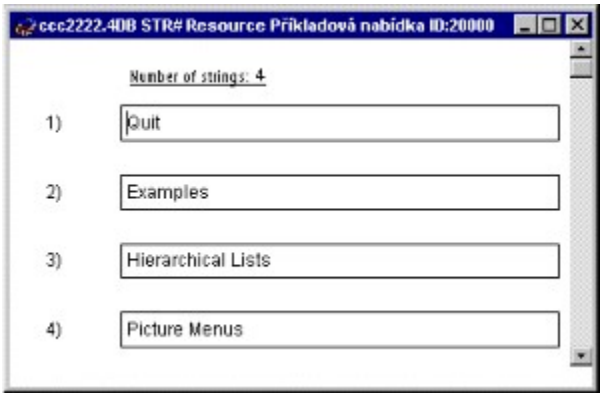
Nyní jsou tyto řetězce uloženy ve zdroji. Je možné změnit jejich hodnotu bez pletení se do vývoje databáze.

6. K upravení hodnoty označte zdroj **Příkladová nabídka a** vyberte položku **Edit STR#** z nabídky **Tools**.

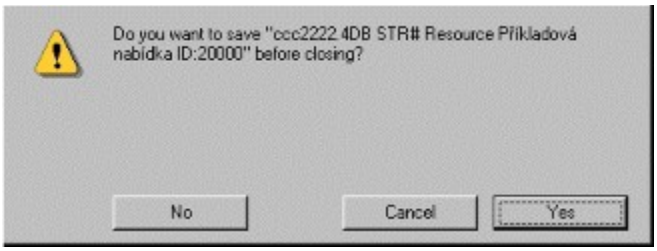
Objeví se okno pro upravení STR# zdroje:



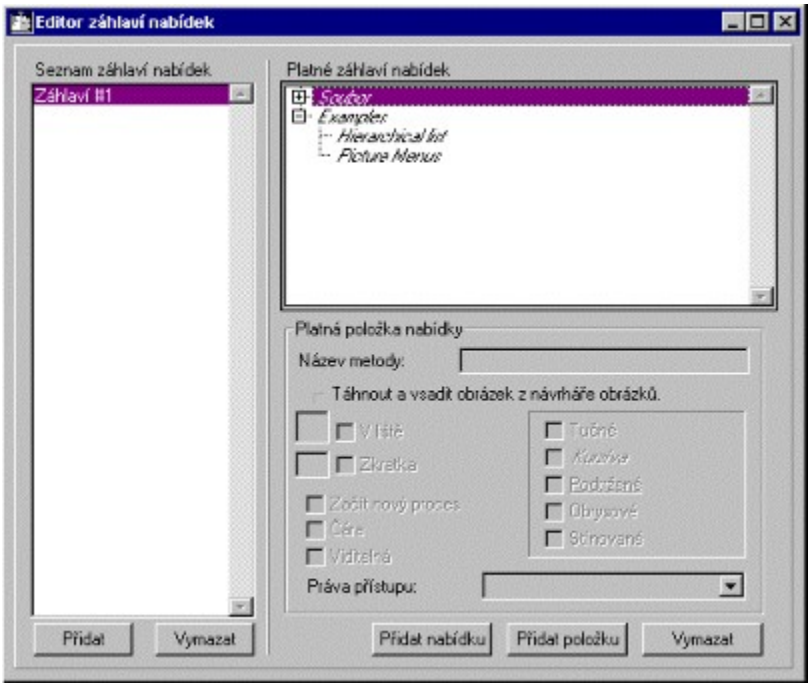
7. Přeložte položky do jiného jazyka. V tomto příkladu byli položky přeloženy do angličtiny.



8. Jakmile jste dokončili překlad, zavřete okno. Klepněte na **Yes** v okně potvrzení:

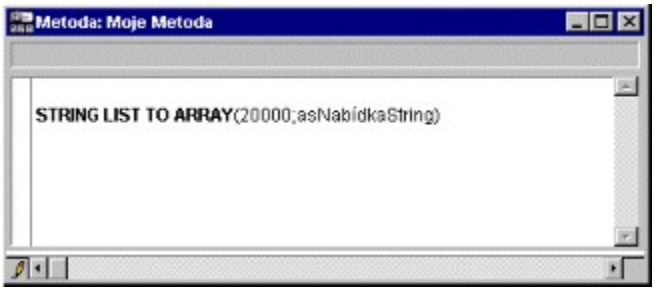


9. Nyní vypněte 4D Insider a otevřete znovu databázi pomocí 4th Dimension. Editor nabídek v Prostředí návrháře nyní bude ukazovat nabídku s odkazy na zdroje v angličtině:

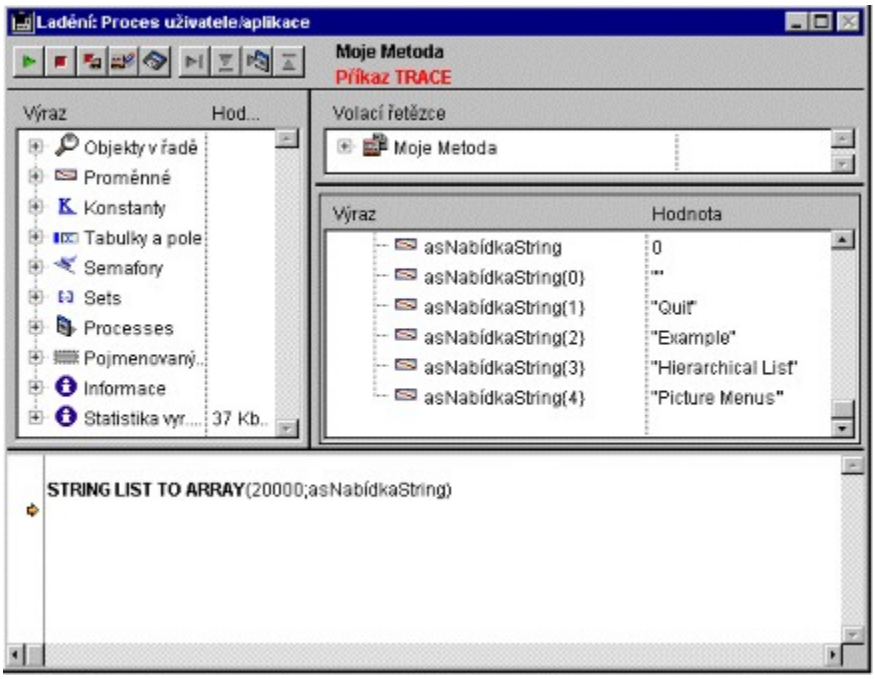


Pokud máte 4D Desktop, podívejte se do *Příručky 4D Insider* pro podrobnější informace o tom, co jsme nyní udělali. Dále pro informace o používání odkazů na zdroje v nabídkách a v objektech formuláře si přečtěte *Příručku návrháře 4th Dimension.*

Příkazy 4D pro zdroje vám umožní použít zdroje vytvořené 4D Insiderem. Následující metoda používá STRING LIST TO ARRAY k načtení STR# zdroje do array:



V okně Ladění se můžete podívat, že array je naplněno řetězci přeloženými ve 4D Insider.



Dále si přečtěte

Zdroje.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Open resource file (Otevřít zdrojový soubor)

Příkazy a odkazy pro Zdroje

Verze 6.0

Open resource file (resSoubor{; SouborTyp}) → DocRef

Parametr	Typ		Popis
resSoubor	Řetězec	→	Krátký nebo dlouhý název zdrojového souboru, nebo prázdný řetězec pro otevření okna Otevřít soubor
SouborTyp	Řetězec	→	Mac OS typ souboru (4-znaky), nebo Windows přípona souboru (1- do 3-znaky), nebo není-li nic uvedeno, soubor zdrojů ("res " / .RES)
Výsledek funkce	DocRef	←	Číslo odkazu na zdrojový soubor

Popis

Příkaz Open resource file otevře zdrojový dokument, jehož název nebo cestu jste zadali do parametru resSoubor.

Pokud předáte název souboru, může být soubor nalezen ve složce kde je struktura databáze. K otevření zdrojového souboru který je umístěný v jiné složce, musíte vložit cestu k souboru.

Pokud do resSoubor předáte prázdný řetězec, příkaz otevře standardní okno Otevřít soubor. Zde může uživatel vybrat zdrojový soubor. Samozřejmě může toto okno zrušit, ale nebude otevřený žádný zdrojový soubor: do DocRef není doplněno nic a proměnná OK je nastavena na 0.

Po správném otevření zdrojového souboru, Open resource file vrátí číslo odkazu na zdrojový soubor a doplní do proměnné OK 1. Jestliže zdrojový soubor neexistuje a nebo soubor, který jste chtěli otevřít není zdrojový soubor, je generována chyba.

Pokud na Macintoshi použijete dialog Otevřít soubor, jsou jako výchozí zobrazeny všechny typy souborů. Aby se zobrazili pouze určité typy, definujte je do volitelného parametru SouborTyp.

Pokud na Windows použijete dialog Otevřít soubor, jsou jako výchozí zobrazeny všechny soubory. K zobrazení pouze určitých typů, zadejte do parametru SouborTyp 1 až 3 znaky dlouhou příponu typu pro Windows nebo Macintosh soubor.

Nezapomeňte provést příkaz CLOSE RESOURCE FILE po skončení práce se zdrojovým souborem. Tyto soubory se taký uzavřou při ukončení aplikace a při jejím zapnutí je třeba je znovu otevřít.

Narozdíl od příkazu Open docement, který otevře dokument (datová část na Macintoshi) s výhradími právy pro čtení-psaní, Open resource file vám ale nebude bránit otevřít jeden zdrojový soubor více sekcemi 4D. Pokud se například pokusíte otevřít jeden dokument dvakrát pomocí Open document, je generována chyba při druhém pokusu. Ale pokud se pokusíte otevřít zdrojový soubor pomocí Open resource file, příkaz vrátí číslo odkazu na již otevřený soubor. Pokud provedete příkaz Open resource file několikrát, musíte pak příkaz LOSE RESOURCE FILE provést pro každý výskyt otevřeného zdrojového souboru. Tato pravidla platí pouze u jedné aplikace 4D. Pokud se pokusíte otevřít zdrojový soubor otevřený jinou aplikací, bude generována chyba.

Tato možnost vícero otevření vám umožní jednoduše získávat čísla odkazů na zdrojové soubory aplikace 4D a databáze bez pletení se do normálních operací 4D (příklad 5 a 6).

UPOZORNĚNÍ

• Buďte opatrní, při zpřístupňování zdrojového souboru aplikace 4D. NEMĚŇTE nic v tomto zdrojovém souboru, protože by jste tím mohli způsobit poškození programu nebo systémovou chybu. Nezapomeňte že vaše databáze může být použita s různými prostředími (4D, Runtime, 4D Engine, 4D Server a 4D Client).
• Pokud zpřístupňujete zdroje databáze a chcete k nim programově přidávat, mazat je měnit, otestujte si prostředí ve kterém pracujete. Ve 4D Server můžete způsobit různé chyby. Například pokud změníte zdroje na zdroji serveru

(metodou databáze nebo uloženou procedurou), ovlivníte tím systém serveru, který distribuuje zdroje na jednotlivé stanice.Nezapomeňte že na 4D Client nemáte přímý přístup ke struktuře databáze: je umístěna na serveru.
• Proto když používáte zdroje, je lepší si je uložit do vlastních souborů.
• Při práci s vlastními zdroji NEPOUŽÍVEJTE záporná čísla pro ID zdrojů; jsou rezervovány pouze pro operační systém. Zároveň nepoužívejte ID mezi 1...14999; tento rozsah je vyhrazen pro účely 4th Dimension. Pro vaše vlastní zdroje můžete použít čísla mezi 15000 a 32767. Jakmile otevřete zdrojový soubor, bude tento soubor první v řetězu zdrojových souborů, který se bude prohledávat.Poud do tohoto zdroje předáte s ID vyhrazenými pro 4D nebo systém, tyto zdroje budou nalezeny jako první příkazy jako třeba GET RESOURCE nebo vnitřními rutinami 4th Dimension. Může to být cíl, kterého jste chtěli dosáhnout, ale pokud si tím nejste jisti, raději nepoužívejte tento způsob protože může vést k systémovým chybám.
• Zdrojové soubory jsou vysoce strukturované a nemohou obsahovat více než 2700 zdrojů na soubor. Pokud pracujete se soubory obsahujícími velké množství zdrojů,je dobré si otestovat jejich počet před přídáváním dalších.

Po otevření zdrojového souboru, můžete analyzovat jeho obsah s použitím příkazů RESOURCE TYPE LIST a RESOURCE LIST.

Příklady

1. Následující příklad se pokusí otevřít, na Windows, zdrojový soubor "Mpřevd.res" umístěné ve složce databáze.

```
$vhResFile:=Open resource file ("Mpředv";"res ")
```

Na Macintoshi budete otevírat soubor "Mpředv".

1. Následující příklad se pokusí otevřít na Windows, zdrojový soubor "Mpřevd.rsr" umístěné ve složce databáze.

```
$vhResFile:=Open resource file ("Mpředv";"rsr ")
```

Na Macintoshi budete otevírat soubor "Mpřevd".

3. Následující příklad zobrazí dialog Otevřít soubor ukazující všechny typy souborů:

```
$vhResFile:= Open resource file ("")
```

4. Následující příklad zobrazí dialog Otevřít soubor ukazující soubory vytvořené příkazem Create resource file, s použitím výchozího typu souborů:

```
$vhResFile:=Open resource file("";"res ")
If (OK=1)
   ALERT("Byl otevřen ""+Document+"".")
   CLOSE RESOURCE FILE($vhResFile)
End if
```

5. Následující příklad vrací číslo odkazu na zdrojový soubor struktury do $vhStructureResFile:

```
If (Na windows )
   $vhStructureResFile:=Open resource file(Replace string(Structure file;".4DB";".RSR"))
Else
   $vhStructureResFile:=Open resource file(Structure file)
End if
```

6. Následující příklad vrací číslo odkazu na zdrojový soubor 4th Dimension do $vhApplResFile:

```
If (Na windows )
   $vhApplResFile:=Open resource file(Replace string(Application file;".EXE";".RSR"))
Else
```

```
    $vhApplResFile:=Open resource file(Application file)
End if
```

Dále si přečtěte

CLOSE RESOURCE FILE, Create resource file, Zdroje.

Systémové proměnné a Sady

Pokud je zdrojový soubor správně otevřen, je proměnná OK nastavena na 1. Pokud zdrojový soubor nemůže být otevřen, nebo uživatel klepne na tlačítko Storno v dialogu Otevřít soubor, je proměnná OK nastavena na 0.

Jestliže je soubor otevřen pomocí dialogu Otevřít soubor, je proměnná Document nastavena na cestu k souboru.

Ovládání chyb

Pokud nemůže být zdrojový soubor otevřen kvůli zdrojům nebo I/O chybě, je generována chyba. Tuto chybu můžete zachytit pomocí metody na zachytávání chyb instalované ON ERR CALL.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Create resource file (Vytvořit zdrojový soubor)

Příkazy a odkazy pro Zdroje

Verze 6.0

Create resource file (resSoubor {; SouborTyp}) → DocRef

Parametr	Typ		Popis
resSoubor	Řetězec	→	Krátký nebo dlouhý název zdrojového souboru, nebo prázdný řetězec pro otevření okna Otevřít soubor
SouborTyp	Řetězec	→	Mac OS typ souboru (4-znaky), nebo Windows přípona souboru (1- do 3-znaky), nebo není-li nic uvedeno, soubor zdrojů ("res " / .RES)
Výsledek funkce	DocRef	←	Číslo odkazu na zdrojový soubor

Popis

Create resource file vytvoří a otevře nový zdrojový soubor, jehož název a cestu zadáte do parametru resSoubor.

Pokud předáte pouze název souboru, bude tento soubor umístěn ve složce databáze. Vložte cestu, pokud chcete aby byl soubor umístěn jinde.

Pokud již soubor s takovým názvem existuje a není otevřen, příkaz jej přepíše novým zdrojovým souborem. Pokud je tento soubor otevřený, vznikne I/O chyba.

Pokud do parametru resSoubor předáte prázdný řetězec, je zobrazen dialog Uložit soubor. Můžete zde vybrat umístění a název nového zdrojového souboru. Pokud tento dialog zrušíte zdrojový soubor není vytvořen a do proměnné OK je dosazeno 0.

Pokud je zdrojový soubor správně vytvořen a otevřen, Create resource file vrátí číslo odkazu a do proměnné OK dosadí 1. Pokud nemůže být zdrojový soubor vytvořen, vznikne chyba.

Na Macintoshi je výchozí typ vytvořený příkazem Create resource file "res". Na Windows je výchozí přípona ".res".

K vytvoření souboru jiného typu:

• Na Macintoshi vložte typ souboru do volitelného parametru SouborTyp.
• Na Windows vložte do SouborTyp 1 až 3 znaky přípony Windows nebo Macintosh typu souboru mapované pomocí MAP FILE TYPES.

Nezapomeňte provést příkaz CLOSE RESOURCE FILE pro zdrojový soubor. Nicméně pokud skončíte aplikaci (nebo otevřete jinou databázi), 4th Dimension automaticky uzavře všechny zdrojové soubory otevřené pomocí Create resource file nebo Open resource file.

Příklady

1. Následující příklad se pokusí vytvořit a otevřít, na Windows, zdrojový soubor "Mpřevd.res" umístěné ve složce databáze.

```
$vhResFile:=Create resource file ("Mpředv")
```

Na Macintoshi budete otevírat soubor "Mpřevd".

1. Následující příklad se pokusí vytvořit a otevřít, na Windows, zdrojový soubor "Mpřevd.rsr" umístěné ve složce databáze.

```
$vhResFile:=Create resource file ("Mpředv";"rsr ")
```

Na Macintoshi budete otevírat soubor "Mpřevd".

3. Následující příklad zobrazí dialog Uložit soubor:

```
$vhResFile:=Create resource file("";"res ")
If (OK=1)
   ALERT("Byl vytvořen ""+Document+""")
   CLOSE RESOURCE FILE($vhResFile)
End if
```

Dále si přečtěte

CLOSE RESOURCE FILE, ON ERR CALL, Open resource file, Zdroje.

Systémové proměnné a Sady

Pokud je zdrojový soubor správně vytvořen a otevřen, je proměnná OK nastavena na 1. Pokud zdrojový soubor nemůže být vytvořen, nebo uživatel klepne na tlačítko Storno v dialogu Uložit soubor, je proměnná OK nastavena na 0.

Jestliže je soubor vytvořen pomocí dialogu Uložit soubor, je proměnná Document nastavena na cestu k souboru.

Ovládání chyb

Pokud nemůže být zdrojový soubor vytvořen nebo otevřen kvůli zdrojům nebo I/O chybě, je generována chyba. Tuto chybu můžete zachytit pomocí metody na zachytávání chyb instalované ON ERR CALL.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CLOSE RESOURCE FILE (ZAVŘÍT ZDROJOVÝ SOUBOR)

Verze 6.0

CLOSE RESOURCE FILE (resSoubor)

Parametr	Typ		Popis
resSoubor	DocRef	→	Číslo odkazu na zdrojový soubor

Popis

Příkaz CLOSE RESOURCE FILE uzavře zdrojový soubor, jehož číslo odkazu předáte jako parametr.

Pokud máte otevřen jeden zdrojový soubor vícekrát, tento příkaz vypne vždy pouze jeden výskyt.

Pokud použijete příkaz CLOSE RESOURCE FILE na zdrojový soubor aplikace nebo databáze, příkaz to zjistí a neudělá nic.

Pokud předáte špatné číslo odkazu na zdrojový soubor, příkaz neprovede nic.

Nezapomeňte provádět příkaz CLOSE RESOURCE FILE pro každý soubor otevřený pomocí Create resource file a Open resource file. Avšak všimněte si, že při vypnutí aplikace (nebo otevření jiné databáze) 4th Dimension automaticky uzavře všechny otevřené zdrojové soubory.

Příklad

Následující příklad vytvoří zdrojový soubor,vloží do něj zdrojový řetězec a opět soubor uzavře.

```
$vhDokumRef:=Create resource file("Just a file")
If (OK=1)
   SET STRING RESOURCE(20000;"Just a string";$vhDokumRef)
   CLOSE RESOURCE FILE($vhDokumRef)
End if
```

Dále si přečtěte

Create resource file, Open resource file.

Systémové proměnné a Sady

Nic není ovlivněno.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

RESOURCE TYPE LIST (SEZNAM TYPŮ ZDROJŮ)

Příkazy a odkazy pro Zdroje

Verze 6.0

RESOURCE TYPE LIST (resTypy {; resSoubor})

Parametr	Typ		Popis
resTypy	Řetězcový array	→	Seznam dostupných typů zdrojů
resSoubor	DocRef	→	Číslo odkazu na zdrojový soubor, nebo všechny otevřené zdroje, pokud vynecháno

Popis

Příkaz RESOURCE TYPE LIST naplní array resTypy dostupnými typy zdrojů z otevřených zdrojových souborů.

Pokud do parametru resSoubor předáte platné číslo odkazu na zdrojový soubor, bude použit pouze tento soubor a array resTypy se naplní typy zdrojů z tohoto souboru. Pokud tento parametr nepředáte, budou použity všechny otevřené zdrojové soubory.

Typ array resTypy můžete předem určit na Text nebo řetězec. Pokud to neuděláte, příkaz jej sám vytvoří jako Text array.

Po dokončení příkazu můžete otestovat počet typů zdrojů pomocí příkazu Size of array použitého na array resTypy.

Příklady

Následující příklad naplní array atResType typy zdrojů, které jsou ve všech otevřených zdrojových souborech:

```
RESOURCE TYPE LIST (atResType)
```

2. Následující příklad vám řekne, jestliže vaše Macintosh struktura databáze používá staré 4D plug-iny, které bude potřeba obnovit pro použití databáze na Windows:

```
$vhResFile:=Open resource file(Structure file)
RESOURCE TYPE LIST(atResType;$vhResFile)
If (Find in array(atResType;"4DEX")>0)
   ALERT("Tato databáze obsahuje starý typ Mac OS 4D plug-ins."+
   (Char(13)*2)+ " Aby jste mohli databázi používat i na Windows, tak je musíte obnovit,.")
End if
```

Poznámka: Soubor struktury není jediný kde mohou být staré verze plug-in. Databáze může také obsahovat Proc.Ext soubor.

3. Následující metoda projektu vrací počet zdrojů uložených ve zdrojovém souboru:

```
   ` Metoda projektu Sečíst zdroje
   ` Sečíst zdroje ( Čas ) → Long
   ` Sečíst zdroje ( DocRef ) → Počet zdrojů
C_LONGINT($0)
C_TIME($1)
$0:=0
RESOURCE TYPE LIST($atResType;$1)
For ($vlElem;1;Size of array($atResType))
   RESOURCE LIST($atResType{$vlElem};$alResID;$atResName;$1)
   $0:=$0+Size of array($alResID)
```

End for

Jakmile je tato metoda v databázi, můžete napsat:

```
$vhResFile:=Open resource file("")
If (OK=1)
   ALERT("Soubor ""+Document+"" obsahuje "+String(Sečíst zdroje ($vhResFile))+" zdroj(ů).")
   CLOSE RESOURCE FILE($vhResFile)
End if
```

Dále si přečtěte

RESOURCE TYPE LIST.

Systémové proměnné a Sady

Nic není ovlivněno.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

RESOURCE LIST (SEZNAM ZDROJŮ)

Příkazy a odkazy pro Zdroje

Verze 6.0

RESOURCE LIST (resTyp; resID; resNázvy{; resSoubor})

Parametr	Typ		Popis
resTyp	Řetězec	→	4-znaky označující typ zdroje
resID	LongInt Array	←	ID zdrojů tohoto typu zdroje
resNázvy	Řetězcové array	←	Názvy zdrojů vybraného typu
resSoubor	DocRef	→	číslo odkazu na zdrojový soubor, nebo pokud vynecháno všechny otevřené zdrojové soubory

Popis

Příkaz RESOURCE LIST naplní array resID a resNázvy ID a názvy zdrojů jejichž typ je zadaný do parametru resTyp.

Důležité: Do resTyp musíte vložit 4 znakový řetězec.

Pokud předáte správné číslo odkazu na zdrojový soubor, jsou použity pouze zdroje z tohoto souboru. Pokud nepoužijete parametr resSoubor, budou použity všechny otevřené zdrojové soubory.

Pokud předdefinujete array před použitím příkazu RESOURCE LIST, musíte vytvořit resID jako LongInt array a resNázvy jako Řetězec nebo Text array. Pokud tyto array nedefinujete předem, příkaz vytvoří resID jako Long Integer array a resNázvy jako Text array.

Po provedení příkazu můžete otestovat kolik zdrojů bylo nalezeno pomocí příkazu Size of array použité na resID nebo resNázvy.

Příklady

1. Následující příklad naplní array $alResID a $atResName ID a názvy ze zdroje řetězce ze zdrojového souboru struktury databáze:

```
If (Na windows )
   $vhStructureResFile:=Open resource file(Replace string(Structure file;".4DB";".RSR"))
Else
   $vhStructureResFile:=Open resource file(Structure file)
End if
If (OK=1)
   RESOURCE LIST("STR#";$alResID;$atResName;$vhStructureResFile)
End if
```

2. Následující příklad zkopíruje obrázkové zdroje ze všech otevřených zdrojových souborů do Knihovny obrázků databáze:

```
RESOURCE LIST("PICT";$alResID;$atResName)
Open window(50;50;550;120;5;"Kopíruji PICT zdroje...")
For ($vlElem;1;Size of array($alResID))
   GET PICTURE RESOURCE($alResID{$vlElem};$vgPicture)
   If (OK=1)
      $vsName:=$atResName{$vlElem}
      If ($vsName="")
```

```
            $vsName:="PICT resID="+String($alResID{$vlElem})
         End if
         ERASE WINDOW
         GOTO XY(2;1)
         MESSAGE("Vkládám obrázek ""+$vsName+"" do Knihovny obrázků.")
         SET PICTURE TO LIBRARY($vgPicture;$alResID{$vlElem};$vsName)
      End if
   End for

   CLOSE WINDOW
```

Dále si přečtěte

RESOURCE TYPE LIST.

Systémové proměnné a Sady

Nic není ovlivněno.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

STRING LIST TO ARRAY (SEZNAM ŘETĚZCŮ DO ARRAY)

Verze 6.0

STRING LIST TO ARRAY (resID; strArray{; resSoubor})

Parametr	Typ		Popis
resID	Číslo	→	Číslo ID zdroje
strArray	Řetězcový array	→	Řetězec nebo Text array k přijetí
		←	řetězců ze zdroje STR#
resSoubor	DocRef	→	Číslo odkazu na zdrojový soubor, nebo pokud vynecháno všechny otevřené zdrojové soubory

Popis

Příkaz STRING LIST TO ARRAY naplní řetězcový array hodnotami uloženými ve zdroji STR# jehož ID zadáte do resID.

Jestliže není tento zdroj nalezen, zůstane array nezměněná a proměnná OK je nastavena na 0.

Pokud předáte správné číslo odkazu na zdrojový soubor, jsou použity pouze zdroje z tohoto souboru. Pokud nepoužijete parametr resSoubor, budou použity všechny otevřené zdrojové soubory.

Pokud předdefinujete array před použitím příkazu STRING LIST TO ARRAY, musíte vytvořit strArray jako Řetězec nebo Text array. Pokud tyto array nedefinujete předem, příkaz vytvoří strArray jako Text array.

Poznámka: Každý řetězec ve zdroji seznamu řetězců může obsahovat až 255 znaků.

Příklad

Podívejte se na příklad u příkazu ARRAY TO STRING LIST.

Dále si přečtěte

ARRAY TO STRING LIST, Get indexed string, Get string resource, Get text resource.

Systémové proměnné a Sady

Pokud je zdroj nalezen OK je nastavena na 1. Jinak je nastavena na 0.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ARRAY TO STRING LIST (ARRAY DO SEZNAMU ŘETĚZCŮ)

Příkazy a odkazy pro Zdroje

Verze 6.0

ARRAY TO STRING LIST (řatězec; resID{; resSoubor})

Parametr	Typ		Popis
strArray	Řetězcový array	→	Řetězec nebo Text array k odeslání do zdroje STR#
resID	Číslo	→	Číslo ID zdroje
resSoubor	DocRef	→	Číslo odkazu na zdrojový soubor, nebo pokud vynecháno platný zdrojový soubor

Popis

Příkaz ARRAY TO STRING LIST vytvoří nebo přepíše zdroj seznam řetězců ("STR#"), jehož ID je předáno do resID. Obsah zdroje je vytvořen z řetězce v array. Array může být text nebo řetězec.

Pokud zdroj nemůže být předán,je proměnná OK nastavena na 0 (nula).

Pokud předáte správné číslo odkazu na zdrojový soubor, jsou použity pouze zdroje z tohoto souboru. Pokud nepoužijete parametr resSoubor, budou použity všechny otevřené zdrojové soubory.

Poznámka: Každý řetězec ve zdroji seznamu řetězců může obsahovat až 255 znaků.

Příklad

Vaše databáze je závislá na předáném seznamu písmen.

V metodě databáze Při ukončení napište následující kód:

```
      ` Metoda databáze Při ukončení
   If (<>vbFontsAreOK)
      FONT LIST($atFont)
      $vhResFile:=Open resource file("FontSet")
      If (OK=1)
         ARRAY TO STRING LIST($atFont;15000;$vhResFile)
         CLOSE RESOURCE FILE($vhResFile)
      End if
   End if
```

Do metody databáze Při spuštění vložte následující:

```
      ` Metoda databáze Při spuštění
   <>vbFontsAreOK:=False
   FONT LIST($atNewFont)
   If (Test path name ("FontSet")#Is a document)
      $vhResFile:=Create resource file("FontSet")
   Else
      $vhResFile:=Open resource file("FontSet")
   End if
   If (OK=1)
      STRING LIST TO ARRAY(15000;$atOldFont;$vhResFile)
```

```
      If (OK=1)
         <>vbFontsAreOK:=True
         For($vlElem;1;Size of array($atNewFont))
            If ($atNewFont{$vlElem}#$atOldFont{$vlElem})
               $vlElem:=MAXLONG
               <>vbFontsAreOK:=False
            End if
         End for
      Else
         <>vbFontsAreOK:=True
      End if
      CLOSE RESOURCE FILE($vhResFile)
   End if
   If(Not(<>vbFontsAreOK))
      CONFIRM("Nepoužíváte stejné nastavení písma, OK?")
      If(OK=1)
         <>vbFontsAreOK:=True
      Else
         QUIT 4D
      End if
   End if
```

Dále si přečtěte

SET STRING RESOURCE, SET TEXT RESOURCE, STRING LIST TO ARRAY.

Systémové proměnné a Sady

Pokud byl zdroj zapsán, OK je nastavena na 1. Jinak je nastavena na 0.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Get indexed string (Získat indexovaný řetězec)

Verze 6.0

Get indexed string (resID; strID{; resSoubor}) → Řetězec

Parametr	Typ		Popis
resID	Číslo	→	Číslo ID zdroje
strID	Číslo	→	Číslo řetězce ve zdroji
resSoubor	DocRef	→	Číslo odkazu na zdrojový soubor, nebo pokud vynecháno všechny otevřené zdrojové soubory
Výsledek funkce	Řetězec	←	Obsah indexovaného řetězce

Popis

Příkaz Get indexed string vrací jeden ze řetězců uložených ve zdroji "STR#". Číslo zdroje je zadané do parametru resID

Číslo řetězce ve zdroji je v parametru strID. Řetězce ve zdroji jsou čísovány od 1 do N. K získání všech zdrojů ze zdroje seznamu řetězců použijte příkaz STRING LIST TO ARRAY.

Pokud není zdroj nebo řetězec ve zdroji nalezen, je vrácen prázdný řetězec a do proměnné OK je dosazeno 0.

Pokud předáte správné číslo odkazu na zdrojový soubor, jsou použity pouze zdroje z tohoto souboru. Pokud nepoužijete parametr resSoubor, budou použity všechny otevřené zdrojové soubory.

Poznámka: Každý řetězec ve zdroji seznamu žetězců může obsahovat až 255 znaků.

Příklad

Podívejte se na příklad u příkazu Month of.

Dále si přečtěte

Get string resource, Get text resource, STRING LIST TO ARRAY.

Systémové proměnné a Sady

Pokud byl zdroj nalezen, je do proměnné OK dosazeno 1, jinak je OK 0.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Get string resource (Získat řetězec ze zdroje)

Příkazy a odkazy pro Zdroje

Verze 6.0

Get string resource (resID{; resSoubor}) → Řetězec

Parametr	Typ		Popis
resID	Číslo	→	Číslo ID zdroje
resSoubor	DocRef	→	Číslo odkazu na zdrojový soubor, nebo pokud vynecháno všechny otevřené zdrojové soubory
Výsledek funkce	Řetězec	←	Obsah zdroje STR

Popis

Příkaz Get string resource vrací řetězec ze zdroje ("STR ") jehož ID je předáno do resID.

Pokud zdroj není nalezen,je vrácen prázdný řetězec a proměnné OK je nastavena na 0 (nula).

Pokud předáte správné číslo odkazu na zdrojový soubor, jsou použity pouze zdroje z tohoto souboru. Pokud nepoužijete parametr resSoubor, budou použity všechny otevřené zdrojové soubory.

Příklad

Následující příklad zobrazí obsah zdroje řetězce ID=20911, který musí být umístěný v platném otevřeném zdrojovém souboru:

```
ALERT ( Get string resource (20911))
```

Dále si přečtěte

Get indexed string, Get text resource, SET STRING RESOURCE, STRING LIST TO ARRAY.

Systémové proměnné a Sady

Pokud byl zdroj nalezen, je do proměnné OK dosazeno 1, jinak je OK 0.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET STRING RESOURCE (NASTAVIT ŘETĚZEC DO ZDROJE)

Verze 6.0

SET STRING RESOURCE (resID; resData{; resSoubor})

Parametr	Typ		Popis
resID	Číslo	→	Číslo ID zdroje
resData	Řetězec	→	Nový obsah zdroje STR
resSoubor	DocRef	→	Číslo odkazu na zdrojový soubor, nebo pokud vynecháno platný zdrojový soubor

Popis

Příkaz SET STRING RESOURCE vytvoří nebo přepíše řetězec ("STR ") zdroje jehož ID je předáno do resID řetězcem předáným do resData.

Pokud zdroj nemůže být přidán, proměnná OK je nastavena na 0 (nula).

Pokud předáte správné číslo odkazu na zdrojový soubor, jsou použity pouze zdroje z tohoto souboru. Pokud nepoužijete parametr resSoubor, vloží se zdroj do vrchního zdroje (poslední otevřený).

Poznámka: Každý řetězec ve zdroji řetězců může obsahovat až 255 znaků.

Dále si přečtěte

Get strinf resource, SET TEXT RESOURCE.

Systémové proměnné a Sady

Pokud byl zdroj správně zapsán, je do proměnné OK dosazeno 1, jinak je OK 0.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Get text resource (Získat text ze zdroje)

Příkazy a odkazy pro Zdroje

Verze 6.0

Get text resource (resID{; resSoubor}) → Text

Parametr	Typ		Popis
resID	Číslo	→	Číslo ID zdroje
resSoubor	DocRef	→	Číslo odkazu na zdrojový soubor, nebo pokud vynecháno všechny otevřené zdrojové soubory
Výsledek funkce	Text	←	Obsah zdroje Text

Popis

Příkaz Get text resource vrátí text uložený ve zdroji ("TEXT") jehož ID je předáno do resID.

Pokud zdroj není nalezen, je vrácen prázdný text a proměnná OK je nastavena na 0 (nula).

Pokud předáte správné číslo odkazu na zdrojový soubor, jsou použity pouze zdroje z tohoto souboru. Pokud nepoužijete parametr resSoubor, budou použity všechny otevřené zdrojové soubory.

Poznámka: Textový zdroj může obsahovat až 32000 znaků.

Příklad

Následující příklad zobrazí obsah textového zdroje ID=20800, který musí být umístěný v platném otevřeném zdrojovém souboru:

```
ALERT ( Get text resource (20911))
```

Dále si přečtěte

Get indexed string, Get string resource, SET TEXT RESOURCE, STRING LIST TO ARRAY.

Systémové proměnné a Sady

Pokud byl zdroj nalezen, je do proměnné OK dosazeno 1, jinak je OK 0.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET TEXT RESOURCE (NASTAVIT TEXT DO ZDROJE)

Verze 6.0

SET TEXT RESOURCE (resID; resData{; resSoubor})

Parametr	Typ		Popis
resID	Číslo	→	Číslo ID zdroje
resData	Řetězec	→	Nový obsah zdroje TEXT
resSoubor	DocRef	→	Číslo odkazu na zdrojový soubor, nebo pokud vynecháno platný zdrojový soubor

Popis

Příkaz SET TEXT RESOURCE vytvoří nebo přepíše text ("TEXT ") zdroje jehož ID je předáno do resID textem předáným do resData.

Pokud zdroj nemůže být přidán, proměnná OK je nastavena na 0 (nula).

Pokud předáte správné číslo odkazu na zdrojový soubor, jsou použity pouze zdroje z tohoto souboru. Pokud nepoužijete parametr resSoubor, vloží se zdroj do vrchniho zdroje (poslední otevřený).

Poznámka: Textový zdroj může obsahovat až 32000 znaků.

Příklad

Podívejte se na příklad u příkazu SET STRING RESOURCE.

Dále si přečtěte

Get text recource, SET STRING RESOURCE.

Systémové proměnné a Sady

Pokud byl zdroj správně zapsán, je do proměnné OK dosazeno 1, jinak je OK 0.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GET PICTURE RESOURCE (ZÍSKAT OBRÁZEK ZE ZDROJE)

Verze 6.0

GET PICTURE RESOURCE (resID; resData{; resSoubor})

Parametr	Typ		Popis
resID	Číslo	→	Číslo ID zdroje
resData	Pole nebo Prom.	→	Obrázkové pole nebo proměnná
		←	pro přijetí obsahu zdroje PICT
resSoubor	DocRef	→	Číslo odkazu na zdrojový soubor, nebo pokud vynecháno všechny otevřené zdrojové soubory

Popis

Příkaz GET PICTURE RESOURCE vrací do obrázkového pole nebo proměnné resData obrázek uložený ve zdroji ("PICT") jehož ID je zadáno v resID.

Pokud zdroj není nalezen, parametr resData zůstane nezměněný a proměnná OK je nastavena na 0 (nula).

Pokud předáte správné číslo odkazu na zdrojový soubor, jsou použity pouze zdroje z tohoto souboru. Pokud nepoužijete parametr resSoubor, budou použity všechny otevřené zdrojové soubory.

Poznámka: Obrázkový zdroj může mít až několik megabytů.

Příklad

Podívejte se na příklad u příkazu RESOURCE LIST.

Dále si přečtěte

GET ICON RESOURCE, ON ERR CALL, SET PICTURE RESOURCE.

Systémové proměnné a Sady

Pokud byl zdroj nalezen, je do proměnné OK dosazeno 1, jinak je OK 0.

Ovládání chyb

Pokud není dostatek paměti k načtení obrázku, je generována chyba. Tuto chybu můžete zachytit pomocí metody instalované příkazem ON ERR CALL.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET PICTURE RESOURCE (NASTAVIT OBRÁZEK DO ZDROJE)

Verze 6.0

SET PICTURE RESOURCE (resID; resData{; resSoubor})

Parametr	Typ		Popis
resID	Číslo	→	Číslo ID zdroje
resData	Řetězec	→	Nový obsah zdroje PICT
resSoubor	DocRef	→	Číslo odkazu na zdrojový soubor, nebo pokud vynecháno platný zdrojový soubor

Popis

Příkaz SET PICTURE RESOURCE vytvoří nebo přepíše obrázkový ("PICT") zdroj jehož ID je v resID.

Pokud zdroj nemůže být přidán, proměnná OK je nastavena na 0 (nula).

Pokud předáte správné číslo odkazu na zdrojový soubor, jsou použity pouze zdroje z tohoto souboru. Pokud nepoužijete parametr resSoubor, vloží se zdroj do vrchniho zdroje (poslední otevřený).

Pokud do parametru resData předáte prázdné pole nebo proměnnou, příkaz neprovede nic a proměnná OK se nastaví na 0.

Poznámka: Obrázkový zdroj může mít až několik megabytů.

Dále si přečtěte

GET PICTURE RESOURCE.

Systémové proměnné a Sady

Pokud byl zdroj správně zapsán, je do proměnné OK dosazeno 1, jinak je OK 0.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GET ICON RESOURCE (ZÍSKAT IKONU ZE ZDROJE)

Příkazy a odkazy pro Zdroje

Verze 6.0

GET ICON RESOURCE (resID; resData{; resSoubor})

Parametr	Typ		Popis
resID	Číslo	→	Číslo ID zdroje
resData	Pole nebo Prom.	→	Obrázkové pole nebo proměnná
		←	pro přijetí obsahu zdroje cicn
resSoubor	DocRef	→	Číslo odkazu na zdrojový soubor, nebo pokud vynecháno všechny otevřené zdrojové soubory

Popis

Příkaz GET ICON RESOURCE vrátí, do obrázkového pole nebo proměnné resData, ikonu uloženou ve zdroji barevných ikon ("cicn") jehož ID je v parametru resID.

Pokud zdroj není nalezen, parametr resData je vrácen nezměněný a proměnná OK je nastavena na 0 (nula).

Pokud předáte správné číslo odkazu na zdrojový soubor, jsou použity pouze zdroje z tohoto souboru. Pokud nepoužijete parametr resSoubor, budou použity všechny otevřené zdrojové soubory.

Příklad

Následující příklad načte, do obrázkového array, barevné ikony umístěné v aplikaci 4D:

```
If (Na windows )
   $vh4DResFile:=Open resource file(Replace string(Application file;".EXE";".RSR"))
Else
   $vh4DResFile:=Open resource file(Application file)
End if
RESOURCE LIST("cicn";$alResID;$asResName;$vh4DResFile)
$vlNbIcons:=Size of array($alResID)
ARRAY PICTURE(ag4DIcon;$vlNbIcons)
For ($vlElem;1;$vlNbIcons)
   GET ICON RESOURCE($alResID{$vlElem};ag4Dicon{$vlElem}; $vh4DResFile)
End for
```

Po provedení této metody, bude array vypadat takto po zobrazení ve formuláři:

Dále si přečtěte

GET PICTURE RESOURCE.

Systémové proměnné a Sady

Pokud byl zdroj nalezen, je do proměnné OK dosazeno 1, jinak je OK 0.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GET RESOURCE (ZÍSKAT OBSAH ZDROJE)

Příkazy a odkazy pro Zdroje

Verze 6.0

GET RESOURCE (resTyp; resID; resData{; resSoubor})

Parametr	Typ		Popis
resTyp	Řetězec	→	4-znaky označující typ zdroje
resID	Číslo	→	Číslo ID zdroje
resData	BLOB	→	BLOB pole nebo proměnná k přijetí dat
		←	Obsah zdroje
resSoubor	DocRef	→	Číslo odkazu na zdrojový soubor, nebo pokud vynecháno všechny otevřené zdrojové soubory

Popis

Příkaz GET RESOURCE vrací do BLOB pole nebo proměnné resData obsah zdroje jehož typ a ID zadáte do parametrů resTyp a resID.

Důležité: Musíte vložit 4 znaky dlouhý řetězec do resTyp.

Pokud zdroj není nalezen, parametr resData zůstane nezměněný a proměnná OK je nastavena na 0 (nula).

Pokud předáte správné číslo odkazu na zdrojový soubor, jsou použity pouze zdroje z tohoto souboru. Pokud nepoužijete parametr resSoubor, budou použity všechny otevřené zdrojové soubory.

Poznámka: Zdroj může být až několik megabytů velký.

Nezávislost na platformě: Nezapameňte že pracujete se základními zdroji pro MacOS. Nezáleží na tom, na jaké platformě, ale vnitřní zdrojová data jako Long Integer jsou ukládány v Macintosh pořadí bytů. Na Windows je u dat pro standardní zdroje (jako řetězcové seznamy a obrázkové zdroje), pokud je potřeba, přehozeno pořadí bytů. Na druhou stránku při vytváření a používání vašich vnitřních datových struktur, je pouze na vás jaké zvolíte pořadí bytů (např. předáním Macintosh byte ordering do příkazu jako je BLOB to longint).

Příklad

Podívejte se na příklad u příkazu SET RESOURCE.

Dále si přečtěte

Příkazy BLOB, Zdroje, SET RESOURCE.

Systémové proměnné a Sady

Pokud byl zdroj nalezen, je do proměnné OK dosazeno 1, jinak je OK 0.

Ovládání chyb

Pokud není dostatek paměti k načtení zdroje, je generována chyba. Tuto chybu můžete zachytit pomocí metody instalované příkazem ON ERR CALL.

SET RESOURCE (NASTAVIT DO ZDROJE)

Příkazy a odkazy pro Zdroje

Verze 6.0

SET RESOURCE (resTyp; resID; resData{; resSoubor})

Parametr	Typ		Popis
resTyp	Řetězec	→	4-znaky označující typ zdroje
resID	Číslo	→	Číslo ID zdroje
resData	BLOB	→	nový obsah zdroje
resSoubor	DocRef	→	Číslo odkazu na zdrojový soubor, nebo pokud vynecháno platný zdrojový soubor

Popis

Příkaz SET RESOURCE vytvoří nebo přepíše zdroj jehož typ a ID jsou předány do parametrů resTyp a resID.

Důležité: Musíte vložit 4 znaky dlouhý řetězec do resTyp.

Pokud zdroj nemůže být přidán, proměnná OK je nastavena na 0 (nula).

Pokud předáte správné číslo odkazu na zdrojový soubor, jsou použity pouze zdroje z tohoto souboru. Pokud nepoužijete parametr resSoubor, vloží se zdroj do vrchniho zdroje (poslední otevřený).

Poznámka: Zdroj může být několik až několik megabytů velký.

Nezávislost na platformě: Nezapomeňte, že pracujete se zdroji základnými pro MacOS. Nezáleží na tom, na jaké platformě ale vnitřní zdrojová data jako Long Integer jsou ukládány v Macintosh pořadí bytů. Na Windows je u dat pro standardní zdroje (jako řetězcové seznamy a obrázkové zdroje), pokud je potřeba, přehozeno pořadí bytů. Na druhou stránku při vytváření a používání vašich vnitřních datových struktur, je pouze na vás jaké zvolíte pořadí bytů (např. předáním Macintosh byte ordering do příkazu jako je LONGINT TO BLOB).

Příklad

Během práce s 4D mohou někteří uživatelé používat různá nastavení v meziprocesních proměnných. Můžete:
1: Použít příkazy SAVE VARIABLES a LOAD VARIABLES k uložení a načtení proměnných z dokumentu na disku.
2. Použít příkazy VARIABLE TO BLOB, BLOB TO DOCUMENT, DOCUMENT TO BLOB a BLOB TO VARIABLE k uložení a načtení proměnných z BLOBu na disku.
3. Použít příkazy VARIABLE TO BLOB, SET RESOURCE, GET RESOURCE a BLOB TO VARIABLE k uložení a načtení proměnných ze zdrojového souboru na disku.

Následující příklad je metoda k třetímu způsobu. Do metody databáze Při ukončení vložíte:

```
   ` Metoda databáze Při ukončení
If (Test path name ("DB_Prefs")#Is a document)
   $vhResFile:=Create resource file("DB_Prefs")
Else
   $vhResFile:=Open resource file("DB_Prefs")
End if
If (OK=1)
   VARIABLE TO BLOB(<>vbAutoRepeat;$vxPrefData)
   VARIABLE TO BLOB(<>vlCurTable;$vxPrefData;*)
   VARIABLE TO BLOB(<>asDfltOption;$vxPrefData;*)
```

```
        ` a tak dále...
    SET RESOURCE("PREF";26500;$vxPrefData;$vhResFile)
    CLOSE RESOURCE FILE($vhResFile)
End if
```

Do metody databáze Při spuštění vložíte:

```
    ` Metoda databáze Při spuštění
C_BOOLEAN(<>vbAutoRepeat)
C_LONGINT(<>vlCurTable)
$vbDone:=False
$vhResFile:=Open resource file("DB_Prefs")
If (OK=1)
    GET RESOURCE("PREF";26500;$vxPrefData;$vhResFile)
    If (OK=1)
        $vlPosun:=0
        BLOB TO VARIABLE($vxPrefData;<>vbAutoRepeat;$vlPosun)
        BLOB TO VARIABLE($vxPrefData;<>vlCurTable;$vlPosun)
        BLOB TO VARIABLE($vxPrefData;<>asDfltOption;$vlPosun)
            ` a tak dále...
        $vbDone:=False
    End if
    CLOSE RESOURCE FILE($vhResFile)
End if
If(Not($vbDone))
    <>vbAutoRepeat:=False
    <>vlCurTable:=0
    ARRY STRING(127;<>asDfltOption;0)
End if
```

Dále si přečtěte

BLOB Commands, GET RESOURCE.

Systémové proměnné a Sady

Pokud byl zdroj správně zapsán, je do proměnné OK dosazeno 1, jinak je OK 0.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Get resource name (Získat název zdroje)

Příkazy a odkazy pro Zdroje

Verze 6.0

Get resource name (resTyp; resID{; resSoubor}) → Řetězec

Parametr	Typ		Popis
resTyp	Řetězec	→	4-znaky označující typ zdroje
resID	Číslo	→	Číslo ID zdroje
resSoubor	DocRef	→	Číslo odkazu na zdrojový soubor, nebo pokud vynecháno všechny otevřené zdrojové soubory
Výsledek funkce	Řetězec	←	Název zdroje

Popis

Příkaz Get resource name vrací název zdroje jehož typ a ID jste zadali do parametrů *resTyp* a *resID*.

Pokud zdroj není nalezen, parametr resData zůstane nezměněný a proměnná OK je nastavena na 0 (nula).

Pokud předáte správné číslo odkazu na zdrojový soubor, jsou použity pouze zdroje z tohoto souboru. Pokud nepoužijete parametr resSoubor, budou použity všechny otevřené zdrojové soubory.

Příklad

Následující metoda projektu zkopíruje všechny zdroje, včetně jejich názvů a vlastností z jednoho zdrojového souboru do druhého:

```
    ` Metoda projektu KOPÍROVAT ZDROJE
    ` KOPÍROVAT ZDROJE ( Řetězec ; Long ; Čas ; Čas )
    ` KOPÍROVAT ZDROJE ( resTyp ; resID ; zdrResSoubor ; cílResSoubor )
C_STRING (4;$1)
C_LONGINT ($2)
C_TIME ($3;$4)
C_BLOB ($vxResData)
GET RESOURCE ($1;$2;$vxData;$3)
If (OK=1)
   SET RESOURCE ($1;$2;$vxData;$4)
   If (OK=1)
      SET RESOURCE NAME ($1;$2; Get resource name ($1;$2;$3);$4)
      SET RESOURCE PROPERTIES ($1;$2; Get resource properties ($1;$2;$3);$4)
   End if
End if
```

Jakmile je tato metoda projektu uložena do vaší databáze, můžete napsat:

KOPÍROVAT ZDROJE ("DATA";15000;$vhResSoubA;$vhResSoubB)

Dále si přečtěte

SET RESOURCE PROPERTIES.

Systémové proměnné a Sady

Proměnná OK je nastavena na 0 jestliže zdroj neexistuje, jinak je 1.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET RESOURCE NAME (NASTAVIT NÁZEV ZDROJE)

Verze 6.0

SET RESOURCE NAME (resTyp; resID; resNázev{; resSoubor})

Parametr	Typ		Popis
resTyp	Řetězec	→	4-znaky označující typ zdroje
resID	Číslo	→	Číslo ID zdroje
resNázev	Řetězec	→	Nový název zdroje
resSoubor	DocRef	→	Číslo odkazu na zdrojový soubor, nebo pokud vynecháno platný zdrojový soubor

Popis

Příkaz SET RESOURCE NAME změní název zdroje jehož typ a ID předáte do parametrů resTyp a resID.

Pokud předáte správné číslo odkazu na zdrojový soubor, jsou použity pouze zdroje z tohoto souboru. Pokud nepoužijete parametr resSoubor, budou použity všechny otevřené zdrojové soubory.

Pokud zdroj není nalezen, příkaz neprovede nic a proměnná OK je nastavena na 0 (nula).

UPOZORNĚNÍ: NEMĚŇTE názvy zdrojů které patří k 4th Dimension nebo k systému. Pokud to uděláte, můžete způsobit systémovou chybu.

Poznámka: Názvy zdrojů mohou být až 255 znaků dlouhé. Nejsou citlivé na velká a malá písmena.

Příklad

Podívejte se na příklad u příkazu Get resource name.

Dále si přečtěte

SET RESOURCE PROPERTIES.

Systémové proměnné a Sady

Proměnná OK je nastavena na 0 jestliže zdroj neexistuje, jinak je 1.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Get resource properties (Získat vlastnosti zdroje)

Příkazy a odkazy pro Zdroje

Verze 6.0

Get resource properties (resTyp; resID{; resSoubor}) → Číslo

Parametr	Typ		Popis
resTyp	Řetězec	→	4-znaky označující typ zdroje
resID	Číslo	→	Číslo ID zdroje
resSoubor	DocRef	→	Číslo odkazu na zdrojový soubor, nebo pokud vynecháno všechny otevřené zdrojové soubory
Výsledek funkce	Číslo	←	Vlastnosti zdroje

Popis

Příkaz Get resource properties vrací vlastnosti zdroje, jehož typ a ID jsme zadali do parametrů resTyp a resID.

Pokud předáte správné číslo odkazu na zdrojový soubor, jsou použity pouze zdroje z tohoto souboru. Pokud nepoužijete parametr resSoubor, budou použity všechny otevřené zdrojové soubory.

Pokud zdroj není nalezen, příkaz vrátí 0 a proměnná OK je nastavena na 0 (nula).

Na číselnou hodnotu vrácenou příkazem Get resource properties je třeba se dívat jako na bitovou hodnotu, kde každý bit má svůj účel. Pro vysvětlení všech těchto nastavení si přečtěte popis u příkazu SET RESOURCE PROPERTIES.

Příklad

Přečtěte si příklad u příkazu Get resource name.

Dále si přečtěte

SET RESOURCE NAME.

Systémové proměnné a Sady

Proměnná OK je nastavena na 0 jestliže zdroj neexistuje, jinak je 1.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET RESOURCE PROPERTIES (NASTAVIT VLASTNOSTI ZDROJE)

Verze 6.0

SET RESOURCE PROPERTIES (resTyp; resID; resVlastn{; resSoubor})

Parametr	Typ		Popis
resTyp	Řetězec	→	4-znaky označující typ zdroje
resID	Číslo	→	Číslo ID zdroje
resVastn	Číslo	→	vlastnosti zdroje v bitové reprezentaci, k nastavení chování zdroje při zavedení aplikace
resSoubor	DocRef	→	Číslo odkazu na zdrojový soubor, nebo pokud vynecháno platný zdrojový soubor

Popis

Příkaz SET RESOURCE PROPERTIES změní vlastnosti zdroje jehož typ a ID jsou zadány do parametrů resTyp a resID.

Pokud předáte správné číslo odkazu na zdrojový soubor, jsou použity pouze zdroje z tohoto souboru. Pokud nepoužijete parametr resSoubor, budou použity všechny otevřené zdrojové soubory.

Pokud zdroj není nalezen, příkaz neprovede nic a proměnná OK je nastavena na 0 (nula).

Číselné hodnoty, které předáte do parametru resVlastn musí být považovány za bitové hodnoty u nichž má každý bit svoji hodnotu. Následující předdefinované konstanty jsou obsaženy ve 4th Dimension:

Konstanta	***Typ***	***Hodnota***
System heap resource mask	*Long Integer*	*64*
System heap resource bit	*Long Integer*	*6*
Purgeable resource mask	*Long Integer*	*32*
Purgeable resource bit	*Long Integer*	*5*
Locked resource mask	*Long Integer*	*16*
Locked resource bit	*Long Integer*	*4*
Protected resource mask	*Long Integer*	*8*
Protected resource bit	*Long Integer*	*3*
Preloaded resource mask	*Long Integer*	*4*
Preloaded resource bit	*Long Integer*	*2*
Changed resource mask	*Long Integer*	*2*
Changed resource bit	*Long Integer*	*1*

S použitím těchto konstant můžete vytvořit všechny vlastnosti zdroje. Podívejte se na následující příklad.

Vlastnosti zdrojů a jejich efekty

• System heap (systémová paměť)
Pokud je nastavena tato vlastnost, zdroj bude načten do systémové paměti místo do paměti 4D. Tuto vlastnost používejte jen pokud jste si **opravdu** jisti co děláte.

• Purgeable (uvolnitelné)
Pokud je označena tato vlastnost, tak po načtení zdroje jej můžete uvolnit z paměti pokud je místo potřebné pro jiná data. Jestli načítáte zdroje do ¨4D BLOBů, je výhodné mít všechny vaše zdroje označené jako uvolnitelné protože tím omezíte použití paměti. Nicméně pokud používáte tyto zdroje často při vaší práci, můžete je chtít mít neuvolnitelné, protože tím omezíte volání na disk a časté načítání těchto zdrojů.

• Locked (zamčeno)
Pokud je označena tato vlastnost, nebudete mít možnost změnit umístění tohoto zdroje po jeho načtení do paměti. Zamčený zdroj nemůže být uvolněn, pokud není uvolnitelný. Zamykání zdrojů má nepříjemné účinky na fragmentaci paměti. Tuto vlastnost používejte jen pokud jste si **opravdu** jisti co děláte.

• Protected (chráněno)
Pokud je tato vlastnost označena, nemůžete měnit název zdroje, ID nebo obsah. Nemůžete jej ani vymazat. Pokud znovu provedete příkaz SET RESOURCE PROPERTIES a odstraníte tuto možnost, budete moci zdroj opět měnit a vymazat. Tuto vlastnost většinou používat nebudete. Poznámka: Tato vlastnost nemá žádný efekt na Windows.

• Preloaded (předem načtený)
Pokud je označena tato vlastnost, je zdroj automaticky načten do paměti ve chvíli kde je zdrojový soubor otevřen. Tato vlastnost je využitelná při optimalizaci načítání zdrojů při otevření zdrojového souboru. Tuto vlastnost většinou používat nebudete.

• Changed (změněn)
Pokud je označena tato vlastnost, je zdroj označen jako "musí být uložen na disk" když je uzavřen zdrojový soubor ve kterém je umístěn. Pak příkaz SET RESOURCE bude ovládat zapisování a přepisování zdroje vnitřně. Tuto vlastnost používejte jen pokud jste si **opravdu** jisti co děláte.

Obvykle budete používat vlasnosti uvolnitelné a vyjímečně předem načtený a chráněno.

UPOZORNĚNÍ: NEMĚŇTE vlastnosti zdrojů 4D a systému. Můžete tím způsobit systémovou chybu.

Příklady

1. Podívejte se na příklad u příkazu Get resource name.

2. Následující příklad udělá zdroj "STR#" ID=17000 uvolnitelný a ostatní vlastnosti nechá nezměněné.

```
$vlResAttr:=Get resource properties ('STR#';17000;$vhResFile)
SET RESOURCE PROPERTIES('STR#';17000;$vlResAttr ?+ Purgeable resource bit;$vhMyResFile)
```

3. Následující příklad udělá zdroj 'STR#' ID=17000 neuvolnitelný a předem načtený:

```
SET RESOURCE PROPERTIES('STR#';17000;Preloaded resource mask;$vhResFile)
```

4. Následující příklad udělá zdroj 'STR#' ID=17000 předem načtený a uvolnitelný:

```
SET RESOURCE PROPERTIES('STR#';17000;Preloaded resource mask+Purgeable resource mask;
$vhResFile)
```

Dále si přečtěte

SET RESOURCE NAME.

Systémové proměnné a Sady

Proměnná OK je nastavena na 0 jestliže zdroj neexistuje, jinak je 1.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

DELETE RESOURCE (VYMAZAT ZDROJ)

Příkazy a odkazy pro Zdroje

Verze 6.0

DELETE RESOURCE (resTyp; resID{; resSoubor})

Parametr	Typ		Popis
resTyp	Řetězec	→	4-znaky označující typ zdroje
resID	Číslo	→	Číslo ID zdroje
resSoubor	DocRef	→	Číslo odkazu na zdrojový soubor, nebo pokud vynecháno platný zdrojový soubor

Popis

Příkaz DELETE RESOURCE vymaže zdroj jehož typ a ID jste zadali do parametrů resTyp a resID.

Pokud předáte správné číslo odkazu na zdrojový soubor, jsou použity pouze zdroje z tohoto souboru. Pokud nepoužijete parametr resSoubor, budou použity všechny otevřené zdrojové soubory.

Pokud zdroj neexistuje, DELETE RESOURCE neprovede nic a nastaví proměnnou OK na 0. Jestliže je zdroj nalezen a vymazán, je proměnná OK nastavena na 1.

UPOZORNĚNÍ: NEMAŽTE zdroje 4D a systému. Můžete tím způsobit systémovou chybu.

Příklady

1: Následující příklad vymaže zdroj "STR#" ID=20000:

```
   ` Povšimněte si že tento příklad vymaže první "STR#" ID=20000 zdroj
   ` nalezený ve všech otevřených zdrojích:
DELETE RESOURCE ("STR#";20000)
```

2. Následující příklad vymaže zdroj "STR#" ID=20000, jestliže bude nalezen ve specifickém souboru:

```
   ` Všiměte si, že tento příklad vymaže "STR#" ID=20000 pouze
   ` jestliže je tento zdroj v platném otevřeném souboru:
DELETE RESOURCE ("STR#";20000;$vhResSoub)
   ` Pokud v souboru definovaném v $vhResSoub jsou jiné zdroje,
   ` zůstanou tyto zdroje nezměněné.
```

3. Metoda projektu VYMAZAT ZDROJE TYPU vymaže všechny zdroje zadaného typu (druhý parametr) z definovaného souboru (prví parametr):

```
   ` Metoda projektu VYMAZAT ZDROJE TYPU
   ` VYMAZAT ZDROJE TYPU ( Čas ; Řetězec )
   ` VYMAZAT ZDROJE TYPU ( resSoubor ; resTyp )
C_TIME($1)
C_STRING(4;$2)
RESOURCE LIST($2;$aiResID;$asResName;$1)
If(OK=1)
   For($vlElem;1;Size of array($aiResID))
      DELETE RESOURCE($2;$aiResID{$vlElem};$1)
   End for
End if
```

Po uložení této metody do databáze můžete napsat:

```
` Vymazat všechny zdroje typu "PREF" ze zdrojového souboru $vhResFile
VYMAZAT ZDROJE TYPU ($vhResFile;"PREF")
```

4. Metoda projektu VYMAZAT ZDROJ S NÁZVEM vymaže zdroje (definovaného typu) jehož název je známý:

```
` Metoda projektu VYMAZAT ZDROJ S NÁZVEM
` VYMAZAT ZDROJ S NÁZVEM ( Čas ; Řetězec ; Řetězec)
` VYMAZAT ZDROJ S NÁZVEM ( resSoubor ; resTyp ; resNázev )
C_TIME($1)
C_STRING(4;$2)
C_STRING(255;$3)
RESOURCE LIST($2;$aiResID;$asResName;$1)
If(OK=1)
   $vlElem:=Find in array($asResName;$3)
   If($vlElem>0)
      DELETE RESOURCE($2;$aiResID{$vlElem};$1)
   End if
End if
```

Po uložení této metody do databáze můžete napsat:

```
`Vymazat ze zdr. souboru $vhResFile zdroje typu "PREF" s názvem "Standardní nastavení":
VYMAZAT ZDROJ S NÁZVEM ($vhResFile;"PREF";"Standardní nastavení")
```

Dále si přečtěte

RESOURCE LIST, SET RESOURCE PROPERTIES.

Systémové proměnné a Sady

Proměnná OK je nastavena na 0 jestliže zdroj neexistuje. Pokud byl zdroj vymazán je proměnná OK nastavena na 1.

Příkazy a odkazy pro Zdroje

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ALL RECORDS (VŠECHNY ZÁZNAMY)

Verze 3

ALL RECORDS ({tabulka})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ve které vybrat všechny záznamy, nebo výchozí tabulka pokud vynecháno

Popis

ALL RECORDS vybere všechny záznamy z tabulky pro platný proces. První záznam ve výběru bude platný záznam a bude načten. ALL RECORDS vrací záznamy ve výchozím pořadí jak jsou uloženy na disku, což je pořadí ve kterém byli zadávány.

Příklad

Následující příklad zobrazí všechny záznamy z tabulky [Lidé]:

```
ALL RECORDS ([Lidé]) ` Vybere všechny záznamy v tabulce
DISPLAY SELECTION ([Lidé]) ` Zobrazí záznamy ve výstupním formuláři
```

Dále si přečtěte

DISPLAY SELECTION, MODIFY SELECTION, ORDER BY, QUERY, Records in selection, Records in table.

Příkazy a odkazy pro Výběr

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Records in selection (Záznamů ve výběru)

Příkazy a odkazy pro Výběr

Verze 3

Records in selection ({tabulka}) → Číslo

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro kterou vrátit počet záznamů ve výběru, nebo výchozí tabulka, pokud vynecháno
Výsledek funkce	Číslo	←	Počet záznamů ve výběru tabulky

Popis

Records in selection vrací počet záznamů v platném výběru tabulky. Naproti tomu vrací Records in table celkový počet záznamů v tabulce.

Příklad

Následující příklad ukáže opakovací techniku nejčastěji používanou pro pohyb všemi záznamy ve výběru. Stejná akce může být provedena také příkazem APPLY TO SELECTION:

```
FIRST RECORD ([Lidé]) ` Start at first record in the selection
For ($vlZaznam; 1; Records in selection ([Lidé])) ` Opakovat pro každý záznam
   Něco udělat ` Něco provést se záznamem
   NEXT RECORD ([Lidé]) ` Přesunout na další záznam
End for
```

Dále si přečtěte

Records in table.

Příkazy a odkazy pro Výběr

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DELETE SELECTION (VYMAZAT VÝBĚR)

Příkazy a odkazy pro Výběr

Verze 3

DELETE SELECTION ({tabulka})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro kterou vymazat výběr záznamů, nebo výchozí tabulka, pokud vynecháno

Popis

DELETE SELECTION vymaže platný výběr záznamů tabulky. Pokud je platný výběr prázdný, příkaz neprovede nic. Po vymazání záznamů je platný výběr prázdný. Záznamy které jsou vymazány během transakce jsou uzamčeny pro ostatní uživatele a procesy, dokud není transakce potvrzena nebo zrušena.

Upozornění: Vymazání výběru záznamů je trvalá operace, která nelze vrátit.

Vlastnost Mazat úplně v okně Vlastnosti tabulky vám umožní urychlit mazání pomocí DELETE SELECTION.

Příklady

1. Následující příklad vybere všechny záznamy v tabulce a umožní uživateli vybrat,které záznamy chce vymazat. Příklad má dvě části. První je metoda pro zobrazení záznamů a druhá je metoda objektu pro tlačítko Vymazat. Zde je první metoda:

```
ALL RECORDS ([Lidé]) ` Vybrat všechny záznamy
OUTPUT FORM ([Lidé]; "Listing") ` Nastavit formulář pro zobrazení
DISPLAY SELECTION ([Lidé]) ` Zobrazit záznamy
```

Následující je metoda objektu pro tlačítko Vymazat, které je v zápatí výstupního formuláře. Metoda používá záznamy označené uživatelem (UserSet) k vymazání výběru. Nezapomeňte, že pokud uživatel neoznačí žádné záznamy, příkaz neprovede nic:

```
    ` Potvrdit že uživatel opravdu chce záznamy vymazat
CONFIRM("Bylo označeno "+String(Records in set ("UserSet"))+
    " lidí k vymazání."+Char(13)+"Klepněte na OK k jejich vymazání.")
If (OK=1)
    USE SET ("UserSet") ` Použít záznamy vybrané uživatelem
    DELETE SELECTION([Lidé]) ` Vymazat výběr záznamů
End if
ALL RECORDS ([Lidé]) ` Vybrat všechny záznamy
```

2. Pokud je během mazání příkazem DELETE SELECTION objeven zamčený záznam, tento záznam není vymazán. Všechny zamčené záznamy jsou přesunuty do sady LockedSet. Po provedení příkazu DELETE SELECTION můžete otestovat sadu LockedSet a podívat jestli byly nějaké záznamy zamčené. Následující smyčka bude probíhat tak dlouho, dokud nebudou vymazány všechny záznamy:

```
Repeat ` Opakovat pro všechny zamčené záznamy
    DELETE SELECTION([TatoTabulka])
    USE SET ("LockedSet") ` Vybrat zamčené záznamy
Until (Records in set("LockedSet")=0) ` Dokud není žádný zamčený záznam
```

Dále si přečtěte

DISPLAY SELECTION, MODIFY SELECTION, Zamykání záznamů, Sady.

Příkazy a odkazy pro Výběr

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Selected record number (Pořadí záznamu ve výběru)

Příkazy a odkazy pro Výběr

Verze 3

Selected record number ({tabulka}) → Číslo

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro kterou získat číslo záznamu, nebo výchozí tabulka, pokud vynecháno
Výsledek funkce	Číslo	←	Číslo označeného záznamu

Popis

Selected record number vrací pozici platného záznamu v platném výběru tabulky.

Jestliže výběr není prázdný a platný záznam je ve výběru, vrátí Selected record number číslo mezi 1 a Records in selection. Jestliže je výběr prázdný nebo není žádný platný záznam,bude vráceno 0 (nula).

Selected record number není to samé jako Record number, který vrací fyzické číslo záznamu. Číslo označeného záznamu závisí na platném výběru a platném záznamu.

Příklad

Následující příklad uloží číslo označeného záznamu do proměnné:

```
ČísPlatOznZáz:=Selected record number ([Lidé]) ` Získat číslo označeného záznamu
```

Dále si přečtěte

O číslech záznamů, GOTO SELECTED RECORD, Records in selection.

Příkazy a odkazy pro Výběr

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GOTO SELECTED RECORD (JÍT NA ZÁZNAM VE VÝBĚRU)

Příkazy a odkazy pro Výběr

Verze 3

GOTO SELECTED RECORD ({tabulka;} záznam)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka na kterou jít pro označený záznam, nebo platná tabulka, pokud vynecháno
záznam	Číslo	→	Pozice záznamu ve výběru

Popis

GOTO SELECTED RECORD posune na určený záznam v platném výběru tabulky a tento záznam udělá platným záznamem. Platný výběr se nebude měnit. Parametr záznam není totéž jako číslo vracené funkcí Record number; udává pozici záznamu v platném výběru. Pozice záznamu záleží na tom jak je vytvořený platný výběr a jestli je tříděný.

Pokud v platném výběru nejsou žádné záznamy nebo číslo není ve výběru, příkaz GOTO SELECTED RECORD neprovede nic.

Příklad

Následující příklad načte data z pole [Lidé]Příjmení do atPrijmeni. Long Ineger array alCislZazn je naplněn čísly která označují čísla záznamů ve výběru. Obě array jsou pak setříděny:

```
    ` Zde vytvořit výběr pro tabulku [Lidé]
    ` ...
    ` Získat příjmení
SELECTION TO ARRAY ([Lidé]Příjmení;atPrijmeni)
    ` Vytvořit array pro čísla záznamů ve výběru
$vlPocZazn:=Size of array (atPrijmeni)
ARRAY LONGINT (alCislZazn;$vlPocZazn)
For ($vlZaznam; 1; $vlPocZazn)
   alCislZazn{$vlZaznam}:=$vlZaznam
End for
    ` Třídit array abecedně
SORT ARRAY (atPrijmeni; alCislZazn; >)
```

Pokud je array atPrijmeni zobrazeno v posuvné oblasti, může uživatel klepnout na jeden řádek. Pokud jsou array setříděny synchronizovaně, bude prvek v array alCislZazn obsahovat číslo záznamu pro záznam pro prvek který byl označen v array atPrijmeni.

Následující metoda objektu pro atPrijmeni označí správný záznam v tabulce [Lidé] podle toho jaký vyberete v posuvné oblasti:

```
Case of
   ÷ (Form event=On Clicked)
      If (atPrijmeni#0)
         GOTO SELECTED RECORD (alCislZazn{atPrijmeni})
      End if
End case
```

Dále si přečtěte

Selected record number.

Příkazy a odkazy pro Výběr

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

FIRST RECORD (PRVNÍ ZÁZNAM)

Příkazy a odkazy pro Výběr

Verze 3

FIRST RECORD ({tabulka})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ve které jít na první záznam, nebo výchozí tabulka pokud vynecháno

Popis

FIRST RECORD udělá první záznam platného výběru platným záznamem pro tabulku a načte tento záznam z disku. Všechny příkazy pro dotazy, výběry a třídění také dělají z prvního záznamu výběru platný záznam. Pokud je platný výběr prázdný, příkaz nebude mít žádný efekt.

Tento příkaz je často používán po příkazu USE SET k započetí prohlížení záznamů od prvního záznamu. Nicméně můžete tento příkaz volat i z podprogramu, pokud si nejste jisti zda je platný záznam první záznam výběru.

Příklad

Následující příklad udělá první záznam tabulky [Zákazníci] platným záznamem:

```
FIRST RECORD ([Zákazníci])
```

Dále si přečtěte

Before selection, End selection, LAST RECORD, NEXT RECORD, PREVIOUS RECORD.

Příkazy a odkazy pro Výběr

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

NEXT RECORD (DALŠÍ ZÁZNAM)

Příkazy a odkazy pro Výběr

Verze 3

NEXT RECORD ({tabulka})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ve které jít na další záznam, nebo výchozí tabulka pokud vynecháno

Popis

NEXT RECORD udělá další záznam platného výběru platným záznamem pro tabulku a načte tento záznam z disku. Pokud je platný výběr prázdný, příkaz nebude mít žádný efekt.

Jestliže NEXT RECORD přesune ukazatel na záznam za konec výběru, vrátí End selection hodnotu True a ukazatel není přesunut. Pokud End selection vrátí True, použijte příkazy FIRST RECORD, LAST RECORD nebo GOTO SELECTED RECORD pro přesunutí ukazatele na platný záznam zpět do platného výběru.

Příklad

Podívejte se na příklad u příkazu DISPLAYSELECTION.

Dále si přečtěte

Before selection, End selection, FIRST RECORD, LAST RECORD, PREVIOUS RECORD.

Příkazy a odkazy pro Výběr

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

LAST RECORD (POSLEDNÍ ZÁZNAM)

Verze 3

LAST RECORD ({tabulka})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ve které jít na poslední záznam, nebo výchozí tabulka pokud vynecháno

Popis

LAST RECORD udělá poslední záznam platného výběru platným záznamem pro tabulku a načte tento záznam z disku. Pokud je platný výběr prázdný, příkaz nebude mít žádný efekt.

Příklad

Následující příklad ukládá poslední záznam tabulky [Zákazníci] platným záznamem:

```
LAST RECORD ([Zákazníci])
```

Dále si přečtěte

Before selection, End selection, FIRST RECORD, NEXT RECORD, PREVIOUS RECORD.

Příkazy a odkazy pro Výběr

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

PREVIOUS RECORD (PŘEDCHOZÍ ZÁZNAM)

Verze 3

PREVIOUS RECORD ({tabulka})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ve které jít na předchozí záznam, nebo výchozí tabulka pokud vynecháno

Popis

PREVIOUS RECORD udělá předchozí záznam platného výběru platným záznamem pro tabulku a načte tento záznam z disku. Pokud je platný výběr prázdný nebo pokud End Selection nebo Before selection vrátí True, příkaz nebude mít žádný efekt.

Jestliže PREVIOUS RECORD přesune ukazatel na záznam před začátek výběru, vrátí Before selection hodnotu True a ukazatel není přesunut. Pokud Before selection vrátí True, použijte příkazy FIRST RECORD, LAST RECORD nebo GOTO SELECTED RECORD pro přesunutí ukazatele na platný záznam zpět do platného výběru.

Dále si přečtěte

Before selection, End selection, FIRST RECORD, LAST RECORD, NEXT RECORD.

Příkazy a odkazy pro Výběr

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Before selection (Před výběrem)

Příkazy a odkazy pro Výběr

Verze 3

Before selection ({tabulka}) → Logické

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro kterou testovat, jestli je ukazatel na platný záznam před prvním záznamem výběru, nebo výchozí tabulka pokud vynecháno
Výsledek funkce	Logické	←	Ano (True) nebo Ne (False)

Popis

Before selection vrátí True, jestliže je ukazatel na platný záznam před prvním záznamem platného výběru pro tabulku. Tento příkaz se často používá k testování jestli PREVIOUS RECORD posunul ukazatel na platný záznam před první záznam platného výběru. Pokud je platný výběr prázdný, Before selection True.

K přesunutí ukazatele na platný záznam zpět do výběru použijte příkazy LAST RECORD, FIRST RECORD nebo GOTO SELECTED RECORD. NEXT RECORD nevrátí ukazatel zpět do výběru.

Before selection vrátí True také v prvním záhlaví při tisku zprávy pomocí PRINT SELECTION nebo z nabídky Tisk. K testování prvního záhlaví můžete použít následující kód a vytisknout pro první stránku jiné záhlaví:

```
   ` Metoda formuláře použitého pro tisk souhrnné zprávy
$vpFormTabulky:=Current form table
Case of
      ` ...
   ÷ (Form event=On Header)
         ` Bude se tisknout záhlaví
      Case of
         ÷ (Before selection($vpFormTabulky→))
            ` Kód pro první záhlaví bude zde
            ` ...
      End case
End case
```

Příklad

Tato metoda formuláře je použita během tisku zprávy. Nastaví proměnnou vTitle která se bude tisknout v oblasti záhlaví první stránky:

```
   ` Metoda formuláře ` [Finance];"Souhrn"
Case of
      ` ...
   ÷ (Form event=On Header)
      Case of
         ÷ (Before selection([Finance))
            vTitle := "Celková zpráva pro rok 1999" ` Nastavit titul první stránky
      Else
         vTitle := "" ` Odstranit titul pro ostatní stránky
```

```
            End case
        End case
```

Dále si přečtěte

End selection, FIRST RECORD, Form event, PREVIOUS RECORD, PRINT SELECTION.

Příkazy a odkazy pro Výběr

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

End selection (Za výběrem)

Příkazy a odkazy pro Výběr

Verze 3

End selection ({tabulka}) → Logické

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro kterou testovat, jestli je ukazatel na platný záznam za posledním záznamem výběru, nebo výchozí tabulka pokud vynecháno
Výsledek funkce	Logické	←	Ano (True) nebo Ne (False)

Popis

End selection vrátí True, jestliže je ukazatel na platný záznam za posledním záznamem platného výběru pro tabulku. Tento příkaz se často používá k testování jestli NEXT RECORD posunul ukazatel na platný záznam za posledním záznamem platného výběru. Pokud je platný záznam prázdný, vrátí End selection True.

K přesunutí ukazatele na platný záznam zpět do výběru použijte příkazy LAST RECORD, FIRST RECORD nebo GOTO SELECTED RECORD. PREVIOUS RECORD nevrátí ukazatel zpět do výběru.

End selection vrátí True také v posledním zápatí při tisku zprávy pomocí PRINT SELECTION nebo z nabídky Tisk. K testování posledního zápatí můžete použít následující kód a vytisknout pro poslední stránku jiné zápatí:

```
   ` Metoda formuláře použitého pro tisk souhrnné zprávy
$vpFormTabulky:=Current form table
Case of
      ` ...
   ÷ (Form event=On Printig Footer)
         ` Bude se tisknout zápatí
      If (End selection($vpFormTabulky→))
            ` Kód pro poslední zápatí bude zde
      Else
         ` Kód pro zápatí bude zde
      End if
End case
```

Příklad

Tato metoda formuláře je použita během tisku zprávy. Nastaví proměnnou vFooter která se bude tisknout v oblasti zápatí poslední stránky:

```
   ` Metoda formuláře ` [Finance];"Souhrn"
Case of
      ` ...
   ÷ (Form event=On Printing Footer)
      If(End selection([Finance]))
         vFooter := "©1999 Acme Corp." ` Nastavit zápatí pro poslední stránku
      Else
         vFooter := "" ` Vymazat zápatí pro ostatní stránky
      End if
```

End case

Dále si přečtěte

Before selection, Form event, LAST RECORD, NEXT RECORD, PRINT SELECTION.

Příkazy a odkazy pro Výběr

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DISPLAY SELECTION (ZOBRAZIT VÝBĚR)

Příkazy a odkazy pro Výběr

Verze 6.0 (Upraveno)

DISPLAY SELECTION ({tabulka} {; *} {; *})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka k zobrazení, nebo pokud vynecháno výchozí tabulka
*		→	Použít výstupní formulář pro výběr jednoho záznamu a skrýt posuvníky ve vstupním formuláři
*		→	Ukázat posuvníky ve vstupním formuláři (přepíše druhou volbu první volitelné *)

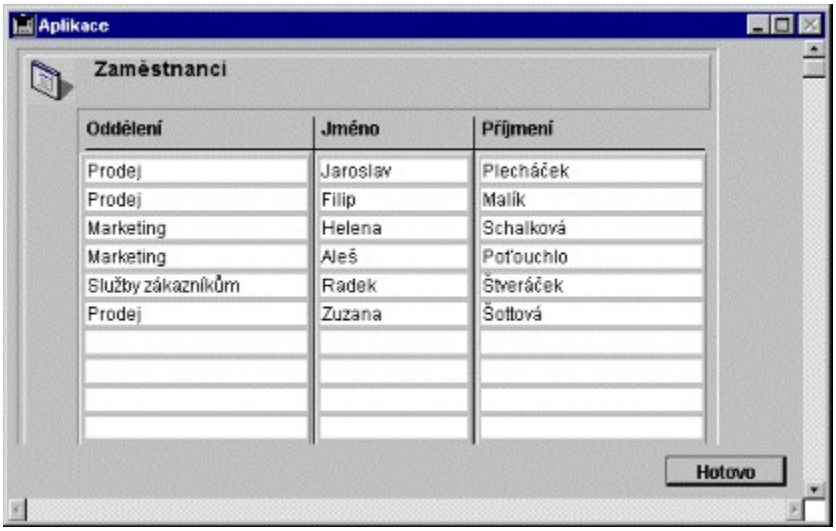
Popis

Příkaz DISPLAY SELECTION zobrazí výběr tabulky ve výstupním formuláři. Záznamy jsou zobrazeny v posuvném seznamu stejném jako v Prostředí uživatele. Když uživatel poklepe na záznam, je záznam zobrazen ve vstupním formuláři. Seznam je zobrazen v okně na popředí.

Pokud chcete mít možnost po poklepání na záznam měnit tento záznam ve vstupním formuláři (jak je možno v prostředí uživatele), použijte příkaz MODIFY SELECTION místo DISPLAY SELECTION.

Všechny následující informace jsou použitelné na oba příkazy, mimo informací o upravování záznamů.

Následující obrázek ukazuje výstupní formulář zobrazený pomocí příkazu DISPLAY SELECTION.



Po provedení příkazu DISPLAY SELECTION nebude žádný záznam platný. Použijte příkazy jako FIRST RECORD nebo LAST RECORD k jeho označení.

DISPLAY SELECTION neumožní uživateli měnit výběr záznamů. MODIFY SELECTION to umožní.

Některá pravidla patřící k volitelnému parametru *:
- Pokud výběr obsahuje pouze jeden záznam a není použita *, je tento záznam zobrazen ve vstupním formuláři.
- Pokud je zadaný první parametr *, je výběr jednoho záznamu zobrazen ve výstupním formuláři.
- Pokud je zadaný první volitelný parametr * a uživatel zobrazí záznam do vstupního formuláře pomocí poklepání na záznam, budou ve vstupním formuláři skryty posuvníky. K obrácení tohoto efktu použijte druhý volitelný parametr

*.

Tlačítko s popiskou Hotovo je automaticky zobrazeno v zápatí formuláře. Předáním proměnné nebo aktivního objektu do formuláře odstraníte toto tlačítko. Klepnutí na toto tlačítko ukončí provádění příkazu. Můžete místo tohoto tlačítka vložit vlastní tlačítka; umístěte je do oblastí záhlaví výstupního formuláře. Můžete použít automatické akce Přijmout a Zrušit nebo použít metody objektu, které budou volat příkazy ACCEPT a CANCEL.

Uživatel může posunovat seznamem záznamů a označit libovolný záznam. Když uživatel klepne na jiný záznam, je první záznam odznačen a druhý je označen. Uživatel může označit skupinu záznamů tím, že klepne na první záznam a Shift+Klepne na poslední záznam. K označení záznamů, které nejdou po sobě, může uživatel použít Ctrl+Klepnutí (Windows) nebo Command+Klepnutí (Macintosh) na jednotlivé záznamy.

Během a po provádění příkazu DISPLAY SELECTION jsou označené záznamy předány do sady s názvem UserSet. UserSet je během zobrazování dostupná pro metody objektů tlačítek nebo pro vybrané položky nabídek. Je také dostupný pro metodu projektu která provedla příkaz DISPLAY SELECTION.

Příklady

1. Následující příklad vybere všechny záznamy z tabulky [Lidé]. Pak použije příkaz DISPLAY SELECTION k zobrazení tohoto výběru a umožní uživateli vybrat záznamy k tisku Nakonec použije vybrané záznamy pomocí USE SET a vytiskne je příkazem PRINT SELECTION:

```
ALL RECORDS([Lidé]) ` Vybrat všechny záznamy
DISPLAY SELECTION ([Lidé]; *) ` Zobrazit záznamy
USE SET ("UserSet") ` Použít pouze záznamy vybrané uživatelem
PRINT SELECTION ([Lidé]) ` Vytisknout záznamy vybrané uživatelem
```

2. Podívejte se na příklad 6 u příkazu Form event. Tento příklad vám umožní testovat všechny události, které nastanou během DISPLAY SELECTION.

3. K zobrazení výsledků, třeba nabídky Dotazy v prostředí uživatele, při použití DISPLAY SELECTION a MODIFY SELECTION v prostředí Vlastních nabídek postupujte následovně:

a. V prostředí návrháře vytvořte záhlaví nabídek s položkami které chcete provést jako Všechnyzáznamy, Dotaz, Třídit.

b. Přiřaďte toto záhlaví nabídek k výstupnímu formuláři zobrazenému příkazem DISPLAY SELECTION nebo MODIFY SELECTION.

c. Přiřaďte následující metody k položkám nabídky:

```
    ` M_ZOB_VŠE (Přiřadit k položkce Zobrazit vše)
$vpCurTable:=Current form table
ALL RECORDS($vpCurTable→)
    ` M_DOTAZ (Přiřadit k položce Dotaz)
$vpCurTable:=Current form table
QUERY($vpCurTable→)
    ` M_TŘÍDIT (Přiřadit k položce Třídit)
$vpCurTable:=Current form table
ORDER BY($vpCurTable→)
```

Můžete použít i příkazy jako PRINT SELECTION, REPORT, atd., k vytvoření všech "standardních" položek nabídek které můžete potřebovat při práci s výběrem záznamů v prostředí Vlastních nabídek. Díky příkazu Current form teble jsou tyto položky generické a metody mohou být použity na jakoukoli tabulku a výstupní formulář.

Dále si přečtěte

Form event, MODIFY SELECTION, Sady.

Příkazy a odkazy pro Výběr

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

MODIFY SELECTION (UPRAVIT VÝBĚR)

Verze 6.0 (upraveno)

MODIFY SELECTION ({tabulka}{; *}{; *})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka k zobrazení a upravení, nebo pokud vynecháno výchozí tabulka
*		→	Použít výstupní formulář pro výběr jednoho záznamu a skrýt posuvníky ve vstupním formuláři
*		→	Ukázat posuvníky ve vstupním formuláři (přepíše druhou volbu první volitelné *)

Popis

MODIFY SELECTION provádí to samé jako DISPLAY SELECTION. Podívejte se na příklad DISPLAY SELECTION pro popis. Rozdíly mezi těmito příkazy jsou následující:

1.DISPLAY SELECTION vám umožní zobrazit platný výběr záznamů ve výstupním formuláři nebo ve vstupním po poklepání na záznam. S použitím MODIFY SELECTION můžete záznamy zobrazit a měnit záznam pokud na něj poklepete a není zamčený jiným procesem nebo uživatelem.

2. DISPLAY SELECTION automaticky přepne tabulku do módu pouze číst. MODIFY SELECTION automaticky přepne tabulku do módu číst/psát. Oba příkazy navrátí původní stav tabulky po svém dokončení.

Dále si přečtěte

DISPLAY SELECTION, Form event, Sady.

Příkazy a odkazy pro Výběr

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

APPLY TO SELECTION (POUŽÍT NA VÝBĚR)

Příkazy a odkazy pro Výběr

Verze 3

APPLY TO SELECTION ({tabulka;} tvrzení)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka na kterou použít tvrzení, nebo výchozí tabulka pokud vynecháno
tvrzení	Řetězec	→	Jeden řádek kódu nebo metoda

Popis

APPLY TO SELECTION použije tvrzení na každý záznam ve výběru. Tvrzení může být výraz nebo metoda. Pokud tvrzení změní záznam v tabulce, je tento záznam uložen. Pokud tvrzení nezmění záznam, záznam není uložen. Když je platný výběr prázdný příkaz neprovede nic. Jestli máte automatický vztah, může tvrzení použít pole z jiné tabulky.

APPLY TO SELECTION může získat informace z výběru (například součet) nebo změnit záznamy ve výběru (například změnit první písmeno pole na velké písmeno). Pokud je tento příkaz použit v transakci, mohou být změny vráceny po zrušení transakce.

4D Server: Server neprovádí žádné příkazy z tvrzení. Každý záznam je poslán na místní stanici a tam je upraven.

Během provádění APPLY TO SELECTION je zobrazen teploměr udávající průběh příkazu. Ke skrytí teploměru použijte příkaz MESSAGES OFF před provedením APPLY TO SELECTION. Když je teploměr zobrazen, může uživatel akci zrušit.

Příklady

1. Následující příklad změní Jména v tabulce [Zaměstnanci] na velká písmena:

```
APPLY TO SELECTION([Zaměstnanci];[ Zaměstnanci]Jméno:= Uppercase([Zaměstnanci]Jméno))
```

2. Pokud během provádění APPLY TO SELECTION je nějaký záznam uzamčený, tento záznam není změněn. Všechny zamčené záznamy jsou uloženy v sadě LockedSet. Po skončení příkazu můžete testovat tuto sadu a zjistit jestli byli nějaké zamčené záznamy. Následující smyčka bude probíhat dokud nebudou změněny všechny záznamy:

```
Repeat
   APPLY TO SELECTION([Zaměstnanci];[ Zaměstnanci]Jméno :=Uppercase([Zaměstnanci]Jméno))
   USE SET ("LockedSet") ` Vybrat pouze zamčené záznamy
Until (Records in set ("LockedSet") = 0) ` Hotovo pokud nejsou žádné zamčené záznamy
```

Systémové proměnné a Sady

Pokud uživatel klepne na tlačítko Stop v teploměru, je proměnná OK nastavena na 0, jinak je nastavena na 1.

Dále si přečtěte

Sady.

REDUCE SELECTION (OMEZIT VÝBĚR)

Příkazy a odkazy pro Výběr

Verze 3

REDUCE SELECTION ({tabulka;}počet)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka na kterou omezit výběr, nebo výchozí tabulka pokud vynecháno
počet	Číslo	→	Počet záznamů které ponechat ve výběru.

Popis

REDUCE SELECTION vytvoří nový výběr pro tabulku. Příkaz vrátí první Počet záznamů z platného výběru tabulky. REDUCE SELECTION je použit na platný výběr platného procesu. Změní platný výběr tabulky platného procesu; první záznam nového výběru bude platným záznamem.

Příklad

Následující příklad nejdříve vytvoří statistiku pro dealer z 20 zemí. Pro každý stát nalezne 3 nejlepší s obratem vyšším než $50000 a ti kteří jsou mezi 100 nejlepšími dealery dostanou cenu. S mnoha řádky kódu může být tento řádek proveden indexovaným hledáním:

```
CREATE EMPTY SET([Dealeři];"Vítězi") ` Vytvořit prázdou sadu
SCAN INDEX([Dealeři]ProdejeSoučet;100;<) ` Prohlédnout od konce indexu
CREATE SET([Dealeři];"100 Nejlepších") ` Umístit vybrané záznamy do sady
For ($Země;1;Records in table([Země])) ` Pro každý Stát
      ` Hledat dealery které v tomto státě prodávají více než za $50000
   QUERY([Dealeři];[Dealeři]Země=[Země]Název;*)
   QUERY(&;[Dealeři];[Dealeři]ProdejeSoučet>=50000)
   CREATE SET([Dealeři];"VítězníPrac") ` Vložit je do sady
      ` Mohou být ve skupině 100 nejlepších dealerů
   INTERSECTION(" VítězníPrac ";"100 Nejlepších";" VítězníPrac ")
   USE SET(" VítězníPrac ") ` Potenciální Vítězi pro Zemi
      ` Třídit výsledek v sestupném pořadí
   ORDER BY([Dealeři];[Dealeři]ProdejeSoučet;<)
   REDUCE SELECTION([Dealeři];3) ` Ponechat 3 nejlepší
   CREATE SET([Dealeři];"VítězníPrac") ` Vítězové pro stát
      ` Vložit do seznamu světových vítězů
   UNION("VítězníPrac";"Vítězi";"Vítězi")
End for
CLEAR SET("100 Nejlepších") ` Tato sada již není potřeba
CLEAR SET("VítězníPrac") ` Tato sada již není potřeba
USE SET("Vítězi") ` Zde jsou vítězové
CLEAR SET("Vítězi") ` Tato sada již není potřeba
OUTPUT FORM([Dealeři];"DopisCena") ` Vybrat dopis
PRINT SELECTION([Dealeři]) ` Tisknout dealery
```

Dále si přečtěte

ORDER BY, QUERY, SCAN INDEX, Sady.

Příkazy a odkazy pro Výběr

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SCAN INDEX (HLEDAT V INDEXU)

Příkazy a odkazy pro Výběr

Verze 3

SCAN INDEX (pole; počet {; > nebo <})

Parametr	Typ		Popis
pole	Pole	→	Indexované pole pro které prohledat index
počet	Číslo	→	Počet záznamů k vrácení
> nebo <		→	> od začátku indexu < od konce indexu

Popis

SCAN INDEX vrací výběr počtu záznamů pro tabulku. Pokud předáte <, SCAN INDEX vrátí počet záznamů z konce indexu (vysoké hodnoty). Pokud předáte >, SCAN INDEX vrátí počet záznamů ze začátku indexu (nízké hodnoty). Tento příkaz je velmi výkonný, protože pro provedení akce používá indexy.

Tento příkaz pracuje pouze s indexovanými poli. Tento příkaz mění platný výběr tabulky v platném procesu, ale neoznačí žádný platný záznam.

Pokud předáte vyšší hodnotu než je počet záznamů v tabulce, příkaz vrátí všechny záznamy v tabulce.

Příklad

Následující příklad pošle dopis 50-ti nejhorším zákazníkům a 50-ti nejlepším zákazníkům.

```
SCAN INDEX([Zákazníci]CelkemProdej;50;<) ` Získat 50 nejhorších zákazníků
ORDER BY([Zákazníci]PSČ;>) ` Třídit dle PSČ
OUTPUT FORM([Zákazníci];"ThreateningMail")
PRINT SELECTION([Zákazníci]) ` Tisknout dopisy
SCAN INDEX([Zákazníci] CelkemProdej;50;>) ` Získat 50 nejlepsích zákazníků
ORDER BY([Zákazníci]PSČ;>) ` Třídit dle PSČ
OUTPUT FORM([Zákazníci];"Thanks Letter")
PRINT SELECTION([Zákazníci]) ` Tisknout dopisy
```

Dále si přečtěte

ORDER BY, QUERY, REDUCE SELECTION.

Příkazy a odkazy pro Výběr

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ONE RECORD SELECT (VYBRAT JEDEN ZÁZNAM)

Příkazy a odkazy pro Výběr

Verze 3

ONE RECORD SELECT ({tabulka})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro kterou omezit výběr na platný záznam nebo výchozí tabulka pokud vynecháno.

Popis

ONE RECORD SELECT omezí platný výběr tabulky na platný záznam. Pokud není žádný platný záznam, příkaz neprovede nic.

Historická poznámka: Tento příkaz byl použit pro "návrat" do výběru záznamu který byl předán a vyjmut ze stack paměti záznamu ve chvíli, kdy byl upravován výběr. Ve verzi 6 vám příkaz SET QUERY DESTINATION umožní provést dotaz bez změny platného výběru tabulky; proto již nepotřebujete ukládat platný záznam do paměti z důvodu dotazu v tabulce. Tento příkaz je ale dále využitelný pokud chcete omezit výběr jen na platný záznam.

Příkazy a odkazy pro Výběr

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

HIGHLIGHT RECORDS (ZVÝRAZNIT ZÁZNAMY)

Verze 6.5

HIGHLIGHT RECORDS ({NázevSady})

Parametr	Typ		Popis
NázevSady	Řetězec	→	Sada (set) záznamů k vysvícení, nebo UserSet není-li parametr uveden

Popis

Příkaz HIGHLIGHT RECORDS vám umožní označit záznamy ve výstupním formuláři. Tato akce je stejná jako tučné označení záznamů s použitím myši nebo **Shift+Klepnutí** nebo **Ctrl+Klepnutí** (**Command+Klepnutí** na MacOs). Označené záznamy budou zvýrazněny. Platný výběr se nezmění.

Poznámka: Sada UserSet je obnovena po překreslení záznamů; to znamená po provedení vnitřního volání příkazu - ne hned po provedení příkazu HIGHLIGHT RECORDS.

- Pokud předáte správný název sady do parametru NázevSady, bude příkaz proveden na tuto sadu.
- Pokud vynecháte parametr NázevSady, příkaz zvýrazní záznamy v sadě UserSet.

Příklad

Ve výstupním formuláři zobrazeném příkazem MODIFY SELECTION, můžete chtít provést hledání bez změny platného výběru. K tomu umístěte do formuláře tlačítko **Hledat** a přiřaďte k němu následující metodu:

```
SET QUERY DESTINATION(Into Set;"UserSet")
QUERY
SET QUERY DESTINATION(Into Current Selection)
HIGHLIGHT RECORDS
```

Po klepnutí na toto tlačítko se zobrazí standardní editor dotazů. Jakmile je dotaz proveden, jsou nalezené záznamy zvýrazněny a platný výběr se nezmění.

Příkazy a odkazy pro Výběr

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Sady

Příkazy a odkazy pro Sady

Verze 6.0 (upraveno)

Sady jsou výkonný způsob jak pracovat s několika výběry. Vedle možnosti vytvořit sadu, přiřadit jí jako platný výběr a uložit, načíst a vymazat sadu vám 4th Dimension nabízí tři základníoperace se sadami:

- Průnik
- Spojení
- Rozdíl

Sady a platný výběr

Sada je kompletní zpodobnění výběru záznamů. Idea sady je úzce svázána s ideou platného výběru. Sady jsou používány hlavně z těchto důvodů:

- K uložení a opětovnému načtení výběru, pokud nezáleží na pořadí
- K zpřístupnění výběru který udělal uživatel na obrazovce (UserSet)
- K provedení logických operací mezi výběry

Platný výběr je seznam odkazů na záznamy které jsou zobrazeny. Tento seznam je v paměti. Pouze záznamy které jsou platně vybrané jsou v seznamu. Výběr neobsahuje záznamy, ale pouze odkazy na záznamy. Každý odkaz na záznam zabírá 4 byty paměti. Když pracujete s tabulkou, vždy pracujete se záznamy v platném výběru. Pokud je výběr tříděný, je pouze změněn seznam odkazů. Pro jednu tabulku je pouze jeden výběr na proces.

Stejně jako platný výběr i sada je výběr záznamů. Sada vše porovádí díky velmi kompaktnímu způsobu uchování záznamů. Každý záznam je zastoupen jedním bitem. Operace používající sady jsou velmi rychlé, protože počítač může akce na bitech provádět rychle. Sada obsahuje jeden bit pro každý záznam v tabulce, ať už je záznam v sadě nebo není. Každý bit je 1 nebo 0 podle toho jestli je v sadě nebo není.

Sady jsou velmi ekonomické pro paměť RAM. Velikost sady v bytech je počet záznamů děleno 8. Pokud například vytvoříte sadu pro tabulku která má 10000 záznamů, bude tato sada velká 1250 bytů, což je 1,2 KB v RAM.

Pro každou tabulku může být více sad. Sady jsou ukládány na disk odděleně od databáze. Pokud chcete změnit záznam náležející sadě, nejdříve použijte sadu pro platný výběr a pak změňte záznam nebo záznamy. Název meziprocesní sady musí být pro databázi jedinečný.

Sada není nikdy tříděná - záznamy jsou pouze evidovány jestli jsou nebo nejsou v sadě. Na druhou stránku pojmenovaný výběr je tříděný, ale vyžaduje více paměti. Jestli chcete vědět více informací o Pojmenovaných výběrech, přečtěte si část Pojmenované výběry.

Sada si "pamatuje", který záznam byl platný ve chvíli, kdy byla sada vytvořena. Následující tabulka porovnává platný výběr a sady:

Porovnání	***Platný výběr***	***Sady***
Počet na tabulku	*1*	*0 a více*
Tříditelné	*Ano*	*Ne*
Může být uložen na disk	*Ne*	*Ano*
RAM na záznam	*Počet záznamů * 4*	*Celkový počet záznamů/8*
Kombinovatelné	*Ne*	*Ano*
Obsahuje platný záznam	*Ano*	*Ano, ten ve chvíli vytvoření*

Jakmile vytvoříte sadu, tak patří k tabulce ve která byla vytvořena. Operace sad mohou být prováděny pouze mezi

sadami z jedné tabulky.

Sady jsou nezávislé na datech. To znamená že po změně provedených do souborů, sady již nemusí souhlasit. Existuje mnoho operací které mohou způsobit neplatnost sad. Například pokud vytvoříte sadu ze všech záznamů zákazníků z Plzně a pak jeden tento záznam změníte na "Praha", sada nebude souhlasit protože to je pouze odrazu výběru záznamů. Vymazání záznamů a jejich nahrazení jinými také změní sadu. Sady mohou být stejné pouze do té doby, než se změní data v záznamech.

Procesové a Meziprocesové Sady

Můžete mít následující tři typy sad:

• **Procesová sada**: Procesová sada může být zpřístupněna pouze z procesu ve kterém byla vytvořena. UserSet a LackedSet jsou sady procesu. Procesové sady jsou ostraněny ve chvíli, kde je skončena metoda procesu. Tyto sady nepotřebují žádnou předponu.

• **Meziprocesová sady**: Sada je meziprocesová jestli jsou před jejím názvem znaky (<>) - "Menší než" následované "větší než". **Poznámka**: Tento zápis může být použit jak na Macintoshi tak na Windows. Pouze na Macintoshi můžete použít ještě znak diamant (Option+Shift+V na US klávesnici).

• **Místní sady/Sady klienta**: Verze 6 obsahuje ještě místní/klient sady. Název této sady musí začínet znakem dolaru ($).

Sady a Transakce

Sada může být vytvořena uvnitř transakce. Je možné vytvořit sadu ze záznamů vytvořených a upravených uvnitř transakce a sadu záznamů vytvořených a upravených mimo transakci. Při ukončení transakce může být sada vytvořená během transakce vymazána, protože nebude odpovídat existujícím záznamům. Mimo případ kdy je transakce potvrzena.

Příklad Sady

Následující příklad vymaže duplikované záznamy z tabulky která obsahuje informace o lidech. Smyčka For...End for se bude přesunovat na všechny záznamy v tabulce a bude porovnávat platný záznam s předchozím záznamem. Jestli bude Jméno, adresa a PSČ stejné, pak bude záznam předán do sady. Na konci smyčky se sada stane platným výběrem a (starý) platný výběr bude vymazán:

```
CREATE EMPTY SET([Lidé];"Duplicates")
    ` Vytvořit prázdnou sagu pro duplikované záznamy
ALL RECORDS([Lidé])
    ` Vybrat všechny záznamy
    ` Třídit záznamy podle PSČ, adresy a jména
    ` a tak budou duplikované záznamy u sebe
ORDER BY ([Lidé];[Lidé]ZIP;>;[Lidé]Address;>;[Lidé]Jméno;>)
    ` Nastavit proměnné tak že budou obsahovat data z předchozího záznamu
$Jméno:=[Lidé]Jméno
$Adresa:=[Lidé]Adresa
$PSČ:=[Lidé]PSČ
    ` Jít na další záznam pro porovnání
NEXT RECORD ([Lidé])
```

```
For ($i; 2; Records in table ([Lidé]))
      ` Začít smyčku na záznamech se začátkem na 2
      `Jestli jméno, adresa, a PSČ jsou stejné
      ` pak je záznam duplikát předchozího.
   If (([Lidé]Jméno=$Jméno) & ([Lidé]Adresa=$Adresa) & ([Lidé]PSČ=$PSČ))
         ` Přidat platný záznam (duplikát) do sady
      ADD TO SET ([Lidé]; "Duplicates")
   Else
         ` Uložit adresu, jméno a PSČ tohoto záznamu
         ` pro porovnání s dalším záznamem
      $Jméno:=[Lidé]Jméno
      $Adresa:=[Lidé]Adresa
      $PSČ:=[Lidé]PSČ
   End if
      ` Přesunout na další záznam
   NEXT RECORD ([Lidé])
End for
   ` Použít duplikované záznamy které byly nalezeny
USE SET ("Duplicates")
   ` Vymazat duplikované záznamy
DELETE SELECTION ([Lidé])
   ` Odstranit sadu z paměti
CLEAR SET ("Duplicates")
```

Jako alternativu k vymazání záznamů je můžete zobrazit na obrazovku nebo vytisknout a tak budete mít lepší možnost porovnání.

Systémová sada UserSet

4th Dimension ovládá systémovou sadu UserSet. Do UserSet se automaticky ukládají záznamy zvýrazněné uživatelem na obrazovce. Takto můžete zobrazit záznamy pomocí MODIFY SELECTION nebo DISPLAY SELECTION, řícti uživateli aby označil záznamy a pak můžete z ručně vybraných záznamů vytvořit platný výběr nebo je uložit do vámi nazvané sady.

Pro jeden proces existuje pouze jedna sada UserSet. Každá tabulka nemá svůj vlastní UserSet. UserSet "patří" tabulce pokud jsou vybrané záznamy z této tabulky.

Následující metoda ukazuje jak můžete zobrazit záznamy, nechat uživatele vybrat a pak použít UserSet k zobrazení vybraných záznamů.

```
OUTPUT FORM ([Lidé]; "Display") ` Nastavit výstupní formulář
ALL RECORDS ([Lidé]) ` Vybrat všechny lidi
ALERT ("Stiskněte Ctrl nebo Command a klepnutím vyberte záznamy.")
DISPLAY SELECTION ([Lidé]) ` Zobrazit lidi
USE SET ("UserSet") ` Použít lidi kteří byli vybráni
ALERT ("Vybrali jste následující lidi.")
DISPLAY SELECTION ([Lidé]) ` Zobrazit vybrané lidi
```

Poznámka: K znovu získání UserSet musíte použít příkaz MODIFY SELECTION nebo DISPLAY SELECTION.

Systémová sady LockedSet

Příkazy APPLY TO SELECTION, ARRAY TO SELECTION a DELETE SELECTION vytvářejí při použití ve víceprocesovém prostředí sadu LockedSet. LockedSet obsahuje záznamy které byly zamčené během provádění těchto příkazů.

Dále si přečtěte

ADD TO SET, CLEAR SET, COPY SET, CREATE EMPTY SET, CREATE SET, DIFFERENCE, INTERSECTION, Is in set, LOAD SET, Records in set, REMOVE FROM SET, SAVE SET, UNION, USE SET.

Příkazy a odkazy pro Sady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CREATE EMPTY SET (VYTVOŘIT PRÁZDNOU SADU)

Verze 3

CREATE EMPTY SET ({tabulka ;}set)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro kterou vytvořit sadu, nebo Výchozí tabulka pokud vynecháno
set	Řetězec	→	Název prázdné sady

Popis

CREATE EMPTY SET vytvoří novou prázdnou sadu pro tabulku. Do této sady můžete vložit záznamy pomocí ADD TO SET. Pokud sada s tímto názvem již existuje, je přepsána prázdnou.

Poznámka: Před příkazem CREATE SET nepotřebujete provádět CREATE EMPTY SET.

Příklad

Podívejte se na příklady v části Sady.

Dále si přečtěte

CLEAR SET, CREATE SET.

Příkazy a odkazy pro Sady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CREATE SET (VYTVOŘIT SADU)

Příkazy a odkazy pro Sady

Verze 3

CREATE SET ({tabulka ;}set)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka pro kterou vytvořit sadu, nebo Výchozí tabulka pokud vynecháno
set	Řetězec	→	Název nové sady

Popis

CREATE SET vytvoří novou sadu, pro tabulku a umístí do ní platný výběr. Ukazatel na platný záznam tabulky je uložen se sadou. Pokud je sada použita příkazem USE SET, je z ní vytvořen platný výběr a platný záznam. Jako u všech sad, není sada tříděná a je zachováno výchozí pořadí. Pokud již existuje sada s tímto názvem je její obsah přepsán.

Příklad

Následující příklad vytvoří sadu po provedení dotazu a uloží ji na disk:

```
QUERY ([Lidé]) ` Nechat uživatele hledat
CREATE SET ([Lidé]; "SearchSet") ` Vytvořit novou sadu
SAVE SET ("SearchSet"; "MySearch") ` Uložit sadu na disk
```

Dále si přečtěte

CLEAR SET, CREATE EMPTY SET.

Příkazy a odkazy pro Sady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CREATE SET FROM ARRAY (VYTVOŘIT SADU Z ARRAY)

Příkazy a odkazy pro Sady

Verze 6.5

CREATE SET FROM ARRAY ({tabulka;} záznArray{; setNázev})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka sady nebo výchozí tabulka pokud vynecháno
záznArray	LongInt | Logické Array	→	Array čísel záznamů nebo Array logické (True=záznam bude ve výběru, False=záznam nebude ve výběru)
setNázev	Řetězec	→	Název sady k vytvoření nebo použije příkaz k UserSet když vynecháno

Popis

Příkaz CREATE SET FROM ARRAY vytvoří setNázev z:

• LogInt array fyzických čísel záznamů tabulky
• Logického array. V tomto případě pokud je hodnota True bude záznam v sadě a pokud je False, záznam nebude v sadě.

Pokud použijete tento příkaz a předáte LongInt array do záznArray, všechna čísla v array zastupují seznam čísel záznamů které budou v setNázev. Pokud je číslo špatné (například záznam nebyl vytvořen), bude generována chyba -10503.

Pokud použijete tento příkaz a předáte logické array do záznArray, N-tý prvek v array zastupuje N-tý záznam a jestli bude (True) nebo nebude (False) v sadě setNázev. Obvykle má array stejný počet prvků jako tabulka záznamů. Pokud má array méně prvků, budou do sady použity pouze definované záznamy.

Poznámka: S logickým array tento příkaz používá prvky od 0 do N-1.

Pokud nedefinujete paramatr setNázev nebo předáte prázdný řetězec, bude příkaz použit na sadu UserSet.

Dále si přečtěte

BOOLEAN ARRAY FROM SET, CREATE SELECTION FROM ARRAY.

Příkazy a odkazy pro Sady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

USE SET (POUŽÍT SADU)

Verze 3

USE SET (set)

Parametr	Typ		Popis
set	Řetězec	→	Název sady k použití

Popis

USE SET použije záznamy v sadě k vytvoření nového platného výběru tabulky.

Při vytváření sady si sada "zapamatuje" platný záznam. USE SET použije tento označený záznam k vytvoření platného záznamu. Pokud byl tento záznam vymazán, 4th Dimension použije první záznam a udělá z něj platný záznam. Příkazy sady IONTERSECTION, UNION, DIFFERENCE a ADD TO SET zruší nastavení platného záznamu. Také když při vytvoření sady není definovaný platný záznam, použije USE SET první záznam jako platný záznam.

UPOZORNĚNÍ: Nezapomeňte, že sada je zástupce výběru ve chvíli, kde je vytvářena. Pokud je záznam obsažený v sadě upraven, sada nemusí být přesná. Proto sady uložené na disku budou přesné pouze v případě, že se záznamy, které obsahuje, nebudou měnit. Mnoho věcí může ovlivnit platnost sady: upravení záznamu obsaženého v sadě, vymazání záznamu sady nebo změna kritérií které určují sadu.

Příklad

Následující příklad použije LOAD SET k načtění sady "NY Acme". Pak použije USE SET k použití sady jako platného výběru:

```
LOAD SET ([Firmy]; "NY Acme"; "NYAcmeSt") ` Načíst sadu do paměti
USE SET ("NY Acme") ` Změnit platný výběr na sadu
CLEAR SET ("NY Acme") ` Odstranit sadu z paměti
```

Dále si přečtěte

CLEAR SET, LOAD SET.

Příkazy a odkazy pro Sady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ADD TO SET (PŘIDAT DO SADY)

Verze 3

ADD TO SET ({tabulka ;}set)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka platného záznamu, nebo Výchozí tabulka pokud vynecháno
set	Řetězec	→	Název sady do které vložit platný záznam

Popis

ADD TO SET přidá platný záznam tabulky do sady. Sada musí existovat jinak je generována chyba. Pokud pro tabulku není platný záznam, příkaz neprovede nic.

Dále si přečtěte

REMOVE FROM SET.

Příkazy a odkazy pro Sady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

REMOVE FROM SET (ODSTRANIT ZE SADY)

Verze 6.0

REMOVE FROM SET ({tabulka ;}set)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka platného záznamu, nebo Výchozí tabulka pokud vynecháno
set	Řetězec	→	Název sady, ze které vyjmout platný záznam

Popis

REMOVE FROM SETodstraní ze sady platný záznam tabulky. Sada musí existovat jinak je generována chyba. Pokud pro tabulku není platný záznam, příkaz neprovede nic.

Dále si přečtěte

ADD TO SET.

Příkazy a odkazy pro Sady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CLEAR SET (ODSTRANIT SADU)

Verze 3

CLEAR SET (set)

Parametr	Typ		Popis
set	Řetězec	→	Název sady, kterou vymazat z paměti

Popis

CLEAR SET odstraní sadu z paměti a uvolní místo které sada používala. Tento příkaz nemá žádný vliv na výběr záznamů nebo záznamy. K uložení sady před jejím odstraněním použijte příkaz SAVE SET. Protože sady zabírají paměť, je dobré je mazat když je už nepotřebujete.

Příklad

Podívejte se na příkad u příkazu USE SET.

Dále si přečtěte

CREATE EMPTY SET, CREATE SET, LOAD SET.

Příkazy a odkazy pro Sady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Is in set (Je v sadě)

Příkazy a odkazy pro Sady

Verze 3

Is in set (set) → Logické

Parametr	Typ		Popis
set	Řetězec	→	Název sady k testování
Výsledek funkce	Logické	←	Platný záznam tabulky je v sadě (True) nebo není (False)

Popis

Is in set testuje jestli je platný záznam tabulky v sadě. Funkce vrací True, jestli je platný záznam v tabulce a False pokud platný záznam není v tabulce.

Příklad

Následující příklad je metoda objektu která testuje jestli je otevřený záznam v sadě nejlepších zákazníků:

```
If (Is in set ("Best")) ` Testovat jestli je to dobrý zákazník
   ALERT ("Je to jeden z našich nejlepších zákazníků.")
Else
   ALERT ("Nejsou mezi našimi nejlepšími zákazníky.")
End if
```

Dále si přečtěte

ADD TO SET, REMOVE FROM SET.

Příkazy a odkazy pro Sady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Records in set (Záznamů v sadě)

Příkazy a odkazy pro Sady

Verze 3

Records in set (set) → Číslo

Parametr	Typ		Popis
set	Řetězec	→	Název sady k testování
Výsledek funkce	Číslo	←	Počet záznamů v sadě

Popis

Records in set vrací počet záznamů v sadě. Pokud sada neexistuje nebo je prázdná, výsledek bude 0.

Příklad

Následující příklad ukáže kolik procent zákazníků je označeno jako nejlepší:

```
   ` Nejdříve vypočítat procenta
$Percent := (Records in set ("Nejlepsi") / Records in table ([Zákazníci])) * 100
   ` Zobrazit upozornění s procenty
ALERT (String ($Percent; "##0%") + " vašich zákazníků jsou nejlepší.")
```

Dále si přečtěte

Records in selection, Records in table.

Příkazy a odkazy pro Sady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SAVE SET (ULOŽIT SADU)

Verze 3

SAVE SET (set; dokument)

Parametr	Typ		Popis
set	Řetězec	→	Název sady k uložení
dokument	Řetězec	→	Název diskového souboru do kterého sadu uložit

Popis

SAVE SET uloží sadu do dokumentu na disk.

Dokument nepotřebuje mít stejný název jako sada. Pokud do názvu dokumentu předáte prázdný řetězec, otevře se okno Vytvořit soubor, kde můžete dokument vytvořit. Sady můžete načíst příkazem LOAD SET.

Pokud uživatel klepne na tlačítko Zrušit v okně Vytvořit soubor nebo se objeví chyba během ukládání, proměnná OK je nastavena na 0, jinak je nastavena na 1.

SAVE SET se používá k uložení výsledků časově náročného hledání.

UPOZORNĚNÍ: Nezapomeňte že sada je zástupce výběru ve chvíli kde je vytvářena. Pokud je záznam obsažený v sadě upraven, sada nemusí být přesná. Proto sady uložené na disku budou přesné pouze v případě, že se záznamy, které obsahuje, nebudou měnit. Mnoho věcí může ovlivnit platnost sady: upravení záznamu obsaženého v sadě, vymazání záznamu sady nebo změna kritérií které určují sadu. Nezapomeňte, že sada neobsahuje pole.

Příklad

Následující příklad zbrazí okno Uložit soubor, kde může uživatel zadat název dokumentu pro sadu:

```
SAVE SET ("MojeSada"; "")
```

Dále si přečtěte

LOAD SET.

Systémové proměnné a Sady

Pokud uživatel klepne na tlačítko Zrušit v okně Uložit soubor nebo se objeví chyba během ukládání, proměnná OK je nastavena na 0, jinak je nastavena na 1.

Příkazy a odkazy pro Sady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

LOAD SET (NAČÍST SADU)

Příkazy a odkazy pro Sady

Verze 3

LOAD SET ({tabulka;}set; dokument)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ke které sada patří, nebo Výchozí tabulka pokud vynecháno
set	Řetězec	→	Název sady k vytvoření
dokument	Řetězec	→	Název diskového souboru, ze kterého sadu načíst

Popis

LOAD SET načte sadu z dokumentu, který byl uložen pomocí SAVE SET.

Sada která je v dokumnetu musí patřit k tabulce. Pokud již existuje vytvořená sada, je přepsána.

Dokument je název dokumentu na disku obsahující sadu. Dokument nemusí mít stejný název jako sada. Pokud do dokumentu předáte prázdný řetězec, je zobrazeno okno Otevřít soubor kde můžete vybrat dokument sady.

Nezapomeňte, že sada je zástupce výběru ve chvíli, kde je vytvářena. Pokud je záznam obsažený v sadě upraven, sada nemusí být přesná. Proto sady načtené z disku budou přesné pouze v případě, že se záznamy, které obsahuje, nebudou měnit. Mnoho věcí může ovlivnit platnost sady: upravení záznamu obsaženého v sadě, vymazání záznamu sady nebo změna kritérií které určují sadu.

Příklad

Následující příklad použije LOAD SET k načtění sady "NY Acme":

```
LOAD SET ([Firmy]; "NY Acme"; "NYAcmeSt") ` Načíst sadu do paměti
USE SET ("NY Acme") ` Změnit platný výběr na sadu
CLEAR SET ("NY Acme") ` Odstranit sadu z paměti
```

Dále si přečtěte

SAVE SET.

Systémové proměnné a Sady

Pokud uživatel klepne na tlačítko Zrušit v okně Otevřít soubor nebo se objeví chyba během načítání, proměnná OK je nastavena na 0, jinak je nastavena na 1.

Příkazy a odkazy pro Sady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DIFFERENCE (ROZDÍL)

Příkazy a odkazy pro Sady

Verze 3

DIFFERENCE (set; odečítSet; výslSet)

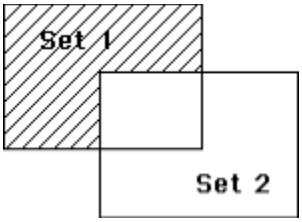
Parametr	Typ		Popis
set	Řetězec	→	Sada
odečístSet	Řetězec	→	Sada k odečtení
výsleSet	Řetězec	→	Výsledná sada z set po odečtení záznamů sady odečístSet

Popis

DIFFERENCE porovná sadu1 se sadou2 a odstraní všechny záznamy, které jsou v sadě2 z výsleSady. Jinými slovy záznamy budou ve výsledné sadě pouze když jsou v sadě1 a nejsou v sadě2. Následující tabulka ukazuje výsledek operace rozdíl:

Set1	***Set2***	***Výsledna sada (Set)***
Ano	*Ne*	*Ano*
Ano	*Ano*	*Ne*
Ne	*Ano*	*Ne*
Ne	*Ne*	*Ne*

Výsledek můžeme znázornit graficky. Zvýrazněná oblast je výsledná sada:



VýsleSet je vytvořena příkazem DIFFERENCE. Pokud již existuje sada s takovým názvem, je přepsána. Jak sada1 tak sada2 musí patřit k jedné tabulce a i výsledná sada bude patřit ke stejné tabulce.
4D Server: V klint/server jsou meziprocesní a procesové sady ovládány na serveru a místní sady jsou ovládány na klientovi. DIFFERENCE vyžaduje aby všechny sady byly na jednom počítači. Jinými slovy místní musí být všechny nebo žádná. Podívejte se na popis 4D Server a Sady (4D Server and Sets) v *příručce ke 4D Server*.

Příklad

Tento příklad vyloučí záznamy, které uživatel označí ve výběru. Záznamy jsou zobrazeny následujícím řádkem:

```
DISPLAY SELECTION ([Zákazníci])  ` Zobrazit seznam zákazníků
```

Na spodu seznamu záznamů je tlačítko s metodou objektu. Tato metoda vyloučí označneé záznamy z výběru a obrazí nový výběr:

```
CREATE SET ([Zákazníci]; "$Current") ` Vytvořit sadu z platného výběru
DIFFERENCE ("$Current";"UserSet";"$Current") ` Vynechat označené záznamy
USE SET ("$Current") ` Použít novou sadu
CLEAR SET ("$Current") ` Odstranit sadu
```

Dále si přečtěte

INTERSECTION, UNION.

Příkazy a odkazy pro Sady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

INTERSECTION (PRŮNIK)

Příkazy a odkazy pro Sady

Verze 3

INTERSECTION (set1; set2; VýsleSet)

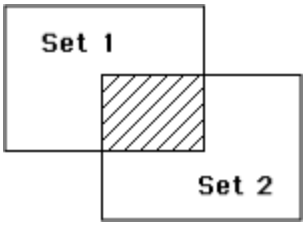
Parametr	Typ		Popis
set1	Řetězec	→	První sada
set2	Řetězec	→	Druhá sada
VýsleSet	Řetězec	→	Výsledná sada

Popis

INTERSECTION porovná set1 a set2 a vybere pouze záznamy které jsou v obou. následující tabulka ukazuje seznam možných výsledků průniku.

Set1	***Set2***	***Výsledná Sada***
Ano	*Ne*	*Ne*
Ano	*Ano*	*Ano*
Ne	*Ano*	*Ne*
Ne	*Ne*	*Ne*

Grafický výsledek průniku je zde. Zvýrazněná oblast je výsledná sada:



Výsledná sada je vytvořena příkazem INTERSECTION. Pokud již existuje sada se stejným názvem jako VýsleSet, je tato sada přepsána. Jak sada1 tak sada2 musí patřit k jedné tabulce a i výsledná sada bude patřit ke stejné tabulce.

4D Server: V klint/server jsou meziprocesní a procesové sady ovládány na serveru a místní sady jsou ovládány na klientovi. INTERSECTION vyžaduje aby všechny sady byly na jednom počítači. Jinými slovy místní musí být všechny nebo žádná. Podívejte se na popis 4D Server a Sady (4D Server and Sets) v *příručce ke 4D Server*.

Příklad

Následující příklad nalezne zákazníky, které mají na starosti Joe a Abby. Každý z obchodních zástupců má sadu která ukazuje jeho nebo její zákazníky. Zákazníky kteří jsou v obou sadách mají na starosti oba prodejci.

```
INTERSECTION ("Joe"; "Abby"; "Oba") ` Vložit zákazníky z obou sad
USE SET ("Oba") ` Použít sadu
CLEAR SET ("Oba") ` Odstranit tuto sadu
DISPLAY SELECTION ([Zákazníci]) ` Zobrazit zákazníky
```

Dále si přečtěte

DIFFERENCE, UNION.

Příkazy a odkazy pro Sady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

UNION (SJEDNOCENÍ)

Příkazy a odkazy pro Sady

Verze 3

UNION (set1; set2; výsleSet)

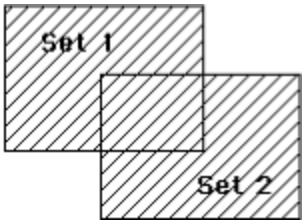
Parametr	Typ		Popis
set1	Řetězec	→	První sada
set2	Řetězec	→	Druhá sada
VýsleSet	Řetězec	→	Výsledná sada

Popis

UNION vytvoří sadu která bude obsahovat záznamy ze set1 a set2. Následující tabulka ukazuje možné výsledky spojení:

Set1	***Set2***	***Výsledná sada***
Ano	*Ne*	*Ano*
Ano	*Ano*	*Ano*
Ne	*Ano*	*Ano*
Ne	*Ne*	*Ne*

Grafické znázornění výsledku je na následujícím obrázku.



Výsledná sada je vytvořena příkazem UNION. Pokud již existuje sada se stejným názvem jako VýsleSet, je tato sada přepsána. Jak sada1 tak sada2 musí patřit k jedné tabulce a i výsledná sada bude patřit ke stejné tabulce. Platný záznam ve výsleSet je záznam ze set1.

4D Server: V klint/server jsou meziprocesní a procesové sady ovládány na serveru a místní sady jsou ovládány na klientovi. UNION vyžaduje aby všechny sady byly na jednom počítači. Jinými slovy místní musí být všechny nebo žádná. Podívejte se na popis 4D Server a Sady (4D Server and Sets) v *příručce ke 4D Server*.

Příklad

Tento příklad přidá záznamy do sady nejlepších zákazníků. Záznamy jsou zobrazeny na obrazovce a uživatel může označit ty, které chce přidat do sady. Po zobrazení zákazníků je načtena sada nejlepších a záznamy které jsou označneé jsou do ní přidány. Nakonec je sada opět uložena na disk:

```
ALL RECORDS ([Zákazníci]) ` Vybrat všechny záznamy
DISPLAY SELECTION ([Zákazníci]) ` Zobrazit zákazníky v seznamu
LOAD SET ("$Best"; "$SaveBest") ` Načíst sadu nejlepších zákazníků
UNION ("$Best"; "UserSet"; "$Best") ` Vložit označené do sady
SAVE SET ("$Best"; "$SaveBest") ` Uložit sadu na disk
```

Dále si přečtěte

DIFFERENCE, INTERSECTION.

Příkazy a odkazy pro Sady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

COPY SET (KOPÍROVAT SADU)

Příkazy a odkazy pro Sady

Verze 3

COPY SET (zdrSet; cílSet)

Parametr	Typ		Popis
zdrSet	Řetězec	→	Zdrojová sada
cílSet	Řetězec	→	Cílová sada

Popis

Příkaz COPY SET zkopíruje obsah zsrSet do sílSet.

Obě sady mohou být procesové, meziprocesní nebo místní.

4D Server: V klint/server jsou meziprocesní a procesové sady ovládány na serveru a místní sady jsou ovládány na klientovi. COPY SET vám umožní kopírovat sady mezi různými počítači. Podívejte se na popis 4D Server a Sady (4D Server and Sets) v *příručce ke 4D Server*.

Příklady

1. Následující příklad, v klient/server, kopíruje místní sadu ovládanou na počítači klienta do sady procesu ovládané:

```
COPY SET("$SetA";"SetB")
```

2. Následující příklad, v klient/server, kopíruje procesovou sadu ovládanou na serveru do místní sady ovládané na klientovi:

```
COPY SET("SetA";"$SetB")
```

Dále si přečtěte

Sady.

Příkazy a odkazy pro Sady

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

String (Řetězec)

Příkazy a odkazy pro Text řetězce

Verze 3

String (výraz{; formát}) → Řetězec

Parametr	Typ		Popis
výraz		→	Výraz jehož hodnotu chceme jako řetězec ve formátu (může být Real, Integer, Long Integer, Datum nebo Čas)
formát	Řetězec | Číslo	→	Formát zobrazení
Výsledek funkce	Řetězec	←	Řetězcový formát výrazu

Popis

Příkaz String vrací řetězcovou formu číselných, datumových nebo časových výrazů předáných do výraz.

Pokud nepředáte volitelný parametr formát, řetězec je vrácen v příslušném výchozím formátu. Pokud tento parametr předáte, vnutíte výslednému řetězci specifický formát.

Číslené výrazy

Pokud je výraz číslený (real, Ineteger, Long Integer), můžete vložit volitelný formát řetězce. Následující příklady:

Příklad	***Výsledek***
String(2^15) ` Použít výchozí formát	*32768 (Výchozí formát použitý zde)*
String(2^15;"###,##0 Obyvatel")	*32,768 Obyvatel*
String(1/3;"##0.00000")	*0.33333*
String(1/3) ` Použít výchozí formát	*0.3333333333333333 (Výchozí formát použit)*
*String(Arctan(1)*4)*	*3.1415926535897931 (Výchozí formát použit)*
*String(Arctan(1)*4;"##0.00")*	*3.14*
String(-1;"&x")	*0xFFFFFFFF*
String(-1;"&$")	*$FFFFFFFF*
String(0 ?+ 7;"&x")	*0x80*
String(0 ?+ 7;"&$")	*$80*
String(0 ?+ 14;"&x")	*0x4000*
String(0 ?+ 14;"&$")	*$4000*
String(Num(1=1);"True;;False")	*True*
String(Num(1=2);"True;;False")	*False*

Formát je definován stejným způsobem jako pro číslené pole ve formuláři. V *Příručce návrháře 4th Dimension* naleznete více informací o formátech zobrazení. Můžete také vložit název vlastního formátu. Název tohoto formátu musí začínat znakem "|".

Datumové výrazy

Pokud je výraz časový, řetězec je vrácen s použitím výchozího zobrazení státu (např. MM/DD/RR na US verzi). Můžete vložit formáty z následující tabuky:

Formát	***Název***	***Příklad***
1	*Číslice*	*29.12.1999*
2	*Zkratky*	*st 29. 12. 1999*
3	*Plné*	*středa 29. prosince 1999*
4	*dd.mm.rrrr*	*29.12.99 nebo 29.12.1899 nebo 29.12.2099*
5	*Den měsíc rok*	*středa 29. prosince 1999*

6	*Zkr: den měsíc rok*	*st 29. 12. 1999*
7	*MM/DD/YYYY Forced*	*29.12.1999*

4D obsahuje následující předdefinované konstanty:

Konstanta	***Typ***	***Hodnota***
Short	*Long Integer*	*1*
Abbreviated	*Long Integer*	*2*
Long	*Long Integer*	*3*
MM DD YYYY	*Long Integer*	*4*
Month Date Year	*Long Integer*	*5*
Abbr Month Date	*Long Integer*	*6*
MM DD YYYY Forced	*Long Integer*	*7*

Tento příklad přijme že platné datum je 29.12.1999:

```
   ` $vsRes přiřadit "31.12.1999"
$vsRes:=String (Current date)
   ` $vsRes přiřadit "pátek 31. prosince 1999"
$vsRes := String (Current date;Month Date Year)
```

Časový výraz

Pokud je výraz časový, je řetězec vrácen s použitím výchozího formátu HH:MN:SS. Do volitelného parametru formát můžete vložit následující hodnoty:

Formát	***Název***	***Příklad***
1	*HH:MN:SS*	*01:02:03*
2	*HH:MN*	*01:02*
3	*hod min sec*	*1 hodin 2 minut 3 sekund*
4	*hod min*	*1 hodin 2 minut*
5	*H:MN Dop/Odp*	*1:02 Dop*

4D obsahuje následující předdefinované konstanty:

Konstanta	***Typ***	***Hodnota***
HH MM SS	*Long Integer*	*1*
HH MM Long	*Integer*	*2*
Hour Min Sec	*Long Integer*	*3*
Hour Min	*Long Integer*	*4*
HH MM AM PM	*Long Integer*	*5*

Tento příklad příjme že platný čas je 5:30 Odp a 45 sekund:

```
$vsResult:=String(Current time) ` $vsResult bdue "17:30:45"
$vsResult:=String(Current time;Hour Min Sec) ` $vsResult bude "17 hodin 30 minut 45 sekund"
```

Dále si přečtěte

Date, Num, Time string.

Příkazy a odkazy pro Text řetězce

4th Dimension 6.5, Seznam témat příkazů

Num (Číslo)

Příkazy a odkazy pro Text řetězce

Verze 3

Num (výraz) → Číslo

Parametr	Typ		Popis
výraz	Řetězec | Logické	→	Řetězec pro který vrátit číselnu hodnotu nebo Logickou vrací 1 nebo 0
Výsledek funkce	Číslo	←	Číslená hodnota řetězce nebo logické hodnoty

Popis

Příkaz Num vrací číselnou hodnotu řetězce nebo logické hodnoty kterou předáte do výrazu.

Výraz řetězec

Pokud řetězec obsahuje pouze jedno nebo více písmen, vrátí Nu 0. Pokud řetězec obsahuje písmena a čísla, bude Num ignorovat písmena a zobrazí pouze čísla. Pokud bude řetězec "a1b2c3" bude výsledek 123.

Poznámka: Pouze prvních 32 znaků řetězce bude převedeno.

Jsou pouze tři znaky se kterými Num zachází jinak. Je to desetiná čárka (","), pomlčka ("-") a "e" nebo "E". Tyto znaky jsou převedeny jako číselné výrazy.

• Čárka je pochopena jako desetinná čárka a musí být předána do číselného řetězce.
• Pomlčka udělá z čísla záporné číslo. Musí být před záporným číslem nebo za "e" jako exponent. Pokud je pomlčka předána do čísleného řetězce, funkce Num vrátí (0). Například Num(123-456) vrátí 0, ale Num(-9) vrátí -9.
• Pokud v řetězci bude e nebo E na pravo od čísla, bude vzat jako esponent. "e" musí být předáno do číselného řetězce. Například Num(123e-2) vrátí 1,23.

Logický výraz

Pokud předáte logický výraz, Num vrátí 1 pokud je výraz True nebo 0 pokud je výraz False.

Příklady

1. Následujíc příklad ukazuje jak pracuje Num při předání různých řetězců. Každý řádek přiřadí hodnotu do proměnné vResult:

```
vResult := Num ("ABCD") ` vResult bude 0
vResult := Num ("A1B2C3") ` vResult bude 123
vResult := Num ("123") ` vResult bude 123
vResult := Num ("123,4") ` vResult bude 123,4
vResult := Num ("_123") ` vResult bude _123
vResult := Num ("_123e2") ` vResult bude _12300
```

2. Zde je [Klienti]Dluh porovnáváno s 1000. Pak Num použitý na tato porovnání vrací 1 nebo 0. Násobení 1 nebo 0 řetězcem opakuje řetězec jednou nebo vrací prázdný řetězec. Jako výsledek do [Klient]Risk se přiřadí buď "Dobré" nebo "Špatné".

```
    ` Pokud klient dluží méně než 1000, risk je dobrý
    ` Pokud klient dluží více než 100, je špatné riskovat
[Klient]Risk := ("Dobré"*Num ([Klienti]Dluh<1000))+ ("Špatné"*Num([Klienti]Dluh>1000))
```

Dále si přečtěte

Logické operátory, String, Operátory řetězce.

Příkazy a odkazy pro Text řetězce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Position (Pozice)

Příkazy a odkazy pro Text řetězce

Verze 3

Position (nalézt; řetězec) → Číslo

Parametr	Typ		Popis
nalézt	Řetězec	→	Řetězec k nalezení
řetězec	Řetězec	→	Řetězec ve kterém hledat
Výsledek funkce	Číslo	←	Pozice prvního výskytu

Popis

Position vrací pozici prvního výskytu hledaného v řetězci.

Pokud řetězec neobsahuje nalézt, bude vrácena 0.

Pokud Position nalezne nalézt v řetězci, bude vrácena pozice prvního znaku.

Pokud budete hledat prázdný řetězec v prázdném řetězci, Position vrátí True.

Upozornění: Nemůžete použít @ v hledaném řetězci. Pokud například předáte hledat "abc@", příkaz bude hledat "abc@" a ne "abc" plus nějaké znaky.

Příklad

1. Tento příklad ukazuje použití Position. Výsledky popsané v komentářích jsou dosazeny do vlVysledek.

```
vlVysledek := Position ("ll"; "Willow") ` vlVysledek bude 3
vlVysledek := Position (vtText1; vtText2) ` Vrátí první výskyt vtText1 v vtText2
```

2. Podívejte se na příklad u příkazu Substring.

Dále si přečtěte

Operátory porovnání, Substring.

Příkazy a odkazy pro Text řetězce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Substring (Podřetězec)

Příkazy a odkazy pro Text řetězce

Verze 3

Substring (zdroj; prvnízn{; početzn}) → Řetězec

Parametr	Typ		Popis
zdroj	Řetězec	→	Řetězec ze kterého chceme podřetězec
prvnízn	Číslo	→	Pozice prvního znaku
početzn	Číslo	→	vPočet znaků které chceme
Výsledek funkce	Řetězec	←	Podřetězec ze zdroje

Popis

Příkaz Substring vrací podřetězec ze zdroje počínaje prvnízn a s délkou *početzn*.

Parametr *prvnízn* ukazuje na první znak v řetězci a *početzn* definuje kolik znaků vzít.

Jestliže je prvnízn plus početzn větší než počet znaků ve zdrojovém řetězci, nebo když početzn není definován, Substring vrátí konečné znaky řetězce počínaje na prvnízn. Pokud je prvnízn větší než počet znaků v řetězci, vrátí funkce prázdný řetězec ("").

Příklady

1. Tento příklad ukazuje použití Substring. Výsledky popsané v komentářích jsou přiřazeny do proměnné vsResult.

```
vsResult := Substring ("08/04/62"; 4; 2) ` vsResult bude "04"
vsResult := Substring ("Emergency"; 1; 6) ` vsResult bude "Emerge"
vsResult := Substring (var; 2) ` vsResult bude obsahovat všechny znaky mimo prvního
```

2. Následující metoda projektu připojí odstavce nalezené v textu (předáný jako první parametr) do řetězce, nebo text array (na který je ukazatel jako druhý parametr):

```
   ` ZÍSKAT ODSTAVCE
   ` ZÍSKAT ODSTAVCE ( text ; Ukazatel )
   ` ZÍSKAT ODSTAVCE ( Text k rozdělení ; → Array odstavců )
C_TEXT ($1)
C_POINTER ($2)
$vlElem:=Size of array($2→)
Repeat
   $vlElem:=$vlElem+1
   INSERT ELEMENT($2→;$vlElem)
   $vlPos:=Position(Char(Carriage return);$1)
   If ($vlPos>0)
      $2→{$vlElem}:=Substring($1;1;$vlPos-1)
      $1:=Substring($1;$vlPos+1)
   Else
      $2→{$vlElem}:=$1
   End if
Until ($1="")
```

Dále si přečtěte

Position.

Příkazy a odkazy pro Text řetězce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Length (Délka)

Příkazy a odkazy pro Text řetězce

Verze 3

Length (řetězec) → Číslo

Parametr	Typ		Popis
řetězec	Řetězec	→	Řetězec pro který získat délku
Výsledek funkce	Číslo	←	Délka řetězce

Popis

Length se používá k získání délky řetězce. Length vrací počet znaků v řetězci.

Poznámka: Test If (vtTest="") je shodný jako If(Length(vtText)=0).

Příklady

Tento příklad ukazuje použití Length. Výsledek popsaný v komentářích je dosazen do proměnné vlVysledek.

```
vlVysledek := Length ("Topaz") ` vlVysledek bude 5
vlVysledek := Length ("Citizen") ` vlVysledek bude 7
```

Příkazy a odkazy pro Text řetězce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Ascii (Ascii kód)

Příkazy a odkazy pro Text řetězce

Verze 3

Ascii (znak) → Číslo

Parametr	Typ		Popis
znak	Řetězec	→	Znak pro který vrátit ASCII kód
Výsledek funkce	Číslo	←	ASCII kód znaku

Popis

Příkaz Ascii vrací ASCII kód znaku.

Pokud je v řetězci více než jeden znak, Ascii vrátí kód prvního znaku.

Funkce Char je opak funkce Ascii. Vrací z kódu ASCII znak, který zastupuje.

Důležité: uvnitř 4D jsou všechny textové hodnoty, pole nebo proměnné které jste nepřevedli na jinou ASCII mapu jsou kódovány na Macintosh jak na Windows tak na Macintoshi. Jestli chcete vědět více informací, přečtěte si část ASCII kódy.

Příklady

1. Malá a velká písmena jsou shodná při porovnávání. Můžete použít Ascii k odlišení jejich hodnot. Tento řádek vrací True:

```
("A"="a")
```

Ale tento řádek vrací False:

```
(Ascii("A")=Ascii("a"))
```

2. Tento příklad vrací ASCII kód prvního znaku v řetězci "ABC":

```
vlAscii := Ascii ("ABC") ` vlAscii bude 65, ASCII kód znaku A.
```

3. Následující příklad testuje entry a tabelátory:

```
For($vlChar;1;Length(vtText))
   Case of
      ÷ (vtText[[$vlChar]]=Char(Carriage return))
         ` Udělat něco
      ÷ (vtText[[$vlChar]]=Char(Tab))
         ` Udělat něco jiného
      ÷ (...)
         ` ...
   End case
End for
```

Při provedení na velké textové bloky, bude tento kód fungovat rychleji zkompilovaný, pokud bude napsán takto:

```
For($vlChar;1;Length(vtText))
   $vlAscii:=Ascii(vtText[[$vlChar]])
   Case of
      ÷ ($vlAscii=Carriage return)
```

```
            ` Udělat něco
        ÷ ($vlAscii=Tab)
            ` Udělat něco jiného
        ÷ (...)
            ` ...
    End case
End for
```

Druhý kód funguje rychleji ze dvou důvodů: používá odkaz pouze na jeden znak během opakování a používá LongInt porovnání místo porovnání řetězce pro testování entru nebo tabelátoru. Používejte tuto techniku při práci s ASCII kódy jako jsou Enter nebo Tab.

Dále si přečtěte

ASCII kódy, Char, Symboly odkazů na znaky.

Příkazy a odkazy pro Text řetězce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Char (Znak)

Příkazy a odkazy pro Text řetězce

Verze 3

Char (asciikód) → Řetězec

Parametr	Typ		Popis
asciikód	Číslo	→	ASCII kód od 0 do 255
Výsledek funkce	Řetězec	←	Znak zastupovaný ASCII kódem

Popis

Příkaz Char vrací znak jehož ASCII kód zadáte do asciikód.

Tip: Při upravování metody je příkaz Char použit hlavně pro znaky které nemohou být napsány z klávesnice, nebo které by mohli být pochopeny jako upravující příkazy v Editoru metod.

Důležité: uvnitř 4D jsou všechny textové hodnoty, pole nebo proměnné které jste nepřevedli na jinou ASCII mapu jsou kódovány na Macintosh jak na Windows tak na Macintoshi. Jestli chcete vědět více informací, přečtěte si část ASCII kódy.

Příklady

Následující příklad používá Char k předání znaku Nový řádek do textu upozornění:

```
ALERT("Zaměstnanci: "+String(Records in table([Zaměstnanci]))+ Char(13)+
    "Klepněte na OK k pokračování.")
```

Dále si přečtěte

Ascii, ASCII kódy, Symboly odkazů na znaky.

Příkazy a odkazy pro Text řetězce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Symboly odkazů na znaky

Příkazy a odkazy pro Text řetězce

Verze 3

Úvod

Symboly odkazů na znaky:

[[...]] Na Windows

větší, menší rovno nebo [[...]] na Macintoshi

jsou použity k odkazu na jeden znak v řetězci. Tento zápis vám umožní odkázat se na jeden specifický znak v textové proměnné, řetězcové proměnné nebo poli.

Poznámka: Na Macintoshi napíšete první dva znaky Option+Shift+"<" a Option+Shift+">".

Pokud je symbol odkazu na znak nalevo od operátoru přiřazení (:=) je hodnota přiřazená na zadanou pozici v řetězci. Například pokud vsName není prázdný řetězec, následující řádek vymění první znak za velké písmeno:

```
If (vsName#"")
   vsName[[1]]:=Uppercase(vsName[[1]])
End if
```

Jinak pokud bude symbol odkazu na znak ve výrazu, bude vrácen znak (na který je odkaz) jako 1 znakový řetězec. Například:

```
   ` Následující příklad testuje jestli poslední znak z vtTest je znak @
If (vtText # "")
   If (Ascii(Substring(vtText;Length(vtText);1))=At Sign)
      ` ...
   End if
End if

   ` S použitím symbolů odkazů na znaky můžete napsat jednodušší verzi:
If (vtText # "")
   If (Ascii(vtText[[Length(vtText)]])=At Sign)
      ` ...
   End if
End if
```

Rozšířená poznámka o nesprávných odkazech na znaky

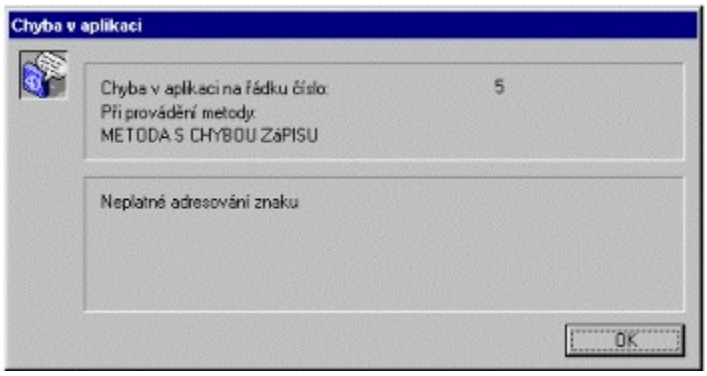
Pokud používáte odkazy na znaky, musíte adresovat exitující znak v řetězci stejně jako musíte adresovat existující prvky v array. Pokud se například odkazujete na 20-tý znak, MUSÍ řetězec obsahovat alespoň 20 znaků.

• Pokud bude odkaz mimo řetězec v nezkompilované verzi, nevznikne chyba syntaxe.
• Pokud bude odkaz mimo řetězec ve zkompilované databázi (bez nastavení), můžete způsobit chybu paměti, pokud například budete hledat znak za koncem řetězce nebo textu.
• Pokud bude odkaz mimo řetězec ve zkompilované databázi, s nastaveným Range Checking On. Například provedete následující kód:

```
vsText:=""
```

```
vsText[[1]]:="A" ` Velmi špatně, fuj!
```

generuje následující chybu v kompilované verzi:



Příklad

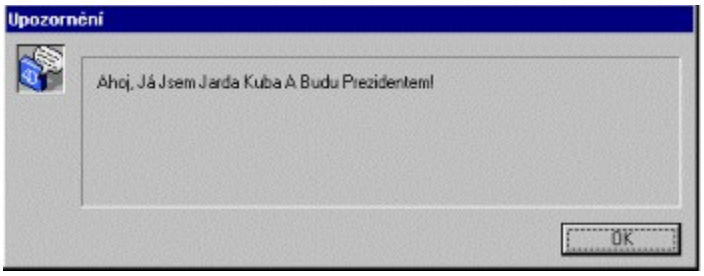
Následující metoda projektu zvětší první znak každého slova textu, který byl předán jako parametr, a vrátí upravený text:

```
    ` Metoda projektu Zvětšit text
    ` Zvětšit text ( Text ) → Text
    ` Zvětšit text ( Zdrojový text ) → Zvětšený text
$0:=$1
$vlLen:=Length($0)
If ($vlLen>0)
   $0[[1]]:=Uppercase($0[[1]])
   For ($vlChar;1;$vlLen-1)
      If (Position($0[[$vlChar]];" !&()-{}:;<>?/,.=+*")>0)
         $0[[$vlChar+1]]:=Uppercase($0[[$vlChar+1]])
      End if
   End for
End if
```

Například následující řádek:

```
ALERT (Zvětšit text ("ahoj, já jsem jarda kuba a budu prezidentem!"))
```

zobrazí následující upozornění:



Dále si přečtěte

Ascii, ASCII kódy, Char.

Příkazy a odkazy pro Text řetězce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Uppercase (Velká písmena)

Příkazy a odkazy pro Text řetězce

Verze 3

Uppercase (řetězec) → Řetězec

Parametr	Typ		Popis
řetězec	Řetězec	→	Řetězec k převodu do velkých písmen
Výsledek funkce	Řetězec	←	Řetězec velkých písmen

Popis

Uppercase vezme řetězec a všechna písmena převede na velká písmena

Příklad

Podívejte se na příklad u příkazu Lowecase.

Dále si přečtěte

Lowercase.

Příkazy a odkazy pro Text řetězce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Lowercase (Malá písmena)

Příkazy a odkazy pro Text řetězce

Verze 3

Lowercase (Řetězec) → Řetězec

Parametr	Typ		Popis
řetězec	Řetězec	→	Řetězec k převodu do malých písmen
Výsledek funkce	Řetězec	←	Řetězec malých písmen

Popis

Lowercase vezme řetězec a převede všechna jeho písmena na malá písmena.

Příklad

Následující metoda projektu zvětší první písmeno předáného řetězce a ostatní zmenší. Například "JARDA" bude "Jarda" a "martin" bude "Martin".

```
   ` Metoda projektu Zvětšit
   ` Zvětšit (Řetězec) → Řetězec
   ` Zvětšit (Text nebo Řetězec) → Zvětšený text
$0 := Lowercase ($1)
If (Length($0)>0)
   $0[[0]]:=Uppercase($0[[1]])
End if
```

Dále si přečtěte

Uppercase.

Příkazy a odkazy pro Text řetězce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Change string (Změnit řetězec)

Příkazy a odkazy pro Text řetězce

Verze 3

Change string (zdroj; novéznaky; kde) → Řetězec

Parametr	Typ		Popis
zdroj	Řetězec	→	Originální řetězec
novéZnaky	Řetězec	→	Nové znaky
kde	Číslo	→	Kde začít změny
Výsledek funkce	Řetězec	←	Výsledný řetězec

Popis

Change string změní skupinu znaků ve zdrojovém řetězci a vrátí výsledný řetězec. Nahradí znaky za novéZnaky od znaku definovaného parametrem kde.

Pokud je novéZnaky prázdný řetězec (""), Change string vrátí zdrojový řetězec nezměněný. Change string vždy vrací řetězec stejné délky jako je zdrojový. Pokud je kde menší než 1 a nebo pokud je větší než počet znaků ve zdroji, vrátí příkaz nezměněný řetězec.

Change string je odlišný od Insert string, protože znaky nahrazuje místo aby je vložil.

Příklad

Následující příklad ukazuje použití příkazu Change string. Výsledky popsané v komentářích jsou přiřazeny do proměnné vsResult.

```
vtResult := Change string ("Acme"; "CME"; 2) ` vtResult bude "ACME"
vtResult := Change string ("November";"Dec"; 1) ` vtResult bude "December"
```

See Also

Delete string, Insert string, Replace string.

Příkazy a odkazy pro Text řetězce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Insert string (Vložit řetězec)

Příkazy a odkazy pro Text řetězce

Verze 3

Insert string (zdroj; co; kam) → Řetězec

Parametr	Typ		Popis
zdroj	Řetězec	→	Řetězec do kterého vložit
co	Řetězec	→	Řetězec který vložit
kam	Číslo	→	Kam vložit
Výsledek funkce	Řetězec	←	Výsledný řetězec

Popis

Insert string vloží řetězec do zdrojového řetězce. Insert string vloží řetězec co před pozici definovou kam.

Pokud je co prázdný řetězec (""), Insert string vrátí zdroj nezměněný.

Pokud je kam větší než počet znaků v řetězci, bude co připojeno za řetězec. Pokud je kam menší než 1, je co předáno před zdrojový řetězec.

Insert string je odlišný od Change string, protože vloží znaky a nepřepíše je.

Příklad

Následující příklad ukazuje použití příkazu Insert string. Výsledky popsané v komentářích jsou přiřazeny do proměnné vsResult.

```
vtResult := Insert string ("The tree"; " green"; 4) ` vtResult bude "The green tree"
vtResult := Insert string ("Shut"; "o"; 3) ` vtResult bude "Shout"
vtResult := Insert string ("Indention"; "ta"; 6) ` vtResult bude "Indentation"
```

Dále si přečtěte

Change string, Delete string, Replace string.

Příkazy a odkazy pro Text řetězce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Delete string (Vymazat řetězec)

Příkazy a odkazy pro Text řetězce

Verze 3

Delete string (zdroj; kde; početZnaků) → Řetězec

Parametr	Typ		Popis
zdroj	Řetězec	→	Řetězec ze kterého mazat
kde	Číslo	→	První znak, který mazat
početZnaků	Číslo	→	Počet znaků k vymazání
Výsledek funkce	Řetězec	←	Výsledný řetězec

Popis

Celete string vymaže početZnaků ze zdroje, začne na pozici kde a vrátí výsledný řetězec.

Delete string vrátí stejný řetězec jako zdroj pokud:

- zdroj je prázdný řetězec
- kde je větší než počet znaků ve zdroji
- početZnaků je nula (0)

Pokud je kde menší než 1, znaky jsou vymazány ze začátku řetězce.

Pokud je početZnaků stejně velké nebo větší než počet znaků, budou vymazány všechny znaky od kde do konce řetězce.

Příklad

Následující příklad ukazuje použití příkazu Delete string. Výsledky popsané v komentářích jsou přiřazeny do proměnné vsResult.

```
vtResult:=Delete string("Lamborghini"; 6; 6) ` vtResult bude "Lambo"
vtResult:=Delete string("Indentation"; 6; 2) ` vtResult bude "Indention"
vtResult:=Delete string(vtVar;3;32000) ` vtResult bude první dva znaky vtVar
```

Dále si přečtěte

Change string, Insert string, Replace string.

Příkazy a odkazy pro Text řetězce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Replace string (Nahradit řetězec)

Příkazy a odkazy pro Text řetězce

Verze 3

Replace string (zdroj; původřet; novýřet{; kolik}) → Řetězec

Parametr	Typ		Popis
zdroj	Řetězec	→	Řetězec ze kterého nahrazovat
původřet	Řetězec	→	Skupina znaků které nahradit
novýřet	Řetězec	→	Skupiny znaků kterými nahrazovat je-li prázdný "" původní řetězec bude vymazán
kolik	Číslo	→	Kolikrát nahradit, je-li vynecháno, jsou nahrazeny všechny výskyty
Výsledek funkce	Řetězec	←	Výsledný řetězec

Popis

Replace string nahradí kolik výskytů původřet ve zdroji za novýřet.

Pokud je novýřet prázdný (""), Replace string vymaže všechny výskyty starýřet ve zdroji.

Pokud je definováno kolik, Replace string nahradí pouze zadaný počet výskytů starýřet a začne od prvního výskytu. Pokud kolik není definováno jsou nahrazeny všechny výskyty.

Pokud je starýřet prázdný, příkaz vrátí nezměněný řetězec.

Příklady

1. Následující příklad ukazuje použití příkazu Replace string. Výsledky popsané v komentářích jsou přiřazeny do proměnné vsResult.

```
vtResult:=Replace string("Willow";" ll";"d") ` Result bude "Widow"
vtResult:=Replace string("Shout"; "o";"") ` Result bude "Shut"
vtResult:=Replace string(vtVar;Char(9);",") ` Nahradí všechny Tabelátory v vtVar za čárky
```

2. Následující příklad odstraní všechny Entry a Tabelátory z textu

```
vtResult:=Replace string(Replace string(vtResult;Char(13);"");Char(9);"")
```

Dále si přečtěte

Change string, Delete string, Insert string.

Příkazy a odkazy pro Text řetězce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Mac to Win (Mac do Win)

Příkazy a odkazy pro Text řetězce

Verze 6.0

Mac to Win (text) → Řetězec

Parametr	Typ		Popis
text	Řetězec	→	Text vyjádřený s použitím MacOS ASCII mapy
Výsledek funkce	Řetězec	←	Text vyjádřený s použitím Windows ANSI mapy

Popis

Příkaz Mac to Win vrací text vyjádřený s použitím Windows ANSI mapy shodný s textem který jste vložili do Text vyjádřený s použitím MacOS ASCII mapy.

Tento příkaz očekává text vyjádřený s použitím ASCII mapy MacOS.

Pokud pracujete pouze na Windows, nebudete potřebovat tento příkaz pro převod ASCII kódů. Pokud kopírujete text mezi 4D a Windows nebo když importujete nebo exportujete data, 4th Dimension automaticky provede tento převod. Nicméně, když používáte diskové číst/psát příkazy jako SEND PACKET nebo RECEIVE PACKET, budete potřebovat ASCII převody. Toto je hlavní účel příkazu Mac to Win.

Uvnitř 4D jsou všechny textové hodnoty, pole nebo proměnné které jste nepřevedli na jinou ASCII mapu jsou kódovány na Macintosh jak na Windows tak na Macintoshi. Jestli chcete vědět více informací, přečtěte si část ASCII kódy.

Příklad

Pokud na Windows zapíšete znaky do dokumentu s použitím SEND PACKET a pokud nepoužijete výstupní ASCII mapu pro filtrování znaků z MacOS na Windows (podívejte se na příklad USE ASCII MAP), budete potřebovat převést text z MacOS na Windows osobně. Můžete to udělat takto:

```
` ...
SEND PACKET ($vhDokumRef;Mac to Win(vtNejakyText))
` ...
```

Dále si přečtěte

ASCII kódy, SEND PACKET, USE ASCII MAP, Win to Mac.

Příkazy a odkazy pro Text řetězce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Win to Mac (Win do Mac)

Příkazy a odkazy pro Text řetězce

Verze 6.0

Win to Mac (text) → Řetězec

Parametr	Typ		Popis
text	Řetězec	→	Text vyjádřený s použitím Windows ANSI
Výsledek funkce	Řetězec	←	mapy Text vyjádřený s použitím MacOS ASCII mapy

Popis

Příkaz Win to Mac vrací text vyjádřený s použitím MACOS ASCII mapy stejný jako text který předáte do Text, vyjádřený pomocí Windows ANCI mapy.

Příkaz očekává parametr text vyjádřený pomocí Windows ANSI mapy.

Pokud pracujete pouze na Windows, nebudete potřebovat tento příkaz pro převod ASCII kódů. Pokud kopírujete text mezi 4D a Windows nebo když importujete nebo exportujete data, 4th Dimension automaticky provede tento převod. Nicméně když používáte diskové číst/psát příkazy jako SEND PACKET nebo RECEIVE PACKET, budete potřebovat ASCII převody. Toto je hlavní účel příkazu Win to Mac.

Uvnitř 4D jsou všechny textové hodnoty, pole nebo proměnné které jste nepřevedli na jinou ASCII mapu jsou kódovány na Macintosh jak na Windows tak na Macintoshi. Jestli chcete vědět více informací, přečtěte si část ASCII kódy.

Příklad

Pokud načítáte znaky z dokumentu pomocí RESIEVE PACKET a nepoužijete vstupní ASCII mapu pro převod znaků z Windows na MacOS (podívejte se na příkaz USE ASCII MAP), budete potřebovat převést text z Windows na MacOS osobně. Můžete to udělat takto:

```
` ...
RECEIVE PACKET ($vhDokumRef;vtNejakyText;16*1024)
vtNejakyText:=Win to Mac(vtNejakyText)
` ...
```

Dále si přečtěte

ASCII kódy, Mac to Win, RECEIVE PACKET, USE ASCII MAP.

Příkazy a odkazy pro Text řetězce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Mac to ISO (Mac do ISO)

Příkazy a odkazy pro Text řetězce

Verze 6.0

Mac to ISO (text) → Řetězec

Parametr	Typ		Popis
text	Řetězec	→	Text vyjádřený s použitím MacOS ASCII mapy
Výsledek funkce	Řetězec	→	Text vyjádřený v tabulce ISO Latin-1, nebo při lokalizované 4D v ASCII Win1250

Popis

Příkaz Mac to ISO vrací text vyjádřený pomocí znakové mapy ISO Latin-1 (nebo ASCII Win1250), stejný jako předáný text vyjádřený s použitím MacOS ASCII mapy.

Tento příklad nebudete většinou používat.

Tento příkaz očekává text vyjádřený pomocí MacOS ASCII mapy.

4D převádí znaky obdržené a poslané na Web prohlížeč. Jako výsledek texty se kterými pracujete uvnitř procesu Web spojení jsou vždy vyjádřeny pomocí MacOS ASCII mapy.

Pokud pracujete pouze na Windows, nebudete potřebovat tento příkaz pro převod ASCIIkódů. Pokud kopírujete text mezi 4D a Windows nebo když importujete nebo exportujete data, 4th Dimension automaticky provede tento převod. Nicméně když používáte diskové číst/psát příkazy jako SEND PACKET nebo RECEIVE PACKET, budete potřebovat ASCIIpřevody.

Uvnitř 4D jsou všechny textové hodnoty, pole nebo proměnné které jste nepřevedli na jinou ASCII mapu jsou kódovány na Macintosh jak na Windows tak na Macintoshi. Jestli chcete vědět více informací, přečtěte si část ASCII kódy.

Na Windows je potřebné že v tomto případě nebudete filtrovat znaky s použitím výstupní ASCII mapy.

Proto nezáleží na jaké platformě jste, ale když potřebujete zapsat ISO Latin-1 HTML dokument na disk, potřebujete pouze převést text pomocí Mac to ISO. Toto je hlavní účel Mac to ISO.

Příklady

1. Následující řádek kódu převede MacOS text uložený v vtNejakyText do ISO Latin-1 (v lokalizované verzi ASCII Win1250):

```
vtNejakyText:=Mac to ISO (vtNejakyText)
```

2. Při vývoji aplikace pro Web, vytváříte HTML stránky, které později pošlete přes Internet nebo Intranet s použitím příkazu SEND HTML FILE. Některé z těchto dokumentů mají odkazy na jiné dokumenty. Následující metoda projektu zjistí cestu HTML z Windows nebo Macintosh cesty zadané jako parametr:

```
   ` Metoda projektu HTML cesta
   ` HTML cesta ( Text ) → Text
   ` HTML cesta ( File manažer cesta ) → HTML cesta
C_TEXT($0;$1)
C_LONGINT($vlChar;$vlAscii)
C_STRING(31;$vsChar)
$0:=""
If (Na windows  )
   $1:=Replace string($1;";"/")
```

```
Else
   $1:=Replace string($1;":";"/")
End if
$1:=Mac to ISO($1)
For ($vlChar;1;Length($1))
   $vlAscii:=Ascii($1[[$vlChar]])
   Case of
      ÷ ($vlAscii>=127)
         $vsChar:="%"+Substring(String($vlAscii;"&$");2)
      ÷ (Position(Char($vlAscii);":<>&%= "+Char(34))>0)
         $vsChar:="%"+Substring(String($vlAscii;"&$");2)
   Else
      $vsChar:=Char($vlAscii)
   End case
   $0:=$0+$vsChar
End for
```

Poznámka: Metoda projektu Na windows je zobrazena v části Systémové dokumenty.

Jakmile je tato metoda předána do databáze, jestliže chcete mít seznam FTP odkazů na dokumenty obsažené v patřičné složce, můžete napsat:

```
   ` Nastavení meziprocesních proměnných, například, v metodě databáze Při spuštění
<>vsFTPURL:="ftp://123.4.56.78/Spiders/"
<>vsFTPDirectory:="APS500:Spiders:" ` Zde je MacOS cesta
   ` ...
   ` ...
ARRAY STRING(31;$asDokumenty;0)
DOCUMENT LIST(...;$asDokumenty)
$vlNbDokumenty:=Size of array($asDokumenty)
jsHandler:=...
For ($vlDocument;1;$vlNbDokumenty)
   vtHTMLCode:=vtHTMLCode+"<P><A HREF="+Char(34)+<>vsFTPURL+
      HTML cesta (Substring($1+$asDokumenty{$vlDocument};
      Length (<>vsFTPDirectory)+1))+Char(34)+jsHandler+">"+
      $asDokumenty{$vlDocument} +"</A></P>"+Char(13)
End for
   ` ...
```

Dále si přečtěte

ASCII kódy, ISO to Mac, SEND HTML FILE, SEND PACKET, USE ASCII MAP.

Příkazy a odkazy pro Text řetězce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ISO to Mac (ISO do Mac)

Příkazy a odkazy pro Text řetězce

Verze 6.0

ISO to Mac (text) → Řetězec

Parametr	Typ		Popis
text	Řetězec	→	Text vyjádřený v tabulce ISO Latin-1, nebo při lokalizované 4D v ASCII Win1250
Výsledek funkce	Řetězec	←	Text vyjádřený s použitím MacOS ASCII mapy

Popis

Příkaz ISO to Mac vrací text vyjádřený pomocí MacOS ASCII mapy stejný jako předaný text vyjádřený s použitím

Tento příklad nebudete většinou používat.

Tento příkaz očekává text vyjádřený pomocí znakové mapy ISO Latin-1 (nebo ASCII Win1250).

4D převádí znaky obdržené a poslané na Web prohlížeč. Jako výsledek texty se kterými pracujete uvnitř procesu Web spojení jsou vždy vyjádřeny pomocí MacOS ASCIImapy.

Pokud pracujete pouze na Windows, nebudete potřebovat tento příkaz pro převod ASCII kódů. Pokud kopírujete text mezi 4D a Windows nebo když importujete nebo exportujete data, 4th Dimension automaticky provede tento převod. Nicméně když používáte diskové číst/psát příkazy jako SEND PACKET nebo RECEIVE PACKET, budete potřebovat ASCII převody.

Uvnitř 4D jsou všechny textové hodnoty, pole nebo proměnné které jste nepřevedli na jinou ASCII mapu jsou kódovány na Macintosh jak na Windows tak na Macintoshi. Jestli chcete vědět více informací, přečtěte si část ASCII kódy.

Na Windows je potřebné že, v tomto případě, nebudete filtrovat znaky s použitím výstupní ASCII mapy.

Proto nezáleží na jaké platformě jste, ale když potřebujete použít RECEIVE PACKET k načtení ISO Latin-1 HTML dokument z disku, potřebujete pouze převést text pomocí ISO to Mac. Toto je hlavní účel příkazu ISO to Mac.

Příklad

Následující řádek kódu převede ISO Latin-1 (v lokalizované verzi ASCII Win1250) kódovaný text uložený v vtNejakyText na MacOS kódovaný text:

```
RECEIVE PACKET ($vhDokumRef;vtNejakyText;16*1024) ` Nejprve z ISO Latin-1 HTML dokumentu
vtNejakyText:=ISO to Mac(vtNejakyText)
```

Dále si přečtěte

ASCII kódy, Mac to ISO, RECEIVE PACKET, USE ASCII MAP.

Příkazy a odkazy pro Text řetězce

4th Dimension 6.5, Seznam témat příkazů

Struktura

Příkazy a odkazy pro Přístup k struktuře

Verze 6.0

Příkazy v této části vrací informace o struktuře databáze. Vrací počet tabulek, početpolí v každé tabulce, názvy tabulek a polí a typ a vlastnosti každého pole.

Určování struktury databáze je velmi výhodné pokud vyvíjíte a používáte skupiny metod projektu a formulářů, které mohou být kopírovány do různých databází.

Možnost číst strukturu vám umožní vyvíjet a používat přenosný kód.

Dále si přečtěte

Count fields, Count tables, Field, GET FIELD PROPERTIES, Ukazatele, SET INDEX, Table, Table name.

Příkazy a odkazy pro Přístup k struktuře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Count tables (Počet tabulek)

Příkazy a odkazy pro Přístup k struktuře

Verze 3

Count tables → Číslo

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Číslo	←	Počet tabulek v databázi

Popis

Count tables vrací počet tabulek v databázi. Tabulky jsou číslovány v pořadí v jakém jsou vytvořeny.

Příklad

Následující příklad vytvoří array, nazvaný asTables, s názvy tabulek v databázi. Tento array může být použit ve formuláři jako rozevírací seznam (nebo ovládací karta, posuvná oblast, atd.) k zobrazení seznamu tabulek:

```
ARRAY STRING (31;asTables;Count tables)
For ($vlTabulka; 1; Size of array(asTables))
   asTables{$vlTabulka}:=Table name ($vlTabulka)
End for
```

Dále si přečtěte

Array, Count fields, Table name.

Příkazy a odkazy pro Přístup k struktuře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Count fields (Počet polí)

Příkazy a odkazy pro Přístup k struktuře

Verze 3

Count fields (tabulkaČíslo | ukazTabulky) → Číslo

Parametr	Typ		Popis
tabulkaČíslo	Číslo	→	Číslo tabulky
| ukazTabulky	| Ukazatel		nebo ukazatel na tabulku
Výsledek funkce	Číslo	←	Počet polí v tabulce

Popis

Příkaz Count fields vrací počet polí v tabulce, jejíž číslo nebo ukazatel jste zadali do parametru tabulkaČíslo nebo ukazTabulky.

Pole jsou číslovány podle pořadí ve kterém jsou vytvořeny.

Příklad

Následující metoda projektu vytvoří array asFields, který obsahuje názvy polí pro tabulku, jejíž ukazatel je předán jako první parametr:

```
$vlTabulka:=Table($1)
ARRAY STRING(31;asFields;Count fields($vlTabulka))
For ($vlPole;1;Size of array(asFields))
   asFields{$vlTabulka}:=Field name($vlTabulka;$vlPole)
End for
```

Dále si přečtěte

Array, Count tables, Field name, GET FIELD PROPERTIES.

Příkazy a odkazy pro Přístup k struktuře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Table name (Název tabulky)

Příkazy a odkazy pro Přístup k struktuře

Verze 3

Table name (tabulkaČíslo | ukazTabulky) → Řetězec

Parametr	Typ		Popis
tabulkaČíslo	Číslo	→	Číslo tabulky
| ukazTabulky	| Ukazatel		nebo ukazatel na tabulku
Výsledek funkce	Řetězec	←	Název tabulky

Popis

Příkaz Table name vrací název tabulky, jejíž číslo nebo ukazatel jste zadali do parametru tabulkaČíslo nebo ukazTabulky.

Příklad

Následující příklad je generická metoda projektu, která zobrazí záznamy tabulky. Odkaz na tabulku je předán jako parametr metody. Table name je použit pro získání názvu tabulky k umístění do titulu okna:

```
` Metoda projektu ZOBRAZIT PLATNÝ VÝBĚR
` ZOBRAZIT PLATNÝ VÝBĚR ( Ukazatel )
` ZOBRAZIT PLATNÝ VÝBĚR (→[Tabulka])
SET WINDOW TITLE("Záznamy pro "+Table name($1)) ` Nastavit titul okna
DISPLAY SELECTION($1→) ` Zobrazit výběr
```

Dále si přečtěte

Count tables, Field name, Table.

Příkazy a odkazy pro Přístup k struktuře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Field name (Název pole)

Příkazy a odkazy pro Přístup k struktuře

Verze 3

Field name (tabulkaČíslo; poleČíslo) | (ukazPole) → Řetězec

Parametr	Typ		Popis
tabulkaČíslo	Číslo	→	Tabulka číslo
poleČíslo | ukazPole	Číslo	→	Číslo pole jestli je předáno číslo tabulky, jinak ukazatel na pole
Výsledek funkce	Řetězec	←	Název pole

Popis

Field name vrací název pole jehož ukazatel zadáte do ukazPole nebo jehož číslo tabulky a pole předáte do tabulkaČíslo a poleČíslo.

Příklady

1. Tento příklad nastaví druhý prvek array FieldArray{1} na název druhého pole první tabulky. FieldArray je dvou-rozměrný array:

```
FieldArray{1}{2}:=Field name(1;2)
```

2. Tento příklad nastaví druhý prvek array FieldArray{1} na název pole [Tabulka1]MojePole. FieldArray je dvou-rozměrný array:

```
FieldArray{1}{2}:=Field name(→[Tabulka1]MojePole)
```

3. Tento příklad zobrazí upozornění. Tato metoda má přiřazen ukazatel na poli:

```
ALERT("Číslo ID pro pole "+Field name($1)+" v tabulce "+
   Table name(Table($1))+" bude delší než pět znaků.")
```

Dále si přečtěte

Count fields, Field, Table name.

Příkazy a odkazy pro Přístup k struktuře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Table (Tabulka)

Příkazy a odkazy pro Přístup k struktuře

Verze 3

Table (tabulkaČíslo | Ukazatel) → Ukazatel | Číslo

Parametr	Typ		Popis
tabulkaČíslo	číslo	→	Dvě syntaxe v jedné; Tabulkačíslo, pořadové číslo tabulky v struktuře,
| ukazatel	| ukazatel		nebo ukazatel, ukazatel na tabulku či některé její pole
Výsledek funkce	ukazatel | číslo	←	Ukazatel na tabulku, je-li předáno číslo, nebo pořadové číslo tabulky v stuktuře, je-li předán ukazatel

Popis

Příklad Table má tři formy:

- Pokud předáte číslo tabulky do tabulkaČíslo, Table vrátí ukazatel na tabulku
- Pokud předáte ukazatel na tabulku do ukazatel, Table vrátí číslo tabulky
- Pokud předáte ukazate na pole do ukazatel, Table vrátí číslo tabulky pro pole

Příklady

1. Tento příklad nastaví proměnnou tablePtr na ukazatel na třetí tabulce databáze:

```
tablePtr:=Table (3)
```

2. Předáním TablePtr (ukazatel na třetí tabulku) do Table vrátí se číslo 3.
Následující řádek nastaví TableNum na 3:

```
TableNum:= Table (TanlePtr)
```

3. Následující příklad nastaví TableNum na číslo tabulky [Table3]:

```
TableNum:= Table (→[Table3])
```

4. Tento příklad nastaví TableNum na číslo tabulky obsahující pole [Table3]Pole1:

```
TableNum:=  Table (→[Table3]Pole1)
```

Dále si přečtěte

Count tables, Field, Ukazatele, Table name.

Příkazy a odkazy pro Přístup k struktuře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Field (Pole)

Příkazy a odkazy pro Přístup k struktuře

Verze 3

Field (tabulkačíslo | UkazatelPole{; polečíslo}) → Číslo | Ukazatel

Parametr	Typ		Popis
tabulkačíslo | ukazatelpole	číslo | ukazatel	→	Dvě syntaxe v jedné pořadové číslo tabulky, nebo ukazatel na pole tabulky
Polečíslo	číslo	→	Je-li uvedeno číslo tabulky, povinné, pořadové číslo pole v tabulce
Výsledek funkce	ukazatel | číslo	←	ukazatel na pole, jsou-li předána čísla nebo pořadové číslo pole v tabulce je-li předán ukazatel

Popis

Příkaz Field má dvě formy:

- Pokud předáte číslo tabulky do tabulkaČíslo a číslo pole do Polečíslo, Field vrátí ukazatel na poli
- Pokud předáte ukazatel na pole do ukazatelpole, Field vrátí číslo pole.

Příklady

1. Následující příklad nastaví proměnnou FieldPtr na ukazatel na druhému poli třetí tabulky:

```
FieldPtr:=Field(3; 2)
```

2. Přiřazením FieldPtr (ukazatel na druhé pole třetí tabulky) do Field vrátí 2. Následující řádek nastaví FieldNum na 2:

```
FieldNum:=Field(FieldPtr)
```

3. Následující příklad nastaví proměnnou FieldNum na číslo pole [Tabulka3]Field2:

```
FieldNum:=Field(→[Tabulka3]Field2)
```

Dále si přečtěte

Count fields, Field name, GET FIELD PROPERTIES, Table.

Příkazy a odkazy pro Přístup k struktuře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GET FIELD PROPERTIES (ZÍSKAT VLASTNOSTI POLE)

Příkazy a odkazy pro Přístup k struktuře

Verze 3

GET FIELD PROPERTIES (tabulkačíslo; polečíslo | ukazpole; poleTyp{; poleDel{; indexované}})

Parametr	Typ		Popis
tabulkačíslo	číslo	→	číslo tabulky
polečíslo	číslo	→	číslo pole, je-li předáváno číslo tabulky
| ukazpole	| ukazatel		nebo ukazatel na pole
poletyp	číslo	←	typ pole
poleDel	číslo	←	délka pole, je-li typ alfa
indexované	logické	←	true - indexované, false - neindexované

Popis

Příkaz GET FIELD PROPERTIES vrací informace o poli definovaném v *ukazpole* nebo *tabulkačíslo* a *polečíslo*.

Můžete vložit:

• číslo pole a tabulky do tabulkačíslo a polečíslo nebo
• ukazatel na poli do ukazpole

Po provedení příkazu:
• poletyp bude obsahovat typ pole. Může obsahovat jednu z následujících konstant:

Konstanta	*Typ*	*Hodnota*
Is Alpha Field	*Long Integer*	*0*
Is Text	*Long Integer*	*2*
Is Real	*Long Integer*	*1*
Is Integer	*Long Integer*	*8*
Is LongInt	*Long Integer*	*9*
Is Date	*Long Integer*	*4*
Is Time	*Long Integer*	*11*
Is Boolean	*Long Integer*	*6*
Is Picture	*Long Integer*	*3*
Is SubTabulka	*Long Integer*	*7*
Is BLOB	*Long Integer*	*30*

• PoleDel vrací délku pole, když je typ pole Alfa. PoleDel je zbytečné pro jiné typy polí.
• Parametr indexované vrací True pokud je pole indexované a False pokud není. Tato hodnota je využitená pole pro typy polí Alfa, Integer, Long Integer, Real, Datum, Čas a Logické.

Příklady

1. Tento příklad nastaví proměnné vType, vLength a v Index na vlastnosti třetího pole první tabulky:

```
GET FIELD PROPERTIES(1; 3;vType;vLength;vIndex)
```

2. Tento příklad nastaví proměnné vType, vLength a vIndex na vlastnosti pole s názvem [Tabulka3]Field2:

```
GET FIELD PROPERTIES(→[File3]Field2;vType;vLength;vIndex)
```

Dále si přečtěte

Field, Field name, SET INDEX.

Příkazy a odkazy pro Přístup k struktuře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET INDEX (Nastavit index)

Příkazy a odkazy pro Přístup k struktuře

Verze 3

SET INDEX (pole; index{; mod{; *}})

Parametr	Typ		Popis
pole	pole	→	Pole, pro které chceme změnit indexování
index	logické	→	True - indexovat, False - mazat indexy
mod	longint	→	mód indexování v procentech
*		→	indexovat asynchronně

Popis

Příkaz SET INDEX vytvoří nebo odstraní indexy pro pole nebo podpole které definujete v parametru pole.

K indexování pole nebo podpole vložte Tru do parametru index. Pokud index již existuje, příkaz neprovede nic. K vymazání indexu vložte False do parametru index. Pokud index neexistuje, příkaz neprovede nic.

SET INDEX nebude indexovat zamčené záznamy; bude čekat dokud se záznamy neodemknou.

Od verze 6.5 můžete vybrat mezi dvěma způsoby indexování: "tradiční" způsob, který byl používán v předchozích verzích 4D a "rychlý" způsob, který ve většině případech o mnoho rychlejší. Více informací najdete v *Příručce návrháře 4th Dimension.*

Způsob indexování volíte určením volitelného parametru. Parametr mod je použitelný pouze při vytváření nových indexů (když je index True).

• Pokud nepředáte parametr mod, bude indexování provedeno **tradičním způsobem**. V tomto případě je indexování prováděno v odděleném procesu a během indexování můžete pracovat s databází. Pokud během vytváření indexu budete provádět akci, která používá indexy, indexy nebudou použity. K zjištění jestli je pole indexované použijte příkaz GET FIELD PROPERTIES.

• Pokud předáte parametr mod, bude indexování provedeno **rychlým způsobem**. V tomto případě nebude možné měnit data během vytváření indexů. Do parametru mod musíte vložit Integer hodnotu která bude zastupovat procenta indexování. Tato hodnota vám umožní definovat pro jaký styl použití bude index nejvýkonnější. Tato hodnota musí být v následujícím rozmezí:
- mod : = 0: index bude nejvýkonější při vkládání záznamů
- mod : = 100: index bude nejvýkonější při provádění dotazů.

Volitelný parametr * definuje asynchronní (simultánní) indexování. Asynchronní indexování umožní provádění jiných metod k okamžitému pokračování ať už je nebo není index dokončen. Nicméně indexování bude zastaveno při každém příkazu který používá indexy.

Příklady

1. Následující příklad indexuje pole [Zákazníci]ID tradičním způsobem:

```
UNLOAD RECORD ([Zákazníci])
SET INDEX ([Zákazníci]ID; True)
```

2. Chcete indexovat pole [Zákazníci]Jméno rychlým způsobem. Toto pole je nejčastěji používáno k provádění dotazů:

```
SET INDEX ([Zákazníci]Jméno; True; 100)
```

3. Chcete indexovat pole [Kontakty]Jméno rychlým způsobem. Toto pole je často používáno pro vkládání a přidávání jmen, ale používá se i dotazům:

```
SET INDEX ([Kontakty]Jméno;True;30)
```

Dále si přečtěte

GET FIELD PROPERTIES, ORDER BY, QUERY.

Příkazy a odkazy pro Přístup k struktuře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Get database parameter (Získat parametr databáze)

Příkazy a odkazy pro Přístup k struktuře

Verze 6.5

Get database parameter ({tabulka;} volba) → Longint

Parametr	Typ		Popis
tabulka	tabulka	→	Číslo tabulky nebo Výchozí tabulka pokud vynecháno
volba	Longint	→	kód parametru databáze
Výsledek funkce	Longint	←	Platná hodnota parametru

Popis

Příkaz Get database parameter vám umožní zjistit, pro platný proces, hodnotu parametru databáze 4D.

Parametr volba označuje parametr k přečtení. 4th Dimension obsahuje následující konstanty které jsou v tématu "Database Parameters":

Konstanta	***Typ***	***Hodnota***
Seq Order Ratio	*Longint*	*1*
Seq Access Optimization	*Longint*	*2*
Seq Distinct Values Ratio	*Longint*	*3*
Index Compacting	*Longint*	*4*
Seq Query Select Ratio	*Longint*	*5*
Minimum Web Process	*Longint*	*6*
Maximum Web Process	*Longint*	*7*
Web conversion mode	*Longint*	*8*
Database Cache Size	*Longint*	*9*

K zjištění jaké hodnoty můžete dostat z parametrů 1 až 8 se podívejte do popisu příkazu SET DATABASE PARAMETER.

Volba Database Cach Size (9) vám umožní získat platnou paměť používanou databází. Je vyjádřená v bytech.

Maximální velikost paměti můžete nastavit jak na Windows tak na Macintoshi, ale pouze na Macintoshi můžete nastavit minimální paměť. Tyto vlastnosti jsou přístupné přes okno Vlastnosti databáze. Aktuální velikost paměti databáze bude záviset na nastavení a na velikosti a na platných zdrojích systému. Příkaz Get database parameter vám umožní získat aktuální velikost paměti přiřazené k databázi od 4D.

Poznámka: Velikost paměti nemůžete nastavit pomocí jazyka. Jinými slovy nemůže být Database Cache Selector nastaven pomocí SET DATABESE PARAMETER.

Dále si přečtěte

DISTINCT VALUES, QUERY SELECTION, SET DATABASE PARAMETER.

Příkazy a odkazy pro Přístup k struktuře

4th Dimension 6.5, Seznam témat příkazů

SET DATABASE PARAMETER (NASTAVIT PARAMETR DATABÁZE)

Příkazy a odkazy pro Přístup k struktuře

Verze 6.5

SET DATABASE PARAMETER ({tabulka;} volba; hodnota)

Parametr	Typ		Popis
tabulka	tabulka	→	Tabulka, pro kterou nastavit parametry, nebo výchozí tabulka, není-li uveden
volba	Longint	→	kód parametru databáze, který má být změněn
hodnota	Longint	→	hodnota parametru

Popis

Příkaz SET DATABASE PARAMETER vám umožní upravit pro platný proces různé vnitřní parametry databáze 4D.

Parametr volba označuje parametr k přečtení. 4th Dimension obsahuje následující konstanty které jsou v tématu "Database Parameters":

Konstanta	*Typ*	*Hodnota*
Seq Order Ratio	*Longint*	*1*
Seq Access Optimization	*Longint*	*2*
Seq Distinct Values Ratio	*Longint*	*3*
Index Compacting	*Longint*	*4*
Seq Query Select Ratio	*Longint*	*5*
Minimum Web Process	*Longint*	*6*
Maximum Web Process	*Longint*	*7*
Web conversion mode	*Longint*	*8*
Database Cache Size	*Longint*	*9*

Hodnota označuje hodnotu paramatru. Předáná hodnota závisí na tom, jaký parametr chcete upravovat. Zde jsou možné hodnoty pro volbu:

Volba = 1 (Seq Order Ratio)

• Hodnota: 0 → 100000

• Popis: Poměr (mezi počtem vybraných záznamů a celkovým počtem záznamů v tabulce) ve kterém je prováděno třídění v postupném módu. Tento poměr musí být násoben sto tísíci.Výchozí hodnota je 9000 (= 9%).

Volba = 2 (Seq Access Optimization)

• Hodnota: 0 nebo 1 (0 = neoptimalizováno, 1 = optimalizováno)

• Popis = Mód optimalizace pro postupné přístupy (třídění, hledání nebo výběr do array). V optimalizovaném módu 4D načte několik záznamů z disku najednou, ale neumístí je do paměti. Tento mód je použitelný hlavně v případě malé paměti počítače. Jako výchozí je nastavena hodnota 1 (optimalizováno).

Volba = 3 (Seq Distinct Values Ratio)

• Hodnota: 0 → 100000

• Popis = Poměr (mezi počtem vybraných záznamů a celkovým počtem záznamů v tabulce) ve kterém je prováděn příkaz DISTINCT VALUES v postupném módu. Tento poměr musí být násoben sto tísíci.

Volba = 4 (Index Compacting)

• Hodnota: 0 nebo 1 (0 = nekompaktovat, 1 = kompaktovat)

• Popis: Tato volba vám umožní zapnout nebo vypnout kompaktování stránek indexu. Jako výchozí je hodnota 1 (pokud je to možné jsou stránky indexu kompaktovány). Stránky indexu, pokud databáze obsahuje mnoho indexů, mohou zabírat velké množství paměti. Pokud je paměť plná a 4D potřebuje další místo, data v paměti nebudou

odstraněna. Před odstraněním dat z paměti, program otestuje jestli může získat nějaké místo kompaktováním stránek indexů. Tato možnost vám umožní vyhnout se pozdnímu načítání dat.

Volba = 5 (Seq Query Select Ratio)

• Hodnota: 0 → 100000
• Popis = Poměr (mezi počtem vybraných záznamů a celkovým počtem záznamů v tabulce) ve kterém je prováděn příkaz QUERY SELECTION v postupném módu. Tento poměr musí být násoben sto tísíci.

Volba = 6 (Minimum Web Process)

• Hodnota: 0 → 32767
• Popis: Minimální počet Web procesů v nekontextním módu. Jako výchozí je tato hodnota nastavena na 0

Volba = 6 (Minimum Web Process)

• Hodnota: 0 → 32767
• Popis: Maximální počet Web procesů v nekontextním módu. Jako výchozí je tato hodnota nastavena na 10. V nekontextním módu, pro Web server schopný reakce, 4D pozdrží proces 5 sekund a pak jej použije pro provedení jiného HTML dotazu. Z důvodů výkonnosti je to mnohem výhodnější než vytářet nový proces pro každý dotaz. Jakmile je proces použit, počká dalších 5 sekund. Jakmile je dosaženo maximálního počtu Web procesů, proces je pak odstraněn. Pokud není po 5-ti sekundách žádný dotaz, je proces odstraněn, pokud není dosaženo minimálního počtu Web procesů (v tom případě je proces opět pozdržen na 5 sekund). Tento parametr vám umožní nastavit jak budou vaše Web procesy pracovat ve vztahu k počtu žádostí, dostupné paměti a stejně tak i k ostatním parametrům.

Volba = 8 (Web conversion mode)

• Hodnota: 1 nebo 2
1 = 6.0.x mód konverze
2 = 6.5 mód konverze

Jako výchozí je nastavena hodnota 2.

• Popis: V některých případech Web konvence formulářů 4D vytvořených ve verzi 6.0.x může být neshodná s 4D 6.5, přesněji formuláře které obsahují odkazy na HTML stránky jako {Mojestránka.htm}. V tomto případě, když chcete zajistit kompatibilitu formulářů, můžete aktivovat 4D 6.0.x mód konverze. Tento mód je nastaven pouze pro proces ze kterého byl volán příkaz SET DATABASE PARAMETER. Tento příkaz může být proveden z metody databáze Při Web spojení a zajistit tím 6.0.x kompatibilitu všech formulářů v databázi, nebo před jedním zobrazením formuláře. Pokud je příkaz volán mimo kontextový mód nebo mimo Web proces, neprovede nic.

Poznámka: S příkazem Get database parameter může být použita ještě jedna volba: Database Cache Size (9). Tato volba nemůže být použita s příkazem SET DATABASE PARAMETER. Více informací naleznete v popisu příkazu Get database parameter.

Dále si přečtěte

DISTINCT VALUES, Get database parameter, QUERY SELECTION.

Příkazy a odkazy pro Přístup k struktuře

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CREATE SUBRECORD (VYTVOŘIT PODZÁZNAM)

Příkazy a odkazy pro Podzáznamy

Verze 3

CREATE SUBRECORD (podtabulka)

Parametr	Typ		Popis
podtabulka	Podtabulka	→	Podtabulka pro kterou vytvořit podzáznam

Popis

CREATE SUBRECORD vytvoří nový podzáznam pro podtabulku a udělá nový podzáznam platným podzáznamem. Podzáznam je uložen pouze pokud je uložen rodičovský záznam. Rodičovský záznam může být uložen příkazem SAVE RECORD nebo použitím příkazu pro potvrzení. Pokud pro tabulku není platný záznam, CREATE SUBRECORD neprovede nic. Pro předání podzáznamu přes vstupní formulář podtabulky, použijte příkaz ADD SUBRECORD.

Příklad

Nsáledující příklad je metoda objektu pro tlačítko. Když je spuštěna (na tlačítko bylo klepnuto) vytvoří podzáznam pro dítě v tabulce [Lidé]. Smyčka Repear umožní uživateli zadávat podzáznamy dokud neklepne na Zrušit. Formulář zobrazí děti v podformuláři, ale neumožní zadávat data, protože vlastnost Dostupné byla odznačena:

```
Repeat
      ` Získat jméno dítěte
   vChild := Request("Jméno (Zrušit když hotovo):")
      ` Pokud uživatel klepne na OK
   If (OK = 1)
         ` Vložit nový podzáznam pro děti
      CREATE SUBRECORD([Lidé]Děti)
         ` Přiřadit jméno dítěte do pole podtabulky
      [Lidé]Děti'Jméno:=vChild
   End if
Until (OK=0)
```

Dále si přečtěte

ADD SUBRECORD, DELETE SUBRECORD, SAVE RECORD.

Příkazy a odkazy pro Podzáznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DELETE SUBRECORD (VYMAZAT PODZÁZNAM)

Příkazy a odkazy pro Podzáznamy

Verze 3

DELETE SUBRECORD (podtabulka)

Parametr	Typ		Popis
podtabulka	Podtabulka	→	Podtabulka pro kterou vymazat podzáznam

Popis

DELETE SUBRECORD vymaže platný podzáznam podtabulky. Pokud není žádný platný podzáznam, DELETE SUBRECORD neprovede nic. Po vymazání podzáznamu bude patný výběr pro podtabuku prázdný. DELETE SUBRECORD nemůže být použit pro prohlížení podvýběru a mazání označených záznamů.

Vymazání podzáznamu není trvalé, dokud není uložen rodičovský záznam. Vymazání rodičovského záznamu automaticky vymaže jeho podzáznamy.

K vymazání podvýběru který chcete vymazat, vytvořte podvýběr, vymažte první podzáznam, znovu vytvořte podvýběr a vymažte první záznam, atd.

Příklady

1. Následující příklad vymaže všechny podzáznamy podtabulky:

```
ALL SUBRECORDS([Lidé]Děti)
While (Records in subselection([Lidé]Děti)>0)
   DELETE SUBRECORD([Lidé]Děti)
   ALL SUBRECORDS([Lidé]Děti)
End while
```

2. Následující příklad vymaže podzáznamy ve kterých je věk dítěte vyšší než 12 let:

```
ALL RECORDS([Lidé]) ` Vybrat všechny záznamy
For ($vlZaznam;1;Records in selection([Lidé])) ` Pro všechny záznamy
      ` Hledat všechny záznamy s podzáznamem s kritérii
   QUERY SUBRECORDS([Lidé]Děti;[Lidé]Children'Věk>=12)
      ` Opakovat dokud nenalezen žádný podzáznam
   While (Records in subselection([Lidé]Děti)>0)
         ` Vymazat podzáznam
      DELETE SUBRECORD([Lidé]Děti)
         ` Hledat znovu
      QUERY SUBRECORDS([Lidé]Děti;[Lidé]Děti'Věk>=12)
   End while
   SAVE RECORD([Lidé]) ` Uložit rodičovský záznam
   NEXT RECORD([Lidé])
End for
```

Dále si přečtěte

ALL SUBRECORDS, QUERY SUBRECORDS, Records in subselection, SAVE RECORD.

Příkazy a odkazy pro Podzáznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ALL SUBRECORDS (VŠECHNY PODZÁZNAMY)

Verze 3

ALL SUBRECORDS (podtabulka)

Parametr	Typ		Popis
podtabulka	Podtabulka	→	Podtabulka pro kterou vybrat všechny podzáznamy

Popis

ALL SUBRECORDS vybere všechny podzáznamy podtabulky do platného podvýběru. Pokud neexistuje platný rodičovský záznam, ALL SUBRECORDS neprovede nic. Když je rodičovský záznam nejdříve načten, podvýběr obsahuje všechny podzáznamy. Podvýběr nemusí obsahovat všechny podzáznamy po provedení příkazů ADD SUBRECORD, QUERY SUBRECORDS nebo DELETE SUBRECORD.

Příklad

Následující příklad vybere všechny podzáznamy aby mohly být všechny sečteny:

```
ALL SUBRECORDS ([Stav]Prodeje)
Celkem := Sum ([Stav]Prodeje'Částka)
```

See Also

QUERY SUBRECORDS, Records in subselection.

Příkazy a odkazy pro Podzáznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Records in subselection (záznamů v podvýběru)

Příkazy a odkazy pro Podzáznamy

Verze 3

Records in subselection (podtabulka) → Číslo

Parametr	**Typ**		**Popis**
podtabulka	Podtabulka	→	Podtabulka pro kterou sečíst podzáznamy
Výsledek funkce	Číslo	←	Počet podzáznamů v platném podvýběru

Popis

Records in subselection vrací počet podzáznamů v platném podvýběru podtabulky. Records in subselection je použit pouze na podzáznamy platného záznamu. Příkaz má stejnou funkci jako Records in selection, ale na podzáznamy. Výsledek není definován, pokud neexistuje platný záznam.

Příklad

Následující příklad vybere všechny podzáznamy a zobrazí počet potomků pro rodičovský záznam:

```
` Zobrazit všechny děti pro zjištění počtu
ALL SUBRECORDS ([Lidé]Děti)
ALERT ("Počet dětí: "+String(Records in subselection ([Lidé]Děti)))
```

Dále si přečtěte

ALL SUBRECORDS, QUERY SUBRECORDS.

Příkazy a odkazy pro Podzáznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

APPLY TO SUBSELECTION (POUŽÍT NA PODVÝBĚR)

Verze 3

APPLY TO SUBSELECTION (podtabulka; tvrzení)

Parametr	Typ		Popis
podtabulka	Podtabulka	→	Podtabulka pro kterou sečíst podzáznamy
tvrzení	Řetězec	→	Jeden řádek kódu nebo název metody

Popis

APPLY TO SUBSELECTION použije tvrzení na každý podzáznam v platném podvýběru podtabulky. Tvrzení může být výraz nebo metoda. Pokud tvrzení změní podzáznam je tento podzáznam uložen pouze pokud je uložen rodičovský záznam. Když je platný podvýběr prázdný příkaz neprovede nic.

APPLY TO SUBSELECTION může získat informace z podvýběru nebo změnit podzáznam.

Příklad

Následující příklad zvětší první písmeno ve jméně [Lidé]Děti:

```
ALL SUBRECORDS ([Lidé]Děti)
APPLY TO SUBSELECTION([Lidé]Děti;[Lidé]Děti'Jméno:=
        Uppercase(Substring([Lidé]Děti'Jméno;1;1))+
        Lowercase(Substring([Lidé]Děti'Jméno;2)))
```

Poznámka: Tvrzení bylo rozděleno do několika řádků pouze účely této příručky.

Dále si přečtěte

ALL SUBRECORDS, QUERY SUBRECORDS, SAVE RECORD.

Příkazy a odkazy pro Podzáznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

FIRST SUBRECORD (PRVNÍ PODZÁZNAM)

Verze 3

FIRST SUBRECORD (podtabulka)

Parametr	Typ		Popis
podtabulka	Podtabulka	→	Podtabulka pro kterou posunout na první podzáznam

Popis

FIRST SUBRECORD udělá z prvního podzáznamu platného podvýběru platný podzáznam. Všechny příkazy dotazů, výběrů a třídění také udělají z prvního podzáznamu platný podzáznam. Pokud je platný podvýběr prázdný, příkaz neprovede nic.

Příklad

Následující příklad spojí jména a příjmení v podtabulce Děti a přesune výsledek do array atPrijmeni:

```
    ` Vytvořit array se jmény
ARRAY TEXT (atPrijmeni; Records in subselection ([Lidé]Děti))
FIRST SUBRECORD ([Lidé]Děti)
    ` Začít na prvním záznamu a pokračovat dále
For ($vlSub; 1; Records in subselection ([Lidé]Děti))
   atPrijmeni{$vlSub} := [Lidé]Děti'Jméno+" "+ [Lidé]Děti'Příjmení
   NEXT SUBRECORD ([Lidé]Děti)
End for
```

Dále si přečtěte

LAST SUBRECORD, NEXT SUBRECORD, PREVIOUS SUBRECORD.

Příkazy a odkazy pro Podzáznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

LAST SUBRECORD (POSLEDNÍ PODZÁZNAM)

Verze 3

LAST SUBRECORD (podtabulka)

Parametr	Typ		Popis
podtabulka	Podtabulka	→	Podtabulka kde se přesunout na poslední podzáznam

Popis

LAST SUBRECORD udělá z posledního podzáznamu platného podvýběru platný podzáznam. Pokud je platný podvýběr prázdný, příkaz neprovede nic.

Příklad

Následující příklad spojí jméno a příjmení záznamů uložených v podtabulce. Zkopíruje jména do array atPrijmeni. Je to stejné jako příklad u příkazu FIRST RECORD, ale budeme se přesunovat z posledního na první podzáznam:

```
    ` Vytvořit array se jmény
ARRAY TEXT (atPrijmeni; Records in subselection ([Lidé]Děti))
LAST SUBRECORD ([Lidé]Děti)
    ` Začít na posledním záznamu a pokračovat nahoru
For ($vlSub; 1; Records in subselection ([Lidé]Děti))
   atPrijmeni$vlSub := [Lidé]Děti'Jméno+" "+ [Lidé]Děti'Příjmení
   PREVIOUS SUBRECORD ([Lidé]Děti)
End for
```

Dále si přečtěte

FIRST SUBRECORD, NEXT SUBRECORD, PREVIOUS SUBRECORD.

Příkazy a odkazy pro Podzáznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

NEXT SUBRECORD (DALŠÍ PODZÁZNAM)

Verze 3

NEXT SUBRECORD (podtabulka)

Parametr	Typ		Popis
podtabulka	Podtabulka	→	Podtabulka kde se přesunout na další podzáznam

Popis

NEXT SUBRECORD přesune ukazatel na platný podzáznam na další podzáznam v platném podvýběru podtabulky. Pokud NEXT SUBRECORD přesune ukazatel za poslední podzáznam, End subselection vrátí True a nebude žádný platný záznam. Pokud End subselection vrátí True, použijte příkazy FIRST SUBRECORD nebo LAST SUBRECORD k přesunutí ukazatele zpět do platného podvýběru. Pokud je podvýběr prázdný, nebo Befere selection vrátí True, NEXT SUBRECORD neprovede nic.

Příklad

Podívejte se na příklad u příkazu FIRST SUBRECORD.

Dále si přečtěte

FIRST SUBRECORD, LAST SUBRECORD, PREVIOUS SUBRECORD.

Příkazy a odkazy pro Podzáznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

PREVIOUS SUBRECORD (PŘEDCHOZÍ PODZÁZNAM)

Verze 3

PREVIOUS SUBRECORD (podtabulka)

Parametr	Typ		Popis
podtabulka	Podtabulka	→	Podtabulka kde se přesunout na předchozí podzáznam

Popis

PREVIOUS SUBRECORD přesune ukazatel na platný podzáznam na předchozí podzáznam v platném podvýběru podtabulky. Pokud PREVIOUS SUBRECORD přesune ukazatel před první podzáznam, Before subselection vrátí True a nebude žádný platný záznam. Pokud Before subselection vrátí True, použijte příkazy FIRST SUBRECORD nebo LAST SUBRECORD k přesunutí ukazatele zpět do platného podvýběru. Pokud je podvýběr prázdný, nebo End selection vrátí True, PREVIOUS SUBRECORD neprovede nic.

Příklad

Podívejte se na příklad u příkazu LAST SUBRECORD.

Dále si přečtěte

FIRST SUBRECORD, LAST SUBRECORD, NEXT SUBRECORD.

Příkazy a odkazy pro Podzáznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Before subselection (Před podvýběrem)

Příkazy a odkazy pro Podzáznamy

Verze 3

Before subselection (podtabulka) _> Logické

Parametr	Typ		Popis
podtabulka	Podtabulka	→	Podtabulka pro kterou zjistit, jestli je ukazatel na platný podzáznam před podvýběrem
Výsledek funkce	Logické	←	Ano (True) nebo Ne (False)

Popis

Before subselection vrátí True, jestliže je ukazatel na platný podzáznam před prvním podzáznamem platného podvýběru pro podtabulku. Tento příkaz se často používá k testování jestli PREVIOUS SUBRECORD posunul ukazatel na platný podzáznam před první podzáznam platného výběru. Pokud je platný podvýběr prázdný, vrátí Before subselection True.

Příklad

Následující příklad je metoda objektu pro tlačítko. Při klepnutí na tlačítko se přesune ukazatel na platný podzáznam na předchozí podzáznam. Pokud je ukazatel před prvním podzáznamem, přesune se na poslední podzáznam:

```
PREVIOUS SUBRECORD ([Lidé]Děti) ` Přesunout na předchozí
If (Before subselection ([Lidé]Děti) ` Pokud se dostaneme daleko...
   LAST SUBRECORD ([Lidé]Děti) ` přesunout na poslední pozáznam
End if
```

Dále si přečtěte

PREVIOUS SUBRECORD.

Příkazy a odkazy pro Podzáznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

End subselection (Za podvýběrem)

Příkazy a odkazy pro Podzáznamy

Verze 3

End subselection (podtabulka) → Logické

Parametr	Typ		Popis
podtabulka	Podtabulka	→	Podtabulka pro kterou zjistit, jestli je ukazatel na platný podzáznam za podvýběrem
Výsledek funkce	Logické	←	Ano (True) nebo Ne (False)

Popis

End subselection vrátí True, jestliže je ukazatel na platný podzáznam za posledním podzáznamem platného podvýběru pro podtabulku. Tento příkaz se často používá k testování jestli NEXT SUBRECORD posunul ukazatel na platný podzáznam za poslední podzáznam platného výběru. Pokud je platný podvýběr prázdný, vrátí End subseletion True.

Příklad

Následující příklad je metoda objektu pro tlačítko. Při klepnutí na tlačítko se přesune ukazatel na platný podzáznam na další podzáznam. Pokud je ukazatel za posledním podzáznamem, přesune se na poslední podzáznam:

```
NEXT SUBRECORD ([Lidé]Děti) ` Přesunout na další podzáznam
If (End subselection ([Lidé]Děti)) ` Pokud jsme příliš daleko...
   FIRST SUBRECORD ([Lidé]Děti) ` posunout na první záznam
End if
```

Dále si přečtěte

NEXT SUBRECORD.

Příkazy a odkazy pro Podzáznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ORDER SUBRECORDS BY (TŘÍDIT PODVÝBĚR DLE)

Verze 3

ORDER SUBRECORDS BY (podtabulka; podpole{; >nebo<}{; podpole2; >nebo<2; ...; podpoleN; >nebo<N})

Parametr	Typ		Popis
podtabulka	Podtabulka	→	Podtabulka kterou třídit vybrané podzáznamy
podpole	Podpole	→	Podpole podle kterého třídit
> nebo <		→	Směr třídění pro každou úroveň > třídit ve vzestupném pořadí < třídit v sestupném pořadí

Popis

ORDER SUBRECORDS BY třídí platný podvýběr podtabulky. Třídí pouze podvýběr podtabulky obsažené v platném záznamu.

Parametr směru definuje jestli třídit ve vzestupném nebo sestupném pořadí. Pokud je směr symbol "větší než" (>), bude podvýběr tříděn ve vzestupném pořadí a pokud bude směr symbol "menší než" (<), bude podvýběr tříděn v sestupném pořadí.

Můžete definovat více než jednu úroveň třídění přidáním více podpolí a symbolů třídění. Po skončení třídění bude první záznam podvýběru platným záznamem. Třídění podvýběru je dynamický proces. Podzáznamy se nikdy neukládají v tříděném pořadí. Pokud neexistuje ani platný záznam ani podzáznam vyšší úrovně, ORDER SUBRECORDS BY neprovede nic.

Jestliže formulář obsahuje podformulář tištěný v pevném rámci, tento příkaz musí být proveden jednou před tiskem Before fáze metody formuláře rodičovského formuláře.

Příklad

Následující příklad třídí podtabulku [Stav]Prodeje ve vzestupném pořadí podle podpole Dollars:

```
ORDER SUBRECORDS BY ([Stav]Prodeje; [Stav]Prodeje'; >)
```

Dále si přečtěte

QUERY SUBRECORDS.

Příkazy a odkazy pro Podzáznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

QUERY SUBRECORDS (DOTAZ NA PODZÁZNAMY)

Verze 3

QUERY SUBRECORDS (podtabulka; VýrazDotazu)

Parametr	Typ		Popis
podtabulka	Podtabulka	→	Podtabulka k hledání
VýrazDotazu	Logické	→	Výraz dotazu

Popis

QUERY SUBRECORDS hledá v podtabulce a vytvoří nový podvýběr. Je to jediný příklad který hledá v podtabulce a vrací výběr podzáznamů. VýrazDotazu je použit na každý podzáznam v podtabulce. Pokud bude výsledek výraz True, je předán podzáznam do nového podvýběru. Jakmile je dotaz dokončen, QUERY SUBRECORDS udělá z prvního podzáznamu platný podzáznam.

Nezapomeňte, že QUERY SUBRECORDS hledá pouze v podzáznamech podtabulky které patří k platnému rodičovskému záznamu a ne ve všech záznamech podtabulky. QUERY SUBRECORDS nemění platný rodičovský záznam.

VýrazDotazu testuje podpole proti nějaké proměnné nebo konstantě s použitím operátoru. VýrazDotazu může obsahovat více testů které jsou spojeny s A (&) nebo s NEBO (|). VýrazDotazu může být funkce nebo obsahovat funkci. Znak výběru náhradou (@) může být použit u řetězcových argumentů.

Pokud neexistuje ani platný záznam ani podzáznam vyšší úrovně, QUERY SUBRECORDS neprovede nic.

Příklad

Následující příklad vyhledá děti nad 10 let:

```
QUERY SUBRECORDS ([Lidé]Děti; [Lidé]Děti'Věk>10)
```

Dále si přečtěte

ALL SUBRECORDS, ORDER SUBRECORDS BY, Records in subselection.

Příkazy a odkazy pro Podzáznamy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Systémové dokumenty

Příkazy a odkazy pro Systémové dokumenty

Verze 6.0

Úvod

Všechny dokumenty a aplikace které používáte na vašem počítači jsou uloženy v **souborech** na vašem pevném disku **připojeném** nebo **umístěném** ve vašem počítači nebo na disketě nebo na jiném médiu. Ve 4th Dimension používáme výrazy **soubor** (file) nebo **dokument** (document) k odkazům na tyto dokumenty nebo aplikace. Většina příkazů v této kapitole používá termín "dokument", protože ve většině případů budete asi na disku pracovat skutečně s dokumenty (spíše než s aplikacemi nebo systémovými soubory).

Disk může být formátován jako celek nebo na více částí. Každá tato část se nazývá **jednotka**. Nezáleží na tom, jestli jsou dvě jednotky umístěny na jednom pevném disku; 4D bude s těmito jednotkami pracovat jako se samostatnými jedinci.

Jednotka může být na disku ve vašem počítači nebo na jiném disku připojeném přes síť protokolem jako NetBEUI (Windows) nebo AFP (Macintosh). Ve všech případech pracují příkazy na úrovni 4D se systémovými dokumenty na všech jednotkách stejně (pokud však používáte moduly Plug-in k rozšíření možností vaší aplikace v této doméně musíte již obvykle vědět konkrétně co děláte).

Každá jednotka má svůj **název jednotky**. Na Windows jsou jednotky definovány písmenem následovaným dvojtečkou. Obvykle A: nebo B: jsou použity k 5 1/4 nebo 3 1/2 disketových jedPoznamkak, C: obvykle označuje jednotku na které máte uložený systém (pokud vaše PC nenastavíte jinak). Jednotky od D: do Z: jsou použity pro další jednotky připojené k vašemu počítači (CD-ROM, přidavá média, síťové jednotky, atd.). Na Macintoshi mají jednotky názvy, jejichž délka je maximálně 31 znaků; jsou to názvy které vidíte na ploše.

Obyčejně umísťujete dokumenty do **složek**, které mohou obsahovat jiné složky. Není dobrý nápad shromažďovat stovky a tisíce souborů na jedné úrovni jednotky; je to nepřehledné a zpomaluje to práci vašeho systému. Na PC se složky nazývají (spíše nazývaly) **direktory**; od Windows 95 se však názvy sjednotily a jsou shodně nazývány složky. Složky jsou nazývány jak na Macintosh tak na Windows.

K jedinečnému definování dokumentu musíte znát název jednotky a název (názvy) složky kde je dokument umístěn, stejně jako název dokumentu samotného. Pokud spojíte všechny tyto názvy, vytvoříte úplnou **cestu** k dokumentu. V této cestě jsou názvy složek odděleny specielním znakem nazvaným **znak (oddělovač) složek**. Na Windows je tento znak obrácené lomítko (; na Macintoshi je tento znak dvojtečka (:).

Podívejme se na příklad. Máme dokument nazvaný Paměť.txt který je ve složce Paměti, která je ve složce Dokumenty, která je ve složce Práce.

Na Windows, pokud je vše na jednotce C:, bude cesta následující:

C: \Práce\Dokumenty\Paměti\Paměť.txt

Na Macintoshi, pokud je vše umístěno na disku Macintosh HD, bude cesta vypadat následovně:

Macintosh HD:Práce:Dokumenty:Paměti:Paměť

Na Windows je dokument následován příponou .TXT; uvidíme proč v následující části.

Na jakékoli platformě může být cesta znázorněna následovně:

NázevJedn OdSlož { NázSložky odSlož { NázSložky OdSlož {...}}} NázevDok

Všechny dokumenty (soubory) umístěné na jednotce mají mnoho charakteristik, které se obvykle nazývají **vlastnosti** nebo **předvolby tj.**; **název** dokumentu (31 znaků na Macintoshi, 8 znaků na Windows 3.1.1, 255 znaků na Windows 95 nebo NT 1.0), **typ** a **původce**.

Typ a Původce dokumentu

Na Windows má dokument **typ**. Na Macintoshi má dokument **typ** a **původce** (creator). Typ dokumentu určuje, co je to dokument a co obsahuje. Například textový dokument obsahuje nějaký text (bez dodatků stylu).

Na Windows je typ určen příponou (nazývanou **přípona souboru**) připojenou k názvu dokumentu. Například .TXT je přípona pro textový dokument Windows. Na Macintoshi je typ souboru udáván **typem souboru**, což je 4 znakový popis (na úrovni Finderu není zobrazen). Například typ souboru pro textový dokument je "TEXT".

Na Macintoshi má dokument ještě **původce**, který označuje aplikaci, která dokument vytvořila. Tento způsob na Windows neexistuje. Původce dokumentu je definován vlastností **původce souboru**, což je 4 znakový popis (není zobrazen na úrovni Finderu). Například původce dokumentu vytvořeného 4D v6 je "4D06".

DocRef: Číslo odkazu na dokument

Dokument může být **otevřený** nebo **zavřený**. S použitím příkazů 4D může být dokument otevřen pouze jedním procesem současně. Jeden proces může mít otevřeno více dokumentů a více procesů může mít otevřeno několik dokumentů, ale jeden dokument nemůže být otevřen více procesy najednou.

Dokument můžete otevřít příkazy Open document, Create document nebo Append document.

Jakmile je dokument jednou otevřen, můžete do něj zapisovat a číst z něj (podívejte se na příkazy RECEIVE PACKET a SEND PACKET). Jakmile jste dokončili práci s dokumentem, obvykle jej uzavřete pomocí příkazu CLOSE DOCUMENT.

Na všechny otevřené dokumenty se odkazujete pomocí **DocRef**, který je vracen příkazy Open document, Create document a Append document. DocRef jedinečně identifikuje otevřený dokument. Formálně je to časový výraz. Všechny příkazy které pracují s dokumenty přijímají DocRef jako parametr. Pokud vložíte špatné DocRef do jednoho z příkazů, vznikne chyba manažeru souborů.

Ovládání I/O chyb

Pokud pracujete s dokumenty (otevřít, zavřít, vymazat, přejmenovat, kopírovat), pokud měníte vlastnosti dokumentu nebo když čtete nebo zapisujete znaky do dokumentu, mohou nastat I/O chyby. Dokument nemusí být nalezen, může být zamčený nebo je již otevřený. Tyto chyby můžete zachytit pomocí metody instalované příkazem ON ERR CALL. Většina chyb, která může nastat při používání systémových dokumentů, je popsána v části Kódy chyb..

Systémová proměnná Document

Tři příkazy Open document, Create document a Append document vám umožní přístup k dokumentům s pomocí standardních oken Otevřít nebo Uložit soubor. Pokud použijete tato okna k otevření dokumentu, 4D vloží celou cestu k dokumentu do systémové proměnné Document. Tato systémová proměnná je odlišná od parametru dokument který se objevuje v seznamu parametrů příkazů.

Definování názvů nebo cest dokumentů

Většina rutin této části přijímá jako dokument jak název tak i cestu k dokumentu (pokud není zadáno jinak). Pokud předáte název, příkaz bude dokument hledat ve složce aplikace, a pokud předáte cestu, tak musí být správná.

Pokud předáte špatný název nebo cestu, příkaz generuje chybu, kterou můžete zachytit pomocí metody insalované ON ERR CALL.

Upozornění: Maximální délka parametru dokument je 255 znaků. Pokud předáte delší název, bude oříznut a bude generována chyba File manažeru.

Použitelné metody projektu při práci s dokumenty na disku

• Zjištění platformy na které pracujete

Ačkoli 4th Dimension obsahuje příkazy jako MAP FILE TYPES, pro eliminování rozdílností kódu podle nastavení platformy, jakmile jednou začnete pracovat na nižší úrovni při práci s dokumenty na disku (jako např. programové určení cesty), potřebujete vědět jestli jste na Windows nebo na Macintoshi.

Metoda projektu Na Windows vám řekne jestli je vaše databáze spuštěna na Windows:

```
` Metoda projektu Na windows
` Na windows   → Logické
` Na windows   → True pokud na Windows
C_BOOLEAN($0)
C_LONGINT($vlPlatform;$vlSystem;$vlMachine)
PLATFORM PROPERTIES($vlPlatform;$vlSystem;$vlMachine)
$0:=($vlPlatform=Windows)
```

• Použití správného oddělovače složek

Na Windows jsou složky v cestě odděleny opačným lomítkem (. Na Macintoshi jsou složky v cestě odděleny dvojtečkou (:). Podle toho na jaké platformě pracujete bude následující metoda měnit oddělovač složek:

```
` Metoda projektu Directory symbol
` Directory symbol → Integer
` Directory symbol → ASCII z " (Windows) nebo ":" (MacOS)
C_INTEGER($0)
If (Na windows )
   $0:=Ascii(")
Else
   $0:=Ascii(":")
End if
```

• Získání názvu souboru z celé cesty

Jakmile jednou získáte celou cestu (cesta + název souboru) k dokumentu, můžete z ní chtít získat pouze název souboru pro, například, zobrazení jeho názvu v titulu okna. Následující metoda projektu to provede jak na Windows tak na Macintoshi:

```
` Metoda projektu Dlouhý název dokumentu
` Dlouhý název dokumentu ( Řetězec ) → Řetězec
` Dlouhý název dokumentu ( Celá cesta ) → název souboru
C_STRING(255;$1;$0)
C_INTEGER($viLen;$viPos;$viChar;$viDirSymbol)
$viDirSymbol:=Directory symbol
$viLen:=Length($1)
$viPos:=0
For ($viChar;$viLen;1;-1)
   If (Ascii($1[[$viChar]])=$viDirSymbol)
      $viPos:=$viChar
      $viChar:=0
```

```
    End if
End for
If ($viPos>0)
    $0:=Substring($1;$viPos+1)
Else
    $0:=$1
End if
If (<>vbDebugOn) ` Nastavit tuto proměnnou v metodě databáze Při spuštění
    If ($0="")
        TRACE
    End if
End if
```

• **Získání cesty z celé cesty**

Jakmile jednou získáte celou cestu (cesta + název souboru) k dokumentu, můžete z ní chtít získat pouze cestu pro, například, uložení dalších dokumentů na stejné místo. Následující metoda projektu to provede jak na Windows tak na Macintoshi:

```
    ` Metoda projektu Dlouhý název cesty
    ` Dlouhý název cesty( Řetězec ) → Řetězec
    ` Dlouhý název cesty( Celá cesta ) → cesta
C_STRING(255;$1;$0)
C_STRING(1;$vsDirSymbol)
C_INTEGER($viLen;$viPos;$viChar;$viDirSymbol)
$viDirSymbol:=Directory symbol
$viLen:=Length($1)
$viPos:=0
For ($viChar;$viLen;1;-1)
    If (Ascii($1[[$viChar]])=$viDirSymbol)
        $viPos:=$viChar
        $viChar:=0
    End if
End for
If ($viPos>0)
    $0:=Substring($1;1;$viPos)
Else
    $0:=$1
End if
If (<>vbDebugOn) ` Nastavit tuto proměnnou v metodě databáze Při spuštění
    If ($0="")
        TRACE
    End if
End if
```

Dále si přečtěte

Append document, CLOSE DOCUMENT, COPY DOCUMENT, Create document, CREATE FOLDER, DELETE DOCUMENT, Document creator, DOCUMENT LIST, Document type, FOLDER LIST, Get document position, GET DOCUMENT PROPERTIES, Get document size, MAP FILE TYPES, MOVE DOCUMENT, Open document, SET DOCUMENT CREATOR, SET DOCUMENT POSITION, SET DOCUMENT PROPERTIES, SET DOCUMENT SIZE, SET DOCUMENT TYPE, Test path name, VOLUME ATTRIBUTES, VOLUME LIST.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Open document (Otevřít dokument)

Příkazy a odkazy pro Systémové dokumenty

Verze 3

Open document (dokument{; TypSouboru{; mod}}) → DocRef

Parametr	Typ		Popis
dokument	Řetězec	→	Název dokumentu nebo celá cesta k dokumentu nebo prázdný řetězec k zobrazení standardního okna otevřít soubor
TypSouboru	Řetězec	→	MacOS typ souboru (4 znakový řetězec) nebo Windows přípona souboru (1 až 3 znaky) nebo TEXT (.TXT) dokument pokud vynecháno
mod	Integer	→	Mod otevření dokumentu
Výsledek funkce	DocRef	←	Číslo odkazu na dokument

Popis

Příkaz Open document otevře dokument, jehož název nebo cestu předáte do *dokument*.

Pokud do dokument předáte prázdný řetězec (""), zobrazí se standardní okno Otevřít soubor. Pokud zrušíte toto okno nebude otevřen žádný dokument, Open document vrátí nulové DocRef a proměnnou OK nastaví na 0.

Pokud je dokument správně otevřen, Open document vrátí číslo odkazu na dokument a proměnnou OK nastaví na 1. Pokud dokument neexistuje nebo je již otevřen, je generována chyba.

Pokud na Macintoshi použijete okno Otevřít soubor, jsou jako výchozí zobrazeny všechny soubory. K zobrazení pouze určitého typu dokumentů vložte jeho typ do volitelného parametru TypSouboru.

Pokud na Windows použijete okno Otevřít soubor, jsou jako výchozí zobrazeny všechny *.* soubory. K zobrazení pouze určitého typu dokumentů vložte do volitelného parametru TypSouboru jeho příponu typu 1 až 3 znaků nebo Macintosh typ souboru s použitím převodu MAP FILE TYPES.

Pokud na Windows nepoužijete okno Otevřít soubor, můžete vložit parametr TypSouboru k definování přípony dokumentu který chcete otevřít. Jako výchozí Open document přiřadí příponu .TXT. Pokud definujete příponu, Open document se pokusí otevřít dokument jehož název je "dokument.TypSouboru". Například:

vhDokumRef:=**Open document** ("C:";"WRI")

se pokusí otevřít dokument "C:.WRI". Pokud do TypSouboru předáte více jak tři znaky, příkaz si vezme pouze první tři znaky. Pokud nepředáte tento parametr, Oped document se pokusí otevřít dokument bez přípony, pokud jej nenajde, pokusí se otevřít dokument s příponou .TXT a pokud ani ten nebude nalezen, vrátí se chyba !Soubor nenalezen".

Jakmile je dokument otevřen, Open document nastaví pozici souboru na začátek, zatímco Append document nastaví konec souboru.

Do otevřeného documentu můžete zapisovat a číst z něj pomocí příkazů RECEIVE PACKET a SEND PACKET, které můžete kombinovat s příkazy Get document position, SET DOCUMENT POSITION k zpřístupnění určité části dokumentu.

Volitelný parametr mod určuje způsob otevření dokumentu. Máte k dispozici čtyři předdefinované konstanty umístěné v tématu "System Dokumenty":

Konstanta	*Typ*	*Hodnota*

Read and Write (výchozí hodnota)	*Integer*	*0*
Write Mode	*Integer*	*1*
Read Mode	*Integer*	*2*
Get Pathname	*Integer*	*3*

Nezapomeňte provést příkaz CLOSE DOCUMENT pro zavření dokumentu.

Příklad

1. Následující příklad otevře existující dokument nazvaný Poznamka, zapíše do něj řetězec "Good bye" a uzavře jej. Pokud dokument obsahuje třeba řetězec "Ahoj", bude přepsán.

```
C_TIME(vhDokumRef)
vhDokumRef:=Open document ("Poznamka") ` Otevřít dokument Poznamka
If (OK=1)
   SEND PACKET (vhDokumRef;"Good-bye") ` Zapsat do dokumentu
   CLOSE DOCUMENT (vhDokumRef) ` Zavřít dokument
End if
```

2. Nyní můžete dokument v módu pouze číst:

```
vDoc:=Open document ("PassFile";"TEXT") ` tento soubor je otevřen
   ` Před zavřením souboru, je možné se na něj podívat v módu pouze číst:
vRef:=Open document ("PassFile";"TEXT";Read Mode)
```

Systémové proměnné a Sady

Pokud je document správně otevřen, je proměnná ok nastavena na 1, jinak je nastavena na 0. Po provedení příkazu je do systémové proměnné Document doplněna celá cesta k dokumentu. Pokud voláte Open document s modem 3, je vráceno ?00:00:00?(prázdný odkaz na dokument). Není otevřen dokument, ale jsou obnoveny proměnné Document a OK:

• OK se rovná 1

• Document obsahuje celou cestu k dokumentu určeného parametrem dokument a je závislá na předáné hodnotě (pokud předáte název dokumentu, Document bude obsahovat tento název a jestliže předáte celou cestu, bude Document obsahovat celou cestu).

Poznámka: Pokud předáte nesprávný název dokumentu nebo předáte prázdný řetězec, je otevřeno okno Otevřít soubor. Pokud uživatel vybere dokument a klepne na tlačítko OK, je document nastaven na cestu k dokumentu a proměnná OK je nastavena na 1. Pokud uživatel klepne na tlačítko Zrušit, proměnná OK je nastavena na 0.

Dále si přečtěte

Append document, Create document.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Create document (Vytvořit dokument)

Příkazy a odkazy pro Systémové dokumenty

Verze 3

Create document (dokument{; typSouboru}) → DocRef

Parametr	Typ		Popis
dokument	Řetězec	→	Název dokumentu nebo celá úplná cesta k dokumentu nebo prázdný řetězec ("") k zobrazení okna pro vytvoření souboru
TypSouboru	Řetězec	→	MacOS typ souboru (4 znaky) nebo Windows přípona (1 až 3 znaky) nebo TEXT (.TXT) dokument není-li uvedeno nic
Výsledek funkce	DocRef	←	Číslo jako odkaz na otevřený dokument

Popis

Příkaz Create document vytvoří nový dokument a vrátí číslo odkazu na dokument.

Do parametru dokument vložte název nebo celou cestu k dokumentu. Pokud dokument již existuje, je přepsán. Nicméně pokud je dokument uzamčený nebo již otevřený, je generována chyba.

Pokud předáte prázdný řetězec ("") do dokument, zobrazí se okno Uložit jako a můžete zadat název dokumentu který chcete vytvořit. Pokud okno zrušíte, není vytvořen žádný dokument, bude vrácen nulový odkaz na dokument a proměnná OK je nastavena na 0.

Pokud je dokument správně vytvořen a otevřen, Create document vrátí číslo odkazu na dokument a proměnná OK je nastavena na 1.

Ať už použijete okno Uložit jako nebo ne, jako výchozí jsou vytvářeny dokumenty .TXT (Windows) nebo TEXT (Macintosh). Pokud chcete vytvořit jiný typ dokumentu, vložte jeho typ do parametru TypSouboru.

Na Macintoshi předáte typ souboru a na Windows předáte 1 až 3 znaky přípony nebo Macintosh typ souboru převedený pomocí MAP FILE TYPES.

Jakmile je dokument vytvořen a otevřen, můžete do něj zapisovat a číst z něj pomocí příkazů RECEIVE PACKET a SEND PACKET, které můžete kombinovat s příkazy Get document position, SET DOCUMENT POSITION k zpřístupnění určité části dokumentu.

Nezapomeňte provést příkaz CLOSE DOCUMENT pro zavření dokumentu.

Příklad

Následující příklad vytvoří a otevře dokument nazvaný Poznamka a zapíše do něj řetězec "Ahoj", pak dokument uzavře:

```
C_TIME(vhDokumRef)
vhDokumRef:=Create document ("Poznamka") ` Vytvořit nový dokument
If (OK=1)
   SEND PACKET(vhDokumRef; "Ahoj") ` Zapsat jedno slovo do dokumentu
   CLOSE DOCUMENT(vhDokumRef) ` Zavřít dokument
End if
```

Dále si přečtěte

Append document, Open document.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Append document (Připojit k dokumentu)

Příkazy a odkazy pro Systémové dokumenty

Verze 3

Append document (dokument{; typSouboru}) → DocRef

Parametr	Typ		Popis
dokument	Řetězec	→	Název dokumentu nebo celá úplná cesta k dokumentu nebo prázdný řetězec ("") k zobrazení okna pro otevření souboru
TypSouboru	Řetězec	→	MacOS typ souboru (4 znaky) nebo Windows přípona (1 až 3 znaky) nebo TEXT (.TXT) dokument není-li uvedeno nic
Výsledek funkce	DocRef	←	Číslo jako odkaz na otevřený dokument

Popis

Append document provede to samé jako Open document; umožní vám otevřít dokument na disku.

Jediný rozdíl je v tom, že Append dokument nastaví pozici souboru na konec souboru a Open document nastaví pozici souboru na začátek dokumentu.

Podívejte se na popis Open document pro informace o používání Append document.

Příklad

Následující příklad otevře existující dokument Poznamka, přidá k němu " a tak dále" a Enter na konec souboru a uzavře dokument. Pokud dokument obsahoval text "Sbohem", bude nyní obsahovat "Sbohem a tak dále" následované entrem:

```
C_TIME(vhDokumRef)
vhDokumRef:=Append document ("Poznamka") ` Otevřít dokument Poznamka
SEND PACKET (vhDokumRef;" a tak dále"+Char(13)) ` Připojit řetězec
CLOSE DOCUMENT (vhDokumRef) ` Zavřít dokument
```

Dále si přečtěte

Create document, Open document.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CLOSE DOCUMENT (ZAVŘÍT DOKUMENT)

Příkazy a odkazy pro Systémové dokumenty

Verze 3

CLOSE DOCUMENT (docRef)

Parametr	Typ		Popis
docRef	DocRef	→	Číslo odkazu na dokument

Popis

CLOSE DOCUMENT zavře dokument definovaný v docRef.

Zavření dokumentu je jediný způsob jak uložit data zapsaná do dokumentu. Musíte zavřít všechny dokumenty které jste otevřeli pomocí Open document, Append document a Create document.

Příklad

Následující příklad umožní uživateli vytvořit nový dokument, zapíše do něj řetězec "Ahoj" a uzavře tento dokument:

```
C_TIME(vhDokumRef)
vhDokumRef:=Create document ("")
If (OK=1)
   SEND PACKET(vhDokumRef; "Ahoj") ` Zapsat jedno slovo do dokumentu
   CLOSE DOCUMENT(vhDokumRef) ` Zavřít dokument
End if
```

Dále si přečtěte

Append document, Create document, Open document.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

COPY DOCUMENT (KOPÍROVAT DOKUMENT)

Verze 6.0

COPY DOCUMENT (zdrojNázev; cílNázev{; *})

Parametr	Typ		Popis
zdrojNázev	Řetězec	→	Název dokumentu který se bude kopírovat
cílNázev	Řetězec	→	Název zkopírovaného dokumentu
*		→	Přepsat existující dokument, existuje-li

Popis

COPY DOCUMENT zkopíruje dokument zdrojNázev do dokumentu cílNázev.

Jak *zdrojNázev* tak *cílNázev* mohou být názvy dokumentů odkazující se do složky struktury nebo cesta odkazující se kamkoliv na disk.

Pokud nepředáte volitelný parametr *, vznikne chyba pokud již exitstuje dokument s názvem cílNázev. S volitelným parametrem COPY DOCUMENT vymaže existující dokument a nahradí jej dokumentem cílNázev.

Příklady

1. Následující příklad zkopíruje dokument do jeho vlastní složky:

```
COPY DOCUMENT("C:\SLOZKA\Nazev";" C:\SLOZKA\Nazev2")
```

(2) Následující příklad zkopíruje dokument do složky databátze (pokud C:\SLOZKA není složka databáze):

```
COPY DOCUMENT("C:\SLOZKA\Nazev";"Nazev")
```

(3) Následující příklad zkopíruje dokument z jedné složky do druhé:

```
COPY DOCUMENT("C:\SLOZKA\Nazev";"C:\ARCHIV\Nazev.OLD")
```

(4) Následující příklad zkopíruje dokument do jeho vlastní složky a připíše existující kopii:

```
COPY DOCUMENT("C:\SLOZKA\Nazev";" C:\SLOZKA\Nazev2";*)
```

Dále si přečtěte

MOVE DOCUMENT.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

MOVE DOCUMENT (PŘESUNOUT DOKUMENT)

Příkazy a odkazy pro Systémové dokumenty

Verze 6.0

MOVE DOCUMENT (zdrCesta; cílCesta)

Parametr	Typ		Popis
zdrCesta	Řetězec	→	Celá cesta k extistujícímu dokumentu
cílCesta	Řetězec	→	Cílová cesta

Popis

Příkaz MOVE DOCUMENT přesune nebo přejmenuje dokument.

Do parametru zdrCesta zadáte celou cestu k existujícímu dokumentu a do cílCesta zadáte nový název a/nebo nové umístění dokumenu.

Upozornění: S použitím MOVE DOCUMENT můžete přesunovat dokument ve složkách na jedné jednotce. Pokud chcete přesunovat dokument mezi různými jednotkami, použijte příkaz COPY DOCUMENT k "přesunutí" dokumentu a pak originál vymažte pomocí DELETE DOCUMENT.

Příklady

1. Následující příklad přejmenuje dokument Nazev:

```
MOVE DOCUMENT("C:\SLOZKA\Nazev";" C:\SLOZKA\NovyNazev ")
```

(2) Následující příklad přejmenuje a přesune dokument Nazev:

```
MOVE DOCUMENT("C:\SLOZKA1\Nazev";" C:\SLOZKA2\Nazev ")
```

(3) Následující příklad přesune dokument Nazev:

```
MOVE DOCUMENT("C:\SLOZKA1\Nazev";" C:\SLOZKA2\Nazev")
```

Poznámka: V posledních dvou příkladech musí existovat cílová složka " C:\SLOZKA2". Příkaz MOVE DOCUMENT pouze přesunuje dokument a nevytváří složky.

Dále si přečtěte

COPY DOCUMENT.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DELETE DOCUMENT (VYMAZAT DOKUMENT)

Příkazy a odkazy pro Systémové dokumenty

Verze 3

DELETE DOCUMENT (dokument)

Parametr	Typ		Popis
dokument	Řetězec	→	Název dokumentu nebo cesta k dokumentu

Popis

Příkaz DELETE DOCUMENT vymaže dokument který definujete v parametru dokument.

Pokud dokument neexistuje, není generována chyba. Ale chyba vznikne pokud se pokusíte vymazat otevřený dokument.

DELETE DOCUMENT nepříjímá prázdný řetězec. Jestliže je předán prázdný řetězec, není otevřeno okno Otevřít soubor a vznikne chyba,

UPOZORNĚNÍ: DELETE DOCUMENT může vymazat jakýkoli soubor na disku. To znamená dokumenty vytvořené jinou aplikací stejně jako aplikace samotné. Tento příkaz by měl být používán velmi opatrně. Vymazání dokumentu je trvalá operace a nelze ji vrátit.

Příklad

Následující příklad vymaže dokument s názvem "Poznamka":

```
DELETE DOCUMENT ("Poznamka") ` vymazat dokument
```

2. Podívejte se na příklad u příkazu APPEND TO CLIPBOARD.

Systémové proměnné a Sady

Vymazání dokumentu nastaví proměnnou OK na 1. Pokud příkaz nemůže vymazat dokument, je proměnná OK nastavena na 0.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Test path name (Testovat cestu)

Příkazy a odkazy pro Systémové dokumenty

Verze 6.0

Test pat hname (cesta) → Číslo

Parametr	Typ		Popis
cesta	Řetězec	→	Cesta ke složce nebo dokumentu
Výsledek funkce	Číslo	←	1, cesta odpovídá existujícímu souboru 0, cesta odpovídá existující složce <0, neplatný parametr, vrací kód chyby OS file manager

Popis

Funkce Test path name testuje jestli dokument nebo složka, jejíž cestu zadáte do parametru existuje na disku.

Pokud je dokument nalezen, Test path name vrátí 1. Jestliže je nalezena složka, výsledek funkce bude 0.

4th Dimension obsahuje následující předdefinované konstanty:

Konstanta	*Typ*	*Hodnota*
Is a document	*Long Integer*	*1*
Is a directory	*Long Integer*	*0*

Pokud není nalezena ani složka ani dokument, funkce vrátí zápornou hodnotu (např. -43 pro Soubor nenalezen).

Příklad

Následující testuje jestli dokument "Žurnál" je ve složce databáze, pak jej vytvoří jestli neexistuje:

```
If (Test path name ("Žurnál") # Is a document)
   $vhDokumRef:=Create document("Žurnál")
   If (OK=1)
      CLOSE DOCUMENT($vhDokumRef)
   End if
End if
```

Dále si přečtěte

Create document, CREATE FOLDER.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CREATE FOLDER (VYTVOŘIT SLOŽKU)

CREATE FOLDER (cestasložky)

Parametr	Typ		Popis
cestasložky	Řetězec	→	Cesta složky která má být vytvořena

Popis

Příkaz CREATE FOLDER vytvoří složku podle cesty kterou předáte do cestasložky.

Pokud předáte název, je složka vytvořena ve složce databáze. Pokud předáte cestu, tak ta musí existovat až po složku kterou chcete vytvořit; počínaje základní jednotkou až po úroveň složky databáze.

Příklady

1. Následující příklad vytvoří složku "Archiv" umístěnou ve složce databáze:

```
CREATE FOLDER ("Archiv")
```

2. Následující příklad vytvoří složku Archiv ve složce databáze a pak v ní vytvoří podsložky "Leden" a "Únor":

```
CREATE FOLDER ("Archiv")
CREATE FOLDER ("Archiv\Leden")
CREATE FOLDER ("Archiv\Únor")
```

3. Následující příklad vytvoří složku Archiv přímo na jednotce C:

```
CREATE FOLDER ("C:\Archiv")
```

4. Následující příklad nebude fungovat pokud na jednotce C není složka "NovéVlast":

```
CREATE FOLDER ("C:\NovéVlast\Obrazky") ` ŠPATNĚ, nejsou možné dvě úrovně složek v příkazu
```

Dále si přečtěte

FOLDER LIST, Test path name.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Select folder (Vybrat složku)

Příkazy a odkazy pro Systémové dokumenty

Verze 6.5

Select folder ({zpráva}) → Řetězec

Parametr	Typ		Popis
zpráva	Řetězec	→	Titulek dialogového okna
Výsledek funkce	Řetězec	←	Cesta k vybrané složce

Popis

Příkaz Select folder vám umožní označit složku a získat kompletní cestu přístupu ke složce.

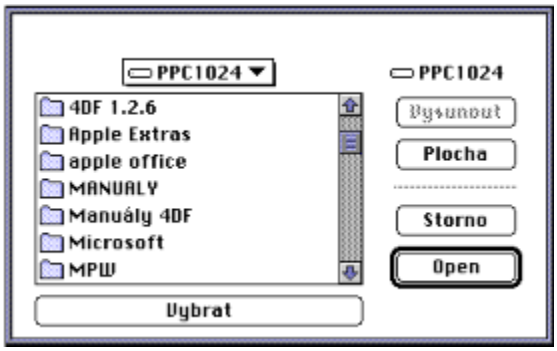
Poznámka: Tento příkaz nemění platnou složku 4D.

Příkaz Select folder zobrazí standardní dialogové okno pro prohlížení složek a jedPoznamkak počítače.

Volitelný parametr zpráva vám umožní nastavit titul tohoto dialogového okna. V následujícím příkladu je titul okna "Vybat adresář":

Windows

Macintosh

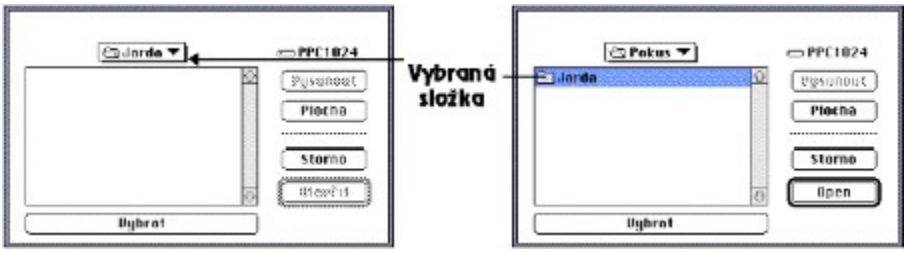


Po vybrání složky uživatel klepne na **OK** (Windows) nebo **Vybrat** (Macintosh). Cesta přístupu je pak vrácena jako výsledek funkce:

• Na Windows je cesta vrácena v následujícím formátu:

"C:\Složka1\Složka2\VybranáSložka"
• Na Macintoshi je cesta vrácena v následujícím formátu:
"Pevný Disk:Složka1:Složka2:VybranáSložka:"

Poznámka: Na Macintoshi je složka vybrána tehdy, když je označená. Pokud není, cesta může být jiná:



4D Server: Tato funkce vám umožní prohlížet jednotky připojené k počítači klienta. Tuto funkci není možné volat z uložené procedury.
Pokud uživatel potvrdí dialogové okno, je proměnná OK nastavena na 1. Pokud klepne na tlačítko **Zrušit**, proměnná OK je nastavena na 0 a výsledek funkce je prázdný řetězec.
Poznámka: Pokud na Windows uživatel vybere špatné složky jako "Stanice" nebo "Koš", je proměnná OK nastavena na 0 i když je okno správně potvrzeno.

Příklad

Následující příklad vám umožní vybrat složku do které budou uloženy obrázky z knihovny obrázků:

```
$PictFolder:=Select folder("Vyberte složku pro vaše obrázky.")
PICTURE LIBRARY LIST (pictRefs;pictNames)
For ($n;1;Size of array(pictNames))
   $vRef:=Create document($PictFolder+pictName{s$n};"PICT")
   If (OK=1)
      GET PICTURE FROM LIBRARY(pictRefs{$n};$vStoredPict)
      SAVE PICTURE TO FILE($vRef;$vStoredPict)
      CLOSE DOCUMENT($vRef)
   End if
End for
```

Dále si přečtěte

CREATE FOLDER, FOLDER LIST.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

VOLUME LIST (SEZNAM JEDNOTEK)

VOLUME LIST (jednotky)

Parametr	Typ		Popis
jednotky	Array	→	Názvy přípojených disků

Popis

Příkaz VOLUME LIST naplní Text nebo Řetězec Array názvy jedPoznamkak definovaných (Windows) nebo připojených (Macintosh) k vašemu počítači.

Na Macintoshi se do array doplní názvy jedPoznamkak viditelných z finderu.

Na druhou stránku na Windows dostanete seznam definovaných jedPoznamkak ať už jsou připojeny nebo ne (např. jednotka A bude zobrazena ať už je disketa v disketové jednotce nebo není).

Příklad

S použitím posuvné oblasti pojmenované asJednotky budete chtít zobrazit jednotky definované nebo připojené na vašem počítači:

```
Case of
   ÷ (Form event=On Load)
      ARRAY STRING(31;asJednotky;0)
      VOLUME LIST(asJednotky)
      ` ...
End case
```

Dále si přečtěte

DOCUMENT LIST, FOLDER LIST, VOLUME ATTRIBUTES.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

VOLUME ATTRIBUTES (VLASTNOSTI JEDNOTKY)

Příkazy a odkazy pro Systémové dokumenty

Verze 6.0

VOLUME ATTRIBUTES (jenotka; velikost; užito; volno)

Parametr	Typ		Popis
Jednotka	řetězec	→	název disku/ části
velikost	číslo	←	velikost diskové části v bytech
užito	číslo	←	užité místo vyjádřené v bytech
volno	číslo	←	volné místo vyjádřené v bytech

Popis

Příkaz VOLUME ATTRIBUTES vrací, vyjádřeno v bytech, velikost, použité místo a volné místo pro jednotku jejíž název předáte do parametru jednotka.

Příklad

Vaše aplikace obsahuje operace které probíhají přes noc nebo přes víkend a potřebují velké místo na disku pro dočasné soubory. Aby jste udělali tuto akci co nejvíce automatickou a přizpůsobivou, napíšete rutinu která automaticky najde jednotku s největší volnou kapacitou. Můžete napsat následující metodu projektu:

```
    ` Metoda projektu Najit jednotku s mistem
    ` Najit jednotku s mistem ( Long ) → Řetězec
    ` Najit jednotku s mistem ( Potřebné místo v bytech ) → Název jednotky nebo prázdný řetězec
C_STRING(31;$0)
C_STRING(255;$vsNazev)
C_LONGINT($1;$vlNbVolumes;$vlVolume)
C_REAL($vlVelikost;$vlUsed;$vlFree)
C_REAL($vlVelikost;$vlUsed;$vlFree)
C_TIME($vhDokumRef)
    ` Nastavit výsledek funkce
$0:=""
    ` Odchytit I/O operace proti přerušení metody
ON ERR CALL("ERROR METHOD")
    ` Získat array jedPoznamkak
ARRAY STRING(31;$asJednotky;0)
gError:=0
VOLUME LIST($asJednotky)
If (gError=0)
        ` Jestliže na Windows, přeskočit (obvykle) dvě disketové jednotky
    If (Na windows)
        $vlVolume:=Find in array($asJednotky;"A:")
        If ($vlVolume>0)
            DELETE ELEMENT($asJednotky;$vlVolume)
        End if
        $vlVolume:=Find in array($asJednotky;"B:")
        If ($vlVolume>0)
            DELETE ELEMENT($asJednotky;$vlVolume)
        End if
    End if
```

```
$vlNbVolumes:=Size of array($asJednotky)
    ` Pro každou jednotku
For ($vlVolume;1;$vlNbVolumes)
        ` Získat velikost, použité místo a volné místo
    gError:=0
    VOLUME ATTRIBUTES($asJednotky{$vlVolume};$vlVelikost;$vlUsed ;$vlFree)
    If (gError=0)
        ` Je dost volného místa (plus extra 32K) ?
        If ($vlFree>=($1+32768))
            ` Pokud ano, testovat jestli je jednotka odemčená...
            $vsNazev:=$asJednotky{$vlVolume}+Char(Directory symbol )
                    +"XYZ"+String(Random)+".TXT"
            $vhDokumRef:=Create document($vsNazev)
            If (OK=1)
                CLOSE DOCUMENT($vhDokumRef)
                DELETE DOCUMENT($vsNazev)
                    ` Pokud je všechno správně, vrátit název jednotky
                $0:=$asJednotky{$vlVolume}
                $vlVolume:=$vlNbVolumes+1
            End if
        End if
    End if
End for
End if
ON ERR CALL("")
```

Jakmile je tato metoda uložena do databáze, můžete napsat:

```
$vsVolume:=Najit jednotku s mistem (100*1024*1024)
If($vsVolume#"")
    ` Pokračovat
Else
    ALERT("Je třeba jednotka s minimálně 100 MB volného místa!")
End if
```

Dále si přečtěte

VOLUME LIST.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

FOLDER LIST (seznam složek)

Příkazy a odkazy pro Systémové dokumenty

Verze 6.0

FOLDER LIST (cesta; složky)

Parametr	**Typ**		**Popis**
cesta	Řetězec	→	Cesta k disku, části nebo složce
složky	Array	→	Názvy složek obsažené v tomto místě

Popis

Příkaz FOLDER LIST naplní text nebo řetězec array názvy složek obsažených na cestě kterou předáte do cesta.

Pokud nejsou žádné složky v tomto místě, příkaz vrátí prázdný array. Pokud je předáná cesta nesprávná, příkaz generuje chybu, kterou můžete zachytit pomocí metody ON ERR CALL.

Upozornění: Maximální délka cesty je 255 znaků. Pokud předáte delší cestu, bude oříznuta a vytvoří se chyba.

Dále si přečtěte

DOCUMENT LIST, VOLUME LIST.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DOCUMENT LIST (SEZNAM DOKUMENTŮ)

Příkazy a odkazy pro Systémové dokumenty

Verze 6.0

DOCUMENT LIST (cesta; dokumenty)

Parametr	Typ		Popis
cesta	Řetězec	→	Cesta k jednotce, části nebo složce
dokumenty	Array	→	Názvy dokumentů (souborů) obsažené v tomto místě

Popis

Příkaz DOCUMENT LIST naplní Text nebo Řetězec array názvy dokumentů obsažených v místě cesty.

Pokud na definovaném místě nejsou žádné dokumenty, příkaz vrátí prázdný array. Pokud je zadaná cesta nesprávná, DOCUMENT LIST generuje chybu, kterou můžete zachytit motodou příkazu ON ERR CALL.

Upozornění: Maximální délka cesty je 255 znaků. Pokud předáte delší cestu, bude oříznuta a vytvoří se chyba.

Dále si přečtěte

FOLDER LIST, VOLUME LIST.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Document type (Typ dokumentu)

Verze 6.0

Document type (dokument) → Řetězec

Parametr	Typ		Popis
dokument	Řetězec	→	Název dokumentu
Výsledek funkce	Řetězec	←	MacOS typ souboru (4 znaky) nebo Windows přípona (1 až 3 znaky)

Popis

Příkaz Document type vrací typ dokumentu jehož název nebo cestu předáte jako parametr.

Na Windows, Document type pro dokument vrací příponu souboru (např. `DOC` pro Microsoft Word dokument, `EXE` pro spustitelný soubor, atd.), nebo odpovídající 4 znakový MacOS typ souboru, jestliže bude typ převeden 4th Dimension na patřičnou příponu souboru Windows nebo provedení příkazu MAP FILE TYPES.

Na Macintoshi, Document type pro dokument vrací 4 znakový typ souboru (např. `TEXT` pro Textový dokument, `APPL` pro poklepatelnu aplikaci, atd.).

Dále si přečtěte

Document creator, GET DOCUMENT PROPERTIES, MAP FILE TYPES, SET DOCUMENT TYPE.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET DOCUMENT TYPE (NASTAVIT TYP DOKUMENTU)

Verze 6.0

SET DOCUMENT TYPE (document; SouborTyp)

Parametr	Typ		Popis
dokument	Řetězec	→	Název dokumentu nebo cesta k dokumentu
SouborTyp	Řetězec	→	MacOS typ souboru (4 znaky) nebo Windows přípona (1 až 3 znaky)

Popis

Příkaz SET DOCUMENT TYPE nastaví typ dokumentu jehož název nebo cestu zadáte do dokument.

Do parametru SouborTyp zadáte nový typ dokumentu.

Podívejte se na popis typů dokumentu v částech Systémové dokumenty a Document type.

Na Windows mění tento příkaz příponu souboru a proto i hodnotu souboru. Například následující instrukce:

```
SET DOCUMENT TYPE ("C:\Docs\ Faktura.asc";"TEXT")
```

přejmenuje soubor "Faktura.asc" na "Faktura.txt". Ve 4th Dimension je chápán Macintosh typ souboru "TEXT" stejný jako Windows typ ".txt".

Pokud typ nemá ekvivalent chápaný správně 4D, musíte vložit příponu. Například následující instrukce přejmenuje soubor "Faktura.asc" na "Faktura.zip":

```
SET DOCUMENT TYPE ("C:\Docs\ Faktura.asc";"zip")
```

Dále si přečtěte

Document type, MAP FILE TYPES, SET DOCUMENT CREATOR, SET DOCUMENT PROPERTIES.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

MAP FILE TYPES (PŘIŘADIT TYPY SOUBORŮ)

Verze 6.5

MAP FILE TYPES (MacOS; Windows; obsah)

Parametr	Typ		Popis
MacOS	řetězec	→	Typy souborů MacOs (4-znaky)
Windows	řetězec	→	Přípony souboru windows (1 do 3- znaky)
obsah	řetězec	→	Text ze seznamu souborů windows

Popis

MAP FILE TYPES vám umožní přiřadit Windows příponu souboru pomocí Macintosh typu souboru.

Tento příkaz potřebujete provést pouze jednou pro ustanovení převodu pro jednu pracovní stanici databáze. Po provedení tohoto příkazu budou automaticky příkazy Append document, Create document, Create resource file a Open resource file při spuštění na Windows přijímat Macintosh typy souboru zadané jako parametr.

Do parametru MacOS zadáte 4 znakový typ souboru. Pokud nepředáte 4 znakový řetězec, příkaz neprovede nic a bude generována chyba.

Da parametru windows předáte 1 až 3 znakovou příponu souboru Windows. Pokud nepředáte 1 až 3 znakový řetězec, příkaz neprovede nic a vytvoří se chyba.

Do parametru obsah předáte řetězec, který se zobrazí v rozevíracím seznamu typů souboru v okně Otevřít soubor. Řetězec obsah je omezen na 32 znaků; další znaky budou ignorovány.

DŮLEŽITÉ: Jakmile jednou zadáte převod přípony souboru Windows na Macintosh typ souboru, nemůžete tuto změnu upravit nebo vymazat na jedné stanici. Pokud potřebujete změnit "mapování" během vývoje a Ladění aplikace, znovu otevřete databázi a změňte mapování přípony souboru.

Příklad

Následující řádek kódu 4D (může být částí metody databáze Při spuštění) mapuje Macintosh typ souboru MS-Word "WDBN" na Windows příponu ".DOC":

```
MAP FILE TYPES ("WDBN";"DOC";"Dokumenty Word")
```

Jakmile je předchozí řádek proveden, zobrazí se v následujícím příkladu v okně Otevřít soubor pouze typ otevření Dokumenty Word jak na Macintoshi tak na Windows:

```
$DocRef:=Open document("";"WDBN")
If (OK=1)
   ` ...
End if
```

Dále si přečtěte

Append document, Create document, Create resource file, Open resource file, Open resource file.

Document creator (Původce dokumentu)

Verze 6.0

Document creator (dokument) → Řetězec

Parametr	Typ		Popis
dokument	Řetězec	→	Název dokumentu nebo cesta k dokumentu
Výsledek funkce	Řetězec	←	Prázdný řetězec (Windows) nebo původce souboru (Macintosh)

Popis

Příkaz Document creator vrací původce dokumentu, jehož název nebo cestu předáte do dokument.

Na Windows bude tento příkaz vracet prázdný řetězec.

Dále si přečtěte

Document type, SET DOCUMENT CREATOR.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET DOCUMENT CREATOR (NASTAVIT PŮVODCE DOKUMENTU)

Verze 6.0

SET DOCUMENT CREATOR (dokument; SouborCreator)

Parametr	Typ		Popis
dokument	Řetězec	→	Název dokumentu nebo cesta k dokumentu
SouborCreator	Řetězec	→	MacOS soubor původce (4-znaky) nebo prázdný řetězec (Windows)

Popis

Příkaz SET DOCUMENT CREATOR nastaví původce souboru, jehož název nebo cestu zadáte do paramatru dokument.

Nového původce předáte do parametru SouborCreator.

Tento příkaz na Windows nedělá nic.

Podívejte se na diskuzi o původcích souboru v části Systémové dokumenty.

Dále si přečtěte

Document creator, SET DOCUMENT PROPERTIES, SET DOCUMENT TYPE.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GET DOCUMENT PROPERTIES (ZÍSKAT VLASTNOSTI DOKUMENTU)

Příkazy a odkazy pro Systémové dokumenty

Verze 6.0

GET DOCUMENT PROPERTIES (dokument; zamčen; neviděn; vytvořenD; vytvořenČ; upravenD; upravenČ)

Parametr	Typ		Popis
dokument	řetězec	→	název dokumentu
zamčen	logické	←	uzamčen (True) nebo neuzamčen (False)
neviděn	logické	←	Neviditelný (True) nebo viditelný (False)
vytvořenD	datum	←	datum vytvoření
vytvořenČ	čas	←	čas vytvoření
upravenD	datum	←	datum poslední úpravy
upravenČ	čas	←	čas poslední úpravy

Popis

Příkaz GET DOCUMENT PROPERTIES vrací infornace o dokumentu jehož cestu nebo název předáte do parametru dokument.

Po provedení příkazu:

• *zamčen* bude obsahovat True pokud je dokument zamčen. Zamčený dokument nemůže být zamčen nebo vymazán.
• *neviděn* vrací True pokud je dokument neviditelný.
• *vytvořenD* a *vytvořenČ* bude obsahovat datum a čas kdy byl dokument vytvořen
• *upravenD* a *upravenČ* bude obsahovat datum a čas posledního upravení dokumentu.

Příklad

Vytvořili jste databázi dokumentace a chcete vyexportovat všechny záznamy z databáze do dokumentů na disku. Protože databáze je neustále obnovována chcete napsat algoritmus pro export který vytvoří nebo změní každý dokument na disku jestliže dokument neexistuje nebo jestli patřičný záznam byl změněn od posledního uložení na disku. Budete porovnávat datum a čas upravení dokumentu (pokud existuje) s datumem upravení záznamu.

Pro tento příklad použijeme následující tabulku:

Dokumenty	
Číslo	L
Věc	A20
Téma	A20
Popis	T
Značka vytvoření	L
Značka upravení	L

Spíše než ukládat datum a čas s každým záznamem, můžete uložit "časovou značku" která bude vyjadřovat rozdíl mezi jakýmkoli datumem a časem a datumem (v tomto příkladu 1. ledna 1995 v 00:00:00) a časem uložení záznamu.

V našem příkladu pole [Dokumenty]Kdy vytvoreno bude obsahovat datum a čas kdy byl záznam poprvé uložen a pole [Dokumenty]Kdy upraveno bude obsahovat datum a čas posledního upravení záznamu.

Následující metoda projektu Časová značka vypočítá rozdíl mezi specifickým datumem a časem a platným datumem a časem, pokud nejsou předány žádné parametry:

` Metoda projektu Časová značka

```
   ` Časová značka   { ( datum ; Čas ) }   → Long
   ` Časová značka   { ( datum ; Čas ) } → Počet sekund od 1 ledna 1995
C_DATE($1;$vdDatum)
C_TIME($2;$vhČas)
C_LONGINT($0)
If (Count parameters=0)
   $vdDatum:=Current date
   $vhČas:=Current time
Else
   $vdDatum:=$1
   $vhČas:=$2
End if
$0:=(($vdDatum-!01.01.95!)*86400)+$vhČas
```

Poznámka: S použitím této metody můžete vypočítat rozdíly mezi datumy 1.1.95 v 00:00:00 a 19.1.2063 v 03:14:07, což znamená rozsah long integer od 0 do 2^31 mínus 1.

Metody projektu Časová značka do datumu a Časová značka do času, získají datum a čas uložený v časové značce:

```
   ` Metoda projektu Časová značka do datumu
   ` Časová značka do datumu ( Long ) → Datum
   ` Časová značka do datumu ( Časová značka ) → Získané datum
C_DATE($0)
C_LONGINT($1)
$0:=!01.01.95!+($16400)
```

```
   ` Metoda projektu Časová značka do času
   ` Časová značka do času ( Long ) → Datum
   ` Časová značka do času ( Časová značka ) → Získaný čas
C_TIME($0)
C_LONGINT($1)
$0:=Time(Time string(†00:00:00†+($1%86400)))
```

Aby jsme měli jistotu, že budou časové značky záznamů obnovovány bez rozdílu jak byl záznam vytvořen nebu upraven, nám stačí napsat následující kód do triggeru tabulky [Dokumenty]:

```
   ` Trigger pro tabulku [Dokumenty]
Case of
   ÷ (Database event=On Saving New Record Event)
      [Dokumenty]Kdy vytvoreno:=Časová značka
      [Dokumenty]Kdy upraveno:= Časová značka
   ÷ (Database event=On Saving Existing Record Event)
      [Dokumenty]Kdy upraveno:= Časová značka
End case
```

Jakmile tyto metody zavedete do databáze, máme všechno co potřebujeme pro metodu projektu VYTVOŘIT DOKUMENTACI. Použijeme příkazy GET DOCUMENT PROPERTIES a SET DOCUMENT PROPERTIES pro práci s datumem a časem vytvoření a upravení dokumentu:

```
   ` Metoda projektu VYTVOŘIT DOKUMENTACI
C_STRING(255;$vsCesta;$vsUplnyNazev;$vsNazev)
C_LONGINT($vlDoc)
```

```
C_BOOLEAN($vbOnWindows;$vbProved;$vbUzamcen;$vbNeviditelny)
C_TIME($vhDokumRef;$vhVytvCas;$vhUprCas)
C_DATE($vdVytvDen;$vdUprDen)
If (Application type=4D Client)
      ` Pokud používáme 4D Client, uložit dokumenty
      ` uložit dokumenty na stroj kde je 4D Client spuštěn
   $vsCesta:=Dlouhý název cesty(Application file)
Else
      ` Jinak, uložit dokumenty k datovému souboru
   $vsCesta:=Dlouhý název cesty(Data file)
End if
   ` Uložit dokumenty do složky s názvem "Dokumentace"
$vsCesta:=$vsCesta+" Dokumentace"+Char(Directory symbol )
   ` Pokud tato složka neexistuje, pak ji vytvoříme
If (Test path name ($vsCesta) # Is a directory)
   CREATE FOLDER($vsCesta)
End if
   ` Vytvořit array existujících dokumentů
   ` protože budeme mazat zastaralé dokumenty, jinými slovy,
   ` dokumenty jejichž odpovídající záznamy byli vymazány.
ARRAY STRING(255;$asDocument;0)
DOCUMENT LIST($vsCesta;$asDocument)
   ` Vybrat všechny záznamy z tabulky [Dokumenty]
ALL RECORDS([Dokumenty])
   ` Pro každý záznam
$vlPocZazn:=Records in selection([Dokumenty])
$vlNbDocs:=0
$vbOnWindows:=Na windows
For ($vlDoc;1;$vlPocZazn)
      ` Předpokládáme, že budeme (znovu) vytvářet dokument na disku
   $vbProved:=True
      ` Zjistit název a cestu dokumentu
   $vsNazev:="DOC"+String([Dokumenty]Číslo;"00000")
   $vsUplnyNazev:=$vsCesta+$vsNazev
      ` Existuje tento dokument?
   If (Test path name ($vsUplnyNazev+".HTM")=Is a document)
         ` Pokud ano, odstranit dokument ze seznamu dokumentů
         ` které mohou být vymazány
      $vlElem:=Find in array($asDocument;$vsNazev+".HTM")
      If ($vlElem>0)
         DELETE ELEMENT($asDocument;$vlElem)
      End if
         ` byl dokument uložen po poslední změně záznamu?
      GET DOCUMENT PROPERTIES($vsUplnyNazev+".HTM"; $vbUzamcen;
            $vbNeviditelny;$vdVytvDen;$vhVytvCas;$vdUprDen;$vhUprCas)
      If (Časová značka ($vdUprDen;$vhUprCas)>= [Dokumenty]Kdy upraveno)
            ` Pokud ano, již nepotřebujeme znovu vytvářet dokumenty
         $vbProved:=False
      End if
   Else
         ` Dokument neexistuje, nastavit tyto dvě proměnné,
         ` protože víme že je budeme počítat před nastavením vlastností dokumentu
```

```
            $vdUprDen:=!00.00.00!
            $vhUprCas:=†00:00:00†
        End if
            ` Potřebujeme (znovu) vytvořit dokument?
        If ($vbProved)
                ` Pokud ano, zvětšit počet obnovaných dokumentů
            $vlNbDocs:=$vlNbDocs+1
                ` Vymazat dopument pokud již existuje
            DELETE DOCUMENT($vsUplnyNazev+".HTM")
                ` A znovu jej vytvořit
            If ($vbOnWindows)
                $vhDokumRef:=Create document($vsUplnyNazev;"HTM")
            Else
                $vhDokumRef:=Create document($vsUplnyNazev+".HTM")
            End if
            If (OK=1)
                    ` Zde zapsat obsah dokumentu
                CLOSE DOCUMENT($vhDokumRef)
                If ($vdUprDen=!00.00.00!)
                        ` Dokument neexistuje, nastavit datum a čas upravení
                        ` na správné hodnoty
                    $vdUprDen:=Current date
                    $vhUprCas:=Current time
                End if
                    ` Změnit vlastnosti dokumentu tak, že datum a čas vytvoření
                    ` bude stejný jako u příslušného záznamu
                SET DOCUMENT PROPERTIES($vsUplnyNazev+".HTM"; $vbUzamcen;
                $vbNeviditelny;Časová značka do datumu([Dokumenty]Kdy vytvoreno);
                Časová značka do času ([Dokumenty]Kdy vytvoreno);$vdUprDen;$vhUprCas)
            End if
        End if
            ` Je třeba vědět co se provádí
        SET WINDOW TITLE("Provádí se dokument "+String($vlDoc)+ " z   +String($vlPocZazn))
        NEXT RECORD([Dokumenty])
    End for
        ` Vymazat staré dokumenty, jinými slovy
        ` ty kteřé jsou v array $asDocument
    For ($vlDoc;1;Size of array($asDocument))
        DELETE DOCUMENT($vsCesta+$asDocument{$vlDoc})
        SET WINDOW TITLE("Mazání starých záznamů:" +Char(34)+$asDocument{$vlDoc}+Char(34))
    End for
        ` Jsme hotovi
    ALERT("Počet provedených dokumentů: "+String($vlPocZazn)+Char(13) +
                "Počet obnovených dokumentů: "+String($vlNbDocs)+Char(13)+
                "Počet vymazaných dokumentů: "+String(Size of array($asDocument)))
```

Dále si přečtěte

Document creator, Document type, SET DOCUMENT PROPERTIES.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET DOCUMENT PROPERTIES (NASTAVIT VLASTNOSTI DOKUMENTU)

Verze 6.0

SET DOCUMENT PROPERTIES (dokument; zamčen; neviděn; vytvořenD; vytvořenČ; upravenD; upravenČ)

Parametr	Typ		Popis
dokument	řetězec	→	název dokumentu
zamčen	logické	→	uzamčen (True) nebo neuzamčen (False)
neviděn	logické	→	Neviditelný (True) nebo viditelný (False)
vytvořenD	datum	→	datum vytvoření
vytvořenČ	čas	→	čas vytvoření
upravenD	datum	→	datum poslední úpravy
upravenČ	čas	→	čas poslední úpravy

Popis

Příkaz SET DOCUMENT PROPERTIES mění informace o dokumentu jehož název cestu jste zadali do parametru dokument.

Před provedením příkazu:

• Do parametru *zamčen* vložte True pro zamknutí dokumentu. Zamčený dokument nemůže být otevřen nebo vymazán. Vložte False pro odemčení dokumentu.
• Do parametru *neviděn* vložte True pro skrytí dokumentu. Vložte False pro zviditelnění dokumentu v okně.
• Datum a čas vytvoření dokumentu vložte do parametrů *vytvořenD* a *vytvořenČ*
• Datum a čas posledního upravení dokumentu vložte do *upravenD* a *upravenČ*

Datumy a časy vytvoření a upravení hlídá systém a upravuje je při vytvoření nebo upravení dokumentu. S použitím tohoto příkazu můžete změnit tyto údaje pro specielní účely. Podívejte se na příklad u příkazu GET DOCUMENT PROPERTIES.

Dále si přečtěte

GET DOCUMENT PROPERTIES, SET DOCUMENT CREATOR, SET DOCUMENT TYPE.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Get document size (Získat velikost dokumentu)

Příkazy a odkazy pro Systémové dokumenty

Verze 6.0

Get document size (dokument{; *}) → Číslo

Parametr	Typ		Popis
dokument	DocRef | Řetězec	→	Číslo odkazu na dokument nebo název dokumentu
*		→	Pouze na MacOS: - pokud vynecháno velikost datové části - pokud definováno velikost zdrojové části
Výsledek funkce	Číslo	←	Velikost (vyjádřená v bytech) dokumentu

Popis

Příkaz Get document size vrací velikost dokumentu, vyjádřenou v bytech.

Pokud je dokument otevřen, předáte číslo odkazu na dokument. Pokud dokument není otevřen, předáte název nebo cestu k dokumentu.

Na Macintoshi, pokud předáte volitelný parametr *, bude vrácena velikost datové části. Pokud tento parametr vynecháte, bude vrácena velikost zdrojové části.

Dále si přečtěte

Get document position, SET DOCUMENT POSITION, SET DOCUMENT SIZE.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET DOCUMENT SIZE (NASTAVIT VELIKOST DOKUMENTU)

Verze 6.0

SET DOCUMENT SIZE (dokument; velikost)

Parametr	Typ		Popis
dokument	DocRef	→	Číslo odkazu na dokument
velikost	Číslo	→	Nová velikost vyjádřená v bytech

Popis

Příkaz SET DOCUMENT SIZE nastaví velikost dokumentu vyjádřenou v bytech.

Pokud je dokument otevřen, předáte do parametru dokument číslo odkazu na dokument.

Na Macintoshi je změněna velikost datové části.

Dále si přečtěte

Get document position, Get document size, SET DOCUMENT POSITION.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Get document position (Získat pozici v dokumentu)

Verze 6.0

Get document position (docRef) → Číslo

Parametr	Typ		Popis
docRef	DocRef	→	Číslo odkazu na dokument
Výsledek funkce	Číslo	←	Pozice souboru (vyjádřený v bytech) od začátku do konce

Popis

Tento příkaz pracuje pouze s otevřenými dokumenty jejichž číslo odkazu předáte pomocí docRef.

Get document position vrací pozici, začne na začátku dokumentu, odkud bude další čtení (RECEIVE PACKET) nebo zapisování (SEND PACKET).

Dále si přečtěte

RECEIVE PACKET, SEND PACKET, SET DOCUMENT POSITION.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET DOCUMENT POSITION (NASTAVIT POZICI V DOKUMENTU)

Verze 6.0

SET DOCUMENT POSITION (docRef; posun; ukotvení)

Parametr	Typ		Popis
docRef	DocRef	→	Číslo odkazu na dokument
posun	Číslo	→	Pozice v dokumentu (v bytech) od počátku souboru
ukotvení	Číslo	→	1 = Relativně k počátku souboru 2 = Relativně ke konci souboru 3 = Relativně k současné pozici

Popis

Tento příkaz pracuje pouze s otevřenými dokumenty, jejichž číslo odkazu předáte do parametru *docRef*.

SET DOCUMENT POSITION nastaví pozici předánou do posun odkud se bude číst (RECEIVE PACKET) nebo kam se bude zapisovat (SEND PACKET).

Pokud vynecháte volitelný parametr ukotvení, pozice je relativně od počátku dokumentu. Pokud předáte tento parametr, použijete jednu ze zobrazených hodnot.

Podle hodnoty *ukotvení* můžete vložit kladný nebo záporný posun.

Dále si přečtěte

Get document position, RECEIVE PACKET, SEND PACKET.

Příkazy a odkazy pro Systémové dokumenty

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Screen height (Výška obrazovky)

Verze 3.5

Screen height ({*}) → Číslo

Parametr	Typ		Popis
*	Číslo	→	Windows: výška okna aplikace nebo pokud definováno *výška obrazovky Macintosh: výška hlavní obrazovky
Výsledek funkce	Číslo	←	Výška vyjádřená v bodech

Popis

Na Windows: Příkaz vrací výšku okna aplikace 4D. Pokud definujete parametr *, vrátí příkaz výšku obrazovky.

Na Macintoshi: Příkaz vrací výšku obrazovky, kde je umístěno záhlaví nabídek.

Dále si přečtěte

SCREEN COORDINATES, Screen width.

Příkazy a odkazy pro Prostředí systému

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Screen width (Šířka obrazovky)

Verze 3.**5**

Screen width ({*}) → Číslo

Parametr	Typ		Popis
*	Číslo	→	Windows: šířka okna aplikace nebo pokud definováno * šířka obrazovky Macintosh: šířka hlavní obrazovky
Výsledek funkce	Číslo	←	Šířka vyjádřená v bodech

Popis

Na Windows: Příkaz vrací šířku okna aplikace 4D. Pokud definujete parametr *, vrátí příkaz šířku obrazovky.

Na Macintoshi: Příkaz vrací šířku obrazovky, kde je umístěno záhlaví nabídek.

Dále si přečtěte

SCREEN COORDINATES, Screen height.

Příkazy a odkazy pro Prostředí systému

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Count screens (Počet obrazovek)

Příkazy a odkazy pro Prostředí systému

Verze 6.0

Count screens → Číslo

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Číslo	←	Počet monitorů

Popis

Příkazt Count screens vrací počet monitorů připojených k vašemu počítači.

Poznámka pro Windows: Na Windows obvykle příkaz vrací 1.

Dále si přečtěte

Menu bar screen, SCREEN COORDINATES, SCREEN DEPTH, Screen height, Screen width.

Příkazy a odkazy pro Prostředí systému

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SCREEN COORDINATES (SOUŘADNICE OBRAZOVKY)

SCREEN COORDINATES (levá; vrch; pravá; spodní{; obrazovka})

Parametr	Typ		Popis
levá	číslo	←	levá souřadnice okna v okně aplikace
vrch	číslo	←	vrchní souřadnice okna v okně aplikace
pravá	číslo	←	pravá souřadnice okna v okně aplikace
spodní	číslo	←	spodní souřadnice okna v okně aplikace
obrazovka	číslo	→	číslo obrazovky, nebo hlavní obrazovka je-li vynecháno

Popis

Příkaz SCREEN COORDINATES vrací souřadnice obrazovky *levá, pravá, spodní* a *vrchní* pro obrazovku *obrazovka*.

Na Windows

Obvykle nebudete používat parametr *obrazovka*.

Na Macintoshi

Pokud vynecháte parametr *obrazovka*, příkaz vrátí souřednice hlavní obrazovky, obrazovky kde je záhlaví nabídek.

Dále si přečtěte

Count screens, Menu bar screen, SCREEN DEPTH.

Příkazy a odkazy pro Prostředí systému

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SCREEN DEPTH (HLOUBKA OBRAZOVKY)

Příkazy a odkazy pro Prostředí systému

Verze 6.0

SCREEN DEPTH (hloubka; barva{; obrazovka})

Parametr	Typ		Popis
hloubka	Číslo	←	hloubka, počet barev obrazovky (počet barev = (2)^ hloubka)
barva	Číslo	←	1 = barevná, 0 = černobílá nebo stupně šedi
obrazovka	Číslo	→	číslo obrazovky, nebo je-li vynecháno hlavní obrazovka

Popis

Příkaz Screed depth vrací hloubku a barvy obrazovky.

Po provedení příkazu:

• Hloubka obrazovky je vrácena do parametru hloubka. Hloubka obrazovky je exponent pro mocninu 2 vyjadřující počet barev zobrazených na monitoru. Pokud je například váš monitor nastaven na 256 barev (2^8), hloubka obrazovky bude 8.

Následující předdefinované konstanty jsou obsaženy ve 4th Dimension:

Konstanty	***Typ***	***Hodnota***
Black and white	*Long Integer*	*0*
Four colors	*Long Integer*	*2*
Sixteen colors	*Long Integer*	*4*
Two fifty six colors	*Long Integer*	*8*
Thousands of colors	*Long Integer*	*16*
Millions of colors 24 bit	*Long Integer*	*24*
Millions of colors 32 bit	*Long Integer*	*32*

Pokud je obrazovka nastavena na zobrazení barev, vrátí se do parametru barva hodnota 1. Pokud je obrazovka nastavena pouze na odstíny šedé, je parametr nastaven na 0. Poznamenejte si, že tato hodnota je významná na Macintoshi.

Ve 4D jsou obsaženy následující konstanty:

Konstanta	***Typ***	***Hodnota***
Is gray scale	*Long Integer*	*0*
Is color	*LongInteger*	*1*

• Volitelný parametr obrazovka definuje pro kterou obrazovku chcete získat informace. Na Windows tento parametr obvykle používat nebudete. Pokud na Macintoshi vynecháte tento parametr, příkaz vrátí informace o hlavní obrazovce, obrazovka kde je záhlaví nabídek.

Příklad

Vaše aplikace zobrazuje mnoho barevných obrázků. Někam do databáze můžete zařadit následující kód:

```
SCREEN DEPTH ($vlDepth;$vlColor)
```

```
If ($vlDepth<8)
   ALERT("Formuláře budou vypadat lépe pokud budou zobrazeny v 256 a více barvách")
End if
```

Dále si přečtěte

Count screens, SET SCREEN DEPTH.

Příkazy a odkazy pro Prostředí systému

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET SCREEN DEPTH (NASTAVIT HLOUBKU OBRAZOVKY)

Příkazy a odkazy pro Prostředí systému

Verze 6.0

SET SCREEN DEPTH (hloubka{; barva{; obrazovka}})

Parametr	Typ		Popis
hloubka	Číslo	→	hloubka, počet barev obrazovky (počet barev = (2)^ hloubka)
barva	Číslo	→	1 = barevná, 0 = černobílá nebo stupně šedi
obrazovka	Číslo	→	číslo obrazovky, nebo je-li vynecháno hlavní obrazovka

Popis

Tento příkaz neprovede nic na Windows.

Na Macintoshi, SET SCREEN DEPTH mění hloubku a barené/stupně šedi nastavení obrazovky jejíž číslo předáte do parametru obrazovka. Pokud tento parametr vynecháte, je příkaz použit na hlavní obrazovku.

Podrobné informace o hodnotách, které se mají vložit do parametrů hloubka a barva si přečtěte popis příkazu SCREEN DEPTH.

Dále si přečtěte

SCREEN DEPTH.

Příkazy a odkazy pro Prostředí systému

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Menu bar screen (Obrazovka záhlaví nabídek)

Verze 6.0

Menu bar screen → Číslo

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Číslo	←	Číslo obrazovky kde jsou umístěny nabídky

Popis

Příkaz Menu bar screen vrací číslo obrazovky kde jsou umístěny nabídky.

Poznámka pro Windows: Na Windows bude tento příkaz většinou vracet 1.

Dále si přečtěte

Count screens, Menu bar height.

Příkazy a odkazy pro Prostředí systému

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Menu bar height (Výška záhlaví nabídek)

Příkazy a odkazy pro Prostředí systému

Verze 6.0

Menu bar height → Číslo

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Číslo	←	Výška záhlaví nabídky (v bodech) (vrátí 0 pro uschované záhlaví)

Popis

Menu bar height vrací výšku záhlaví nabídky vyjádřenou v bodech.

Dále si přečtěte

HIDE MENU BAR, Menu bar screen, SHOW MENU BAR.

Příkazy a odkazy pro Prostředí systému

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

FONT LIST (SEZNAM PÍSEM)

Příkazy a odkazy pro Prostředí systému

Verze 6.0

FONT LIST (písma)

Parametr	Typ		Popis
písma	Array	←	Array názvů písem

Popis

Příkaz FONT LIST naplní řetězec nebo text array názvy písem dostupných ve vašem systému.

Příklad

Ve formuláři potřebujete vytvořit rozevírací nabídku obsahující písma obsažená ve vašem systému. Metoda rozevíracího seznamu je zobrazená zde:

```
Case of
   ÷ (Form event=On Load)
      ARRAY STRING(63;asFont;0)
      FONT LIST(asFont)
   ` ...
End case
```

Dále si přečtěte

Font name, Font number.

Příkazy a odkazy pro Prostředí systému

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Font name (Název písma)

Příkazy a odkazy pro Prostředí systému

Verze 6.0

Font name (ČísloPísma) → Řetězec

Parametr	Typ		Popis
ČísloPísma	Číslo	→	Číslo písma pro které chceme název
Výsledek funkce	Řetězec	←	Název písma

Popis

Font name vrací název písma jehož číslo zadáte do jako parametr. Pokud není žádné písmo s takovým číslem, příkaz vrátí prázdný řetězec.

Příklady

1. K zobrazení objektu formuláře s výchozím písmem systému, napište:

    FONT (můjObj; **Font name** (0)) ` 0 je číslo výchozího systémového písma

2. K zobrazení objektu formuláře se zobrazeným výchozím písmem aplikace, napište:

    FONT (můjObj;**Font name**(1)) ` 1 je číslo výchozího písma aplikace

Dále si přečtěte

FONT LIST, Font number.

Příkazy a odkazy pro Prostředí systému

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Font number (Číslo písma)

Příkazy a odkazy pro Prostředí systému

Verze 6.0

Font number (NázevPísma) → Číslo

Parametr	Typ		Popis
NázevPísma	Řetězec	→	Název písma pro které chceme číslo
Výsledek funkce	Číslo	←	Číslo písma

Popis

Příkaz Font number vrací číslo písma jehož název zadáte jako parametr. Pokud není žádné písmo se zadaným názvem, příkaz vrátí 0.

Příklad

Některé fomruláře ve vaší databázi používají písmo "Specialní Typ". Můžete použít následující kód:

```
If (Font number("Specialní Typ")=0)
   ALERT("Tento formulář bude vypadat lépe, pokud nainstalujete písmo Specialní Typ.")
End if
```

Dále si přečtěte

FONT LIST, Font name.

Příkazy a odkazy pro Prostředí systému

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

System folder (Systémová složka)

Verze 6.0

System folder → Řetězec

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Řetězec	←	Cesta k aktivní složce systému

Popis

Příkaz System folder vrací cestu k aktivní složce systému Windows nebo Macintosh

Dále si přečtěte

ACI folder, Temporary folder.

Příkazy a odkazy pro Prostředí systému

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Temporary folder (Dočasná složka)

Verze 6.0

Temporary folder → Řetězec

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Řetězec	←	Cesta k dočasné složce

Popis

Příkaz Temporary folder vrací cestu k dočasné složce nastavené vaším systémem.

Příklad

Podívejte se na příklad u příkazu APPEND TO CLIPBOARD.

Dále si přečtěte

System folder.

Příkazy a odkazy pro Prostředí systému

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Current machine (Platný stroj)

Příkazy a odkazy pro Prostředí systému

Verze 6.0

Current machine → Řetězec

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Řetězec	←	Síťový název stroje

Popis

Příkaz Current machine vrací síťový název počítače, jak je nastaveno v předvolbách sítě.

Příklad

I pokud nepoužíváte klient/server verzi 4th Dimension, může vaše aplikace provádět různé síťové úkoly, při kterých potřebuje správné nastavení počítače. Do metody databáze Při spuštění vložte následující:

```
If ( (Current machine="") | (Current machine owner=""))
   ` Zobrazí dialogová okna pro nastavení
   ` síťových názvů vašeho počítače.
End if
```

Dále si přečtěte

Current machine owner.

Příkazy a odkazy pro Prostředí systému

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Current machine owner (Platný vlastník stroje)

Příkazy a odkazy pro Prostředí systému

Verze 6.0

Current machine owner → Řetězec

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Řetězec	←	Síťové jméno vlastníka stroje

Popis

Příkaz Current machine owner vrací jméno vlastníka stroje, jak je zadán ve vlastnostech sítě.

Příklad

Podívejte se na příklad u příkazu Current machine.

Dále si přečtěte

Current machine.

Příkazy a odkazy pro Prostředí systému

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Gestalt

Příkazy a odkazy pro Prostředí systému

Verze 6.0

Gestalt (dotaz; hodnota) → Číslo

Parametr	Typ		Popis
dotaz	řetězec	→	4-znakový dotaz systémové službě gestalt
hodnota	číslo	←	výsledná odpověď gestalt (především MacOs)
Výsledek funkce	číslo	←	výsledný kód chyby

Popis

Příkaz Gestalt vrací do hodnota číselnou hodnotu která určuje charakteristiku vašeho hardwaru a softwaru, podle dotazu který předáte do parametru dotaz.

Pokud je vyžadovaná informace vynechána, Gestalt vrátí 0 jako výsledek funkce; jinak vrací chybu -5550. Jestliže je dotaz neznámý, Gestalt vrátí chybu -5551.

Důležité: Gestalt Manager je součástí MacOS. Na Windows jsou předány některé části, ale jejich použitelnost je omezené.

Jestli chcete vědět více informací o dotazech které můžete vložit, podívejte se do příslušné části v Apple Developer documentation.

Příklad

Na Macintoshi, ve verzi MacOS 7.6, zobrazí následující kód hlášení "Používáte systém verze 0x0760":

```
$vlKodChyby:=Gestalt("sysv";$vlInfo)
If ($vlKodChyby=0)
   ALERT("Používáte systém verze "+String($vlInfo;"&x"))
End if
```

Příkazy a odkazy pro Prostředí systému

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

LOG EVENT (LOG UDÁLOST)

Verze 6.5

LOG EVENT (zpráva{; důležitost})

Parametr	Typ		Popis
zpráva	řetězec	→	Obsah zprávy pro WinNT LogEvents
důležitost	Integer	→	úroveň důležitosti zprávy

Popis

Příkaz LOG EVENT vám umožní vložit vlastní zprávu která se bude zobrazovat ve Windows NT **Log events**. Tato služba ovládá log soubor který udržuje zprávy přicházející z běžících aplikací. Tímto vám umožní sledovat průběh práce. Jestli chcete vědět více informací, přečtěte si *Příručku návrháře.*

Poznámka: Aby byla tato možnost přístupná, musí být služba Windows NT **Log events** spuštěna.

Zprávu k zobrazení v log events vložte do parametru zpráva.

Můžete určit i důležitost zprávy, což vám usnadní čtení a porozumění log událostem. Jsou tři úrovně důležitosti: Informace, Upozornění a Chyba. Parametr důležitost vám umožní nastavit úroveň důležitosti zprávy.

4th Dimension obsahuje následující předdefinované konstanty k určení důležitosti, umístěné v části "Windows NT Log events":

Konstanta	***Typ***	***Hodnota***
Information Message	*Integer*	*0*
Warning Message	*Integer*	*1*
Error Message	*Integer*	*2*

Pokud do důležitost nepředáte nic, nebo předáte nsprávnou hodnotu, je použita výchozí hodnota (0).

Příklad

Pokud chcete mít přehled tom, kdy byla vaše databáze otevřena, můžete do metody databáze Při spuštění vložit tento řádek:

```
LOG EVENT ("Databáze Faktury byla otevřena.")
```

Pokaždé když bude databáze otevřena, tato informece se zapíše do Windows NT log events a její úroveň důležitosti bude 0.

Příkazy a odkazy pro Prostředí systému

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DEFAULT TABLE (VÝCHOZÍ TABULKA)

Příkazy a odkazy pro Tabulky

Verze 3

DEFAULT TABLE (tabulka)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka, kterou nastavit jako výchozí

Popis

DEFAULT TABLE nastaví tabulka jako výchozí tabulku platného procesu.

Pro proces není určena výchozí tabulka, dokud není proveden příkaz DEFAULT TABLE. Po nastavení výchozí tabulky budou všechny příkazy u kterých vynecháte parametr tabulka používat tuto výchozí tabulku. Například tento příkaz:

```
INPUT FORM ([Tabulka]; "formulář")
```

Pokud bude nastavena jako výchozí tabulka [Tabulka], provede následující řádek stejnou akci:

```
INPUT FORM ("formulář")
```

Hlavní důvod pro nastavení výchozí tabulky, je možnost napsat metodu nezávislou na tabulce. To vám umožní použít jednu metodu pro více tabulek. K napsání kódu nezávislého na tabulce můžete také použít ukazatele. Více informací o tomto způsobu najdete v popisu příkazu Table name.

DEFAULT TABLE vám neumožní vynechat názvy tabulek při odkazování na pole. Například:

```
[Tabulka]Moje Pole := "Řetězec"    ` SPRÁVNĚ
```

nemůže být napsáno, protože byla nastavena výchozí tabulka, takto:

```
DEFAULT TABLE ([Tabulka])
Moje Pole := "Řetězec"    ` ŠPATNĚ
```

Nicméně názvy tabulek můžete vynechat při odkazování na pole v metodě tabulky, formuláře a objektu, který patří k tabulce.

Ve 4th Dimension jsou všechny tabulky "otevřené" a připravené pro použití. DEFAULT TABLE neotevře tabulku, nastaví výchozí tabulku nebo připraví tabulku pro vstup nebo výstup. DEFAULT TABLE je způsob jak zjednodušit programovací zápis, omezit množství psaní a udělat kód přehlednější.

Tip: Ačkoli použití DEFAULT TABLE a vynechání názvů tabulek může udělat kód čitelnější, mnoho programátorů zjistilo, že použití tohoto příkazu způsobuje více prohlémů a zmatku než přináší užitku.

Příklad

Následující příklad nejdříve ukáže kód bez použití DEFAULT TABLE. Pak ukáže ten samý kód, ale s použitím DEFAULT TABLE. Příklad je smyčka používaná k přidání záznamů do databáze. Příkazy INPUT FORM a ADD RECORD vyžadují jako první parametr tabulku:

```
INPUT FORM ([Zákazníci];"Add Recs")
Repeat
   ADD RECORD ([Zákazníci])
Until (OK = 0)
```

Definování výchozí tabulky v kódu:

```
DEFAULT TABLE ([Zákazníci])
INPUT FORM ("Add Recs")
Repeat
   ADD RECORD
Until (OK = 0)
```

Dále si přečtěte

Current default table.

Příkazy a odkazy pro Tabulky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Current default table (Platná výchozí tabulka)

Příkazy a odkazy pro Tabulky

Verze 3

Current default table → Ukazatel

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Ukazatel	←	Ukazatel na výchozí tabulce

Popis

Current default table vrací ukazatel na výchozí tabulku, která byla nastavena příkazem DEFAULT TABLE.

Příklad

Za předpokladu že byla nastavena výchozí tabulka, nastaví následující řádek název platné tabulky jako titul okna:

```
SET WINDOW TITLE(Table name(Current default table))
```

Dále si přečtěte

DEFAULT TABLE, Table, Table name.

Příkazy a odkazy pro Tabulky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

INPUT FORM (VSTUPNÍ FORMULÁŘ)

Příkazy a odkazy pro Tabulky

Verze 6.0 (upraveno)

INPUT FORM ({tabulka; }{formulář{; *})

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ke které patří vstupní formulář nebo pokud vynecháno, použít výchozí tabulku
formulář	Řetězec	→	Název vstupního formuláře
*		→	Automatická velikost okna

Popis

Příkaz INPUT FORM nastaví platný vstupní formulář pro tabulku. Formulář musí patřit k tabulce.

Rozsah tohoto příkazu je platný proces. Každá tabulka má svůj vlastní vstupní formulář pro každý proces.

INPUT FORM nezobrazí formulář; pouze určí. který formulář bude pro vstup dat, import nebo akce jiných příkazů. Informace o vytváření formulářů najdete v *Přiručce návrháře 4th Dimension.*

Výchozí vstupní formulář pro každou tabulku je definován v Průzkumníku v prostředí návrháře. Tento formulář bude použit pokud nebude použit příkaz INPUT FORM pro nastavení jiného formuláře, nebo pokud nastavíte neplatný formulář.

Vstupní formulář je použit mnoha příkazy, které umožňují zadání nových dat nebo upravení starých dat. Následující příkazy zobrazí vstupní formulář pro vstup dat nebo účely dotazu:

- ADD RECORD
- DISPLAY RECORD
- MODIFY RECORD
- QUERY BY EXAMPLE

Příkazy DISPLAY SELECTION a MODIFY SELECTION zobrazí seznam záznamů s použitím výstupního formuláře. Uživatel může poklepat na záznam a zobrazit jej ve vstupním formuláři.

Příkazy importu IMPORT TEXT, IMPORT SYLK a IMPORT DIF používají k importování záznamů platný vstupní formulář.

Volitelný parametr * je použit v porovnání s nastavením vlastností formuláře v prostředí návrháře a příkazu Open window. Definování tohoto parametru řekne 4D aby použila nastavení formuláře a změnila velikost okna ve kterém se formulář zobrazí podle tohoto nastavení. Podívejte se na Open window pro více informací.

Poznámka: Ať už použijete parametr *, INPUT FORM změní vstupní formulář pro tabulku.

Příklad

Následující příklad je typické použití INPUT FORM:

```
INPUT FORM ([Companies]; "New Comp") ` Formulář pro přidání firem
ADD RECORD ([Companies]) ` Přidat novou firmu
```

Dále si přečtěte

ADD RECORD, DISPLAY RECORD, DISPLAY SELECTION, IMPORT DIF, IMPORT SYLK, IMPORT TEXT, MODIFY RECORD, MODIFY SELECTION, Open window, OUTPUT FORM, QUERY BY EXAMPLE.

Příkazy a odkazy pro Tabulky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

OUTPUT FORM (VÝSTUPNÍ FORMULÁŘ)

Verze 3

OUTPUT FORM ({tabulka ;}formulář)

Parametr	Typ		Popis
tabulka	Tabulka	→	Tabulka ke které patří vstupní formulář nebo pokud vynecháno, použít výchozí tabulku
formulář	Řetězec	→	Název výstupního formuláře

Popis

Příkaz OUTPUT FORM nastaví platný výstupní formulář pro tabulku. Formulář musí patřit k tabulce.

Rozsah tohoto příkazu je platný proces. Každá tabulka má svůj vlastní výstupní formulář pro každý proces.

OUTPUT FORM nezobrazí formulář; pouze určí který formulář bude tištěn, zobrazen nebo použit jinými příkazy. Informace o vytváření formulářů najdete v *Přiručce návrháře 4th Dimension.*

Výchozí výstupní formulář pro každou tabulku je definován v Průzkumníku v prostředí návrháře. Tento formulář bude použit pokud nebude použit příkaz OUTPUT FORM pro nastavení jiného formuláře, nebo pokud nastavíte neplatný formulář.

Výstupní formulář je použit třemi skupinymi příkazů. Jedna skupina zobrazuje seznam záznamů na obrazovku, další skupiny vytváří zprávy a další exportuje data. Příkazy DISPLAY SELECTION a MODIFY SELECTION zobrazují seznam záznamů na obrazovku s použitím výstupního formuláře. Výstupní formulář používáte pro vytváření zpráv pomocí příkazů PRINT LABEL a PRINT SELECTION. Každý z exportních příkazů (EXPORT DIF, EXPORT SYLK a EXPORT TEXT) také používají výstupní formulář.

Příklad

Následující příklad ukazuje typické použití příkazu OUTPUT FORM. Ačkoli se OUTPUT FORM používá před použitím výstupního formuláře, není to nutné. Tento příkaz může být použit ve kterékoli metodě, pokud je tato spuštěna před touto metodou:

```
INPUT FORM ([Zboží]; "Parts In") ` Vybrat vstupní formulář
OUTPUT FORM ([Zboží]; "Parts List") ` vybrat výstupní formulář
MODIFY SELECTION ([Zboží]) ` Tento příkaz používá oba formuláře
```

Dále si přečtěte

DISPLAY SELECTION, EXPORT DIF, EXPORT SYLK, EXPORT TEXT, INPUT FORM, MODIFY SELECTION, PRINT LABEL, PRINT SELECTION.

Current form table (Tabulka platného formuláře)

Příkazy a odkazy pro Tabulky

Verze 6.0

Current form table → Ukazatel

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Ukazatel	←	Ukazatel na tabulce platného zobrazeného formuláře

Popis

Příkaz Current form table vrátí ukazatel na tabulce, ke které patří platně zobrazený nebo tištěný formulář v platném procesu.

Pokud není žádný zobrazený nebo tištěný formulář, příkaz nevrátí nic.

Pokud pro platný formulář je otevřeno více oken s formuláři, je použito aktivní okno.

Pokud platně zobrazený formulář je formulář obsahu pro oblast podformuláře, jste ve vstupu dat a poklepete na záznam nebo podzáznam v poklepatelné oblasti podformuláře. V tomto případě příkaz vrátí:

- Ukazatel na tabulce zobrazené v oblasti podformuláře, pokud podformulář zobrazuje tabulku.
- Nevýznamný ukazatel, pokud oblast podformuláře zobrazuje podtabulku.

Příklad

Ve vašich aplikacích použijete následující konvence pro zobrazení záznamů:

Pokud je ve formuláři proměnná vsTentoZaznam, zobrazí "Nový záznam" pokud pracujete s novým záznamem. Pokud pracujete s 56-tým záznamem z 5200 záznamů, zobrazí "56 z 5200".

K tomu použijte metodu objektu a vytvořte proměnnou vsTentoZaznam a pak ji zkopírujte do všech formulářů:

```
      ` Metoda objektu vsTentoZaznam nedostupná proměnná
Case of
   ÷ (Form event =On Load)
      C_STRING (31;vsTentoZaznam)
      C_POINTER ($vpMaterskaTab)
      C_LONGINT ($vlZaznamNum)
      $vpMaterskaTab:=Current form table
      $vlZaznamNum:=Record number ($vpMaterskaTab→)
      Case of
         ÷ ($vlZaznamNum=-3)
            vsTentoZaznam:="Nový záznam"
         ÷ ($vlZaznamNum=-1)
            vsTentoZaznam:="Žádný záznam"
         ÷ ($vlZaznamNum>=0)
            vsTentoZaznam:=String (Selected record number ($vpMaterskaTab→))+
            " z "+String (Records in selection ($vpMaterskaTab→))
      End case
End case
```

Dále si přečtěte

DIALOG, INPUT FORM, OUTPUT FORM, PRINT SELECTION.

Příkazy a odkazy pro Tabulky

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Použití Transakcí

Příkazy a odkazy pro Transakce

Verze 6.0

Transakce je série následných úprav dat provedených v procesu. Transakce není do databáze uložena trvale, dokud není potrzena. Pokud transakce není dokončena, ať už proto že je zrušena nebo kvůli nějaké venkovní události, úpravy nejsou uloženy.

Všechny změny provedené do databáze během transakce jsou ukládány do dočasného bufferu. Pokud je transakce potvrzena pomocí VALIDATE TRANSACTION, jsou změny uloženy trvale. Jestliže je transakce zrušena pomocí CANCEL TRANSACTION, změny nejsou uloženy.

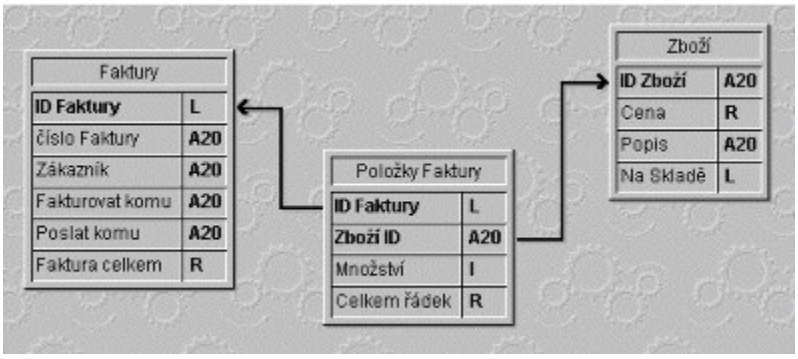
Transakce pracuje s dočasnými adresami záznamů a proto po potvrzení nebo zrušení transakce bude výběr záznamů každé tabulky v platném procesu prázdný. Z tohoto důvodu buďte opatrní při používání pojmenovaných výběrů uvnitř transakce. Po potvrzení nebo zrušení transakce, může pojmenovaný výběr vytvořený před nebo během transakce, obsahovat neplatné adresy na záznamy. Pojmenovaný výběr může například obsahovat adresu na vymazaný záznam nebo dočasnou adresu na záznamy vytvořené během transakce. Toto upozornění platí také pro sady, protože ty používají bitové tabulky adres záznamů.

Následující příkazy používají trvalá čísla záznamů - nepoužívejte je během transakce:
- GOTO RECORD
- RELATE ONE SELECTION
- RELATE MANY SELECTION

Příklad transakce

V tomto příkladu je databází jednoduchý systém faktur. Položky faktur jsou uloženy v tabulce s názvem [Položky Faktury], která je vztažena k tabulce [Faktury] pomocí polí [Faktury]Faktura ID a [Položky Faktury]Faktura ID. Jakmile je přidána nová faktura, je vytvořeno jedinečné ID s použitím příkazu Sequence number. Vztah mezi těmito tabulkami je automatický. Je označeno políčko Auto přiřazení vztažených hodnot.

Vztah mezi tabulkami [Položky Faktury] a [Zboží] je ruční.



Při přidání faktury jsou provedeny následující akce:
- Přidání záznamu do tabulky [Faktury]
- Přidání záznamů do tabulky [Položky Faktury]
- Obnovení pole [Zboží]Na Skladě pro každou položku ve faktuře

Tento příklad je typický případ pro použití transakce. Musíte si být jisti, že můžete uložit všechny tyto záznamy po provedení operace najednou nebo že je můžete všechny najednou zrušit pokud není možné některé záznamy přidat

nebo obnovit. Jinými slovy, musíte uložit vztažená data.

Pokud nepoužijete transakci, nemůžete zaručit integritu dat ve vaší databázi. Pokud například bude jeden záznam v tabulce [Zboží] uzamčen, nebudete moci obnovit počet na skladě. Proto bude mít toto pole nesprávnou hodnotu. Součet prodaných částí a částí na skladě se nebude rovnat originální hodnotě v záznamu. Této situaci se můžete vyhnout s použitím transakce.

Existuje několik způsobů jak zadat data pomocí transakcí:

1. Můžete nechat 4D ovládat transakce s použitím vlastnosti **Automatické transakce během vstupu dat** v okně Předvolby databáze v prostředí Návrháře. V tomto případě 4D založí transakci a uloží ji nebo zruší podle toho, jestli potvrdíte nebo zrušíte vstup dat v mateřském formuláři. Vstup dat z formuláře, který obsahuje vztaženou tabulku v podformuláři vyžaduje transakce. Tato možnost je použita v celé databázi.

Pokud sami chcete ovládat transakce, musíte použít příkazy START TRANSACTION, VALIDATE TRANSACTION a CANCEL TRANSACTION.

2. Můžete napsat:

```
READ WRITE([Položky Faktury])
READ WRITE([Zboží])
INPUT FORM([Faktury];"Vstup")
Repeat
   START TRANSACTION
   ADD RECORD([Faktury])
   If (OK=1)
      VALIDATE TRANSACTION
   Else
      CANCEL TRANSACTION
   End if
Until (OK=0)
READ ONLY(*)
```

3. K omezení zamykání záznamů při vstupu dat, můžete také řídit transakce z metody formuláře a zpřístupnit tabulku do módu číst/psát pouze když je to potřeba.

K provedení vstupu dat použijete vstupní formulář tabulky [Faktury], který obsahuje podformulář z tabulky [Položky Faktury]. Formulář má dvě tlačítka bCancel a bOK, které jsou obě bez automatické akce.

Smyčka pro přidání záznamů:

```
READ WRITE([Položky Faktury])
READ ONLY([Zboží])
INPUT FORM([Faktury];"Vstup")
Repeat
   ADD RECORD([Faktury])
Until (bOK=0)
READ ONLY([Položky Faktury])
```

Všimněte si, že tabulka [Zboží] je během vstupu dat v módu pouze číst. Mód číst/psát je povolen pouze pokud je vstup dat potvrzen.

Transakce je zapnuta v metodě vstupního formuláře tabulky [Faktury]:

```
Case of
   ÷ (Form Event=On Load)
```

```
        START TRANSACTION
        [Faktury]Faktura ID:=Sequence number([Faktury]Faktura ID)
    Else
       [Faktury]Faktura Celkem:=Sum([Položky Faktury]Total line)
    End case
```

Pokud klepnete na tlačítko bCancel, musí být zrušen jak vstup dat tak i transakce. Zde je metoda objektu bCancel:

```
    Case of
       ÷ (Form Event=On Clicked)
          CANCEL TRANSACTION
          CANCEL
    End case
```

Pokud klepnete na tlačítko bOK, musí být potvrzen vstup dat i transakce. Zde je metoda objektu tlačítka bOK:

```
    Case of
       ÷ (Form Event=On Clicked)
          $NbLines:=Records in selection([Položky Faktury])
          READ WRITE([Zboží]) ` Přepnout do módu číst/psát pro tabulku [Zboží]
          FIRST RECORD([Položky Faktury]) ` Začít na prvním řádku
          $ValidTrans:=True ` Předpokládáme že je vše OK
          For ($Line;1;$NbLines) ` Pro každý řádek
             RELATE ONE([Položky Faktury]Zboží ID)
             OK:=1 ` Předpokládáme že chcete pokračovat
                ` Zkusit zpřístupnit záznam v číst/psát módu
             While (Locked([Zboží]) & (OK=1))
                CONFIRM("Část "+[Položky Faktury]Zboží ID+" se používá. Čekat?")
                If (OK=1)
                   DELAY PROCESS( Current process;60)
                   LOAD RECORD([Zboží])
                End if
             End while
             If (OK=1)
                   ` Obnovit množství na skladě
                [Zboží]Na Skladě:=[Zboží]Na Skladě-[Položky Faktury]Množství
                SAVE RECORD([Zboží]) ` uložit záznam
             Else
                $Line:=$NbLines+1 ` Opustit smyčku
                $ValidTrans:=False
             End if
             NEXT RECORD([Položky Faktury]) ` Na další řádek
          End for
          READ ONLY([Zboží]) ` Nastavit tabulku na pouze číst
          If ($ValidTrans)
             SAVE RECORD([Faktury]) ` Uložit záznam faktury
             VALIDATE TRANSACTION ` Potvrdit všechny akce v databázi
          Else
             CANCEL TRANSACTION ` Zrušit vše
          End if
          CANCEL ` Opustit formulář
```

End case

V tomto kódu provádíme příkaz CANCEL bez rozdílu na které tlačítko bylo klepnuto. Nový záznam není uložen příkazem ACCEPT ale příkazem SAVE RECORD. Všimněte si, že SAVE RECORD je provedeno před příkazem VALIDATE TRANSACTION a proto bude uložení záznamu částí transakce. Provedení příkazu ACCEPT může také potvrdit záznam, ale v tomto případě může být transakce potvrzena před uložením záznamu. Jinými slovy může být záznam uložen mimo transakci.

Podle vašich potřeb, můžete nechat 4D aby ovládala transakce, nebo můžete upravit databázi sami, jak bylo ukázáno v těchto příkladech.V posledním příkladu může být ovládání zamčených záznamů v tabulce [Zboží] vyvíjeno dále.

Dále si přečtěte

CANCEL TRANSACTION, In transaction, START TRANSACTION, VALIDATE TRANSACTION.

Příkazy a odkazy pro Transakce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

START TRANSACTION (ZAČÍT TRANSAKCI)

Verze 3

START TRANSACTION

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz START TRANSACTION začne v platném procesu transakci. Všechny změny v databázi jsou dočasné, dokud není transakce potvrzena nebo zrušena.

Pokud máte několik globálních procesů, můžete mít několik transakcí. Transakce nemůžete hromadit. Pokud začnete jinou transakci uvnitř transakce, 4D bude druhou ignorovat.

Dále si přečtěte

CANCEL TRANSACTION, In transaction, Použití transakcí, VALIDATE TRANSACTION.

Příkazy a odkazy pro Transakce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

VALIDATE TRANSACTION (POTVRDIT TRANSAKCI)

Verze 3

VALIDATE TRANSACTION

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

VALIDATE TRANSACTION potvrdí transakci v platném procesu, která byla spuštěna příkazem START TRANSACTION. VALIDATE TRANSACTION uloží změny provedené v databázi během transakce.

Dále si přečtěte

CANCEL TRANSACTION, In transaction, START TRANSACTION, Použití transakcí.

Příkazy a odkazy pro Transakce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CANCEL TRANSACTION (ZRUŠIT TRANSAKCI)

Příkazy a odkazy pro Transakce

Verze 3

CANCEL TRANSACTION

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

CANCEL TRANSACTION zruší transakci v platném procesu spuštěnou příkazem START TRANSACTION.
CANCEL TRANSACTION ponechá databázi beze změn provedených během transakce.

Dále si přečtěte

In transaction, START TRANSACTION, Použití transakcí, VALIDATE TRANSACTION.

Příkazy a odkazy pro Transakce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

In transaction (V transakci)

Příkazy a odkazy pro Transakce

Verze 6.0

In transaction → Logické

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Logické	←	Vrací True pokud je platný proces v transakci

Popis

Příkaz In transaction vrací True pokud je platný proces v transakci, jinak vrací False.

Příklad

Pokud provádíte operace s více záznamy (přidávání, upravování nebo mazání záznamů), můžete se setkat se zamčenými záznamy. V tomto případě, pokud potřebujete zachovat integritu dat, musíte být v transakci a tak se můžete vrátit na začátek akce a ponechat databázi beze změn.

Pokud provádíte akci z triggeru nebo podprogramu formuláře (ten může být volán ať už je transakce nebo není), můžete použít In transaction k testování jestli metoda procesu nebo volající metoda začala transakci. Pokud transakce nebyla zapnuta, nezačnete akci, protože víte, že ji nebudete moci vrátit.

Dále si přečtěte

CANCEL TRANSACTION, START TRANSACTION, Triggery, VALIDATE TRANSACTION.

Příkazy a odkazy pro Transakce

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Triggery

Příkazy a odkazy pro Triggery

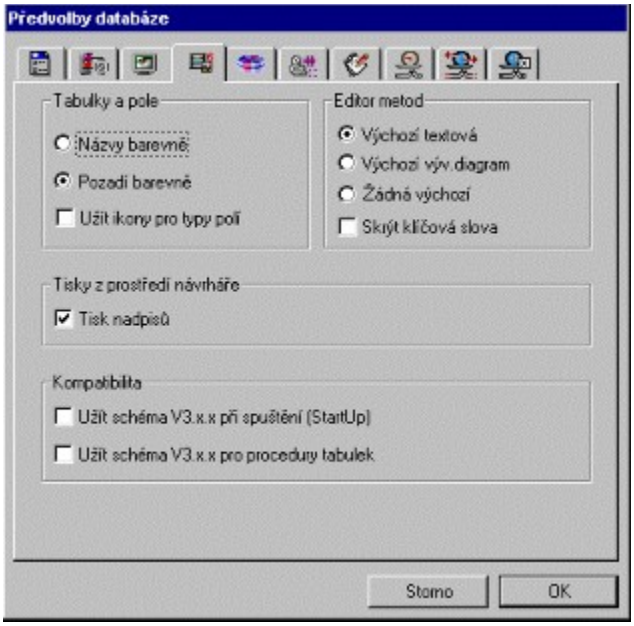
Verze 6.0

Trigger je metoda přiřazená k tabulce. Je to vlastnost tabulky. Vy sami nemůžete trigger volat; je automaticky volán 4D když pracujete se záznamy (přidání, vymazání, upravení a načtení). Můžete začít s velmi jednoduchým triggerem a postupně jej upravovat.

Triggery mohou zabránit nesprávným operacím se záznamy. Je to velmi silný nástroj pro hlídání operací se záznamy a stejně i pro zabránění ztráty dat nebo zabránění neoprávněnému zásahu. Například v systému faktur, můžete zabránit přidání faktury bez definování zákazníka, pro kterého je faktura vystavována.

Kompatibilita s předchozí verzí 4D

Triggery jsou nový typ metod obsažený ve verzi 6. V předchozí verzi 4D existovaly metody tabulky (procedury souboru) a byly prováděny pouze když byl pro vstup dat, zobrazení nebo tisk použit formulář - byly používány velmi málo. Triggery jsou prováděny na mnohem nižší úrovni než staré procedury tabulek. Nezáleží na tom jestli se záznamem provádíte akci uživatelskou (jako přidání záznamu) nebo programátorskou (jako volání příkazu SAVE RECORD), trigger tabulky bude vždy při této akci proveden. Tento nový způsob se velmi liší od procedur tabulek. Pokud převádíte databázi z verze 3 a chcete využít nové způsoby triggerů, musíte odznačit možnost **Užít schéma V3.x.x pro procedury tabulek** v **Předvolbách databáze**, jejichž okno je zde ukázáno:



Aktivování a vytvoření Triggeru

Jako výchozí nemá tabulka vytvořená v prostředí návrháře žádný trigger.

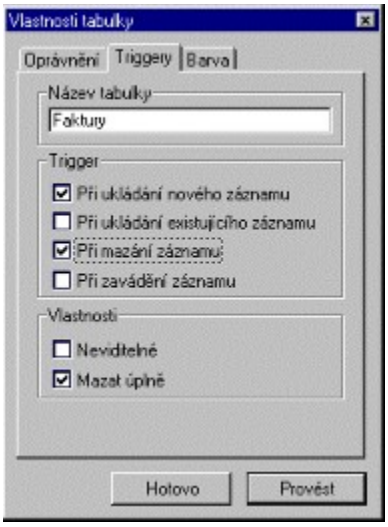
K použití triggeru pro tabulku musíte:

- Aktivovat trigger a říci 4D kdy jej má provádět.
- Napsat kód triggeru

Aktivování triggeru pokud není napsán jeho kód nebo napsání triggeru bez jeho aktivování, nemá žádný vliv na akce prováděné na tabulce.

1. Aktivování triggeru

K aktivování triggeru musíte označit nějaký z **Triggerů** (událostí databáze) pro tabulku v okně **Vlastnosti tabulky**:



Při ukládání nového záznamu

Při označení této možnosti bude trigger spuštěn pokaždé, když bude do tabulky přidán nový záznam.

To se stane když:

• Přidání záznamu ze vstupu dat (v prostředí uživatele nebo příkazem ADD RECORD)
• Vytvoření a uložení záznamu pomocí CREATE RECORD a SAVE RECORD. Trigger bude volán ve chvíli provedení příkazu SAVE RECORD.
• Importování záznamů (Prostředí uživatele nebo použití příkazů importu)
• Provedení ostatních příkazů které vytvářejí a/nebo ukládají záznam (např. ARRAY TO SELECTION, SAVE RELATED ONE, atd.).
• Použití plug-in, které provádějí příkazy CREATE RECORD a SAVE RECORD.

Při ukládání existujícího záznamu

Při označení této možnosti bude trigger spuštěn pokaždé, když bude záznam tabulky upraven.

To se stane když:

• Upravení záznamu ze vstupu dat (Prostředí uživatele nebo příkaz MODIFY RECORD)
• Uložení existujícího záznamu příkazem SAVE RECORD.
• Provedení ostatních příkazů které ukládají existující záznam (např. ARRAY TO SELECTION, APPLY TO SELECTION, MODIFY SELECTION, atd.).
• Použití plug-in který provede příkaz SAVE RECORD.

Při mazání záznamu

Pokud je označena tato možnost, bude trigger proveden pokaždé když bude vymazán záznam z tabulky.

To se stane když:

• Vymazání záznamu (prostředí uživatele nebo příkaz DELETE RECORD nebo DELETE SELECTION).
• Provedení akce která vymaže vztažené záznamy přes kontrolu mazání v nastavení vztahu.
• Použití plug-in který provede příkaz DELETE RECORD.

Při zavádění záznamu

Při označení této možnosti bude trigger proveden pokaždé, když bude záznam tabulky načten. Obsahuje to všechny situace při kterých je načte platný záznam z datového souboru. Tuto možnost budete používat častěji než předchozí tři.

Pro optimalizaci operací 4D, možnost Při zavádění záznamu nikdy nevolá trigger při použití příkazu který používá indexy.

Pokud jsou používány indexy, záznamy nejsou načteny a naopak když indexy nejsou používány (pokud používané pole není indexované) jsou záznamy načteny. To může znamenat že trigger, pro který je označena možnost Při zavádění záznamu, bude nebo nebude proveden podle vlastnosti Indexované pro prováděné pole. Spíše než předání chování, které by bylo těžké předvídat, bylo rozhodnuto že trigger s označenou možností Při zavádění záznamu se nebude provádět, pokud je používám příkaz který používá indexy.

Poznámka: Pokud je označena možnost Při zavádění záznamu, trigger se bude provádět při načtení existujícího záznamu z datového souboru, až na následující výjimky:
• Dotazy: Uživatel provádí dotaz pomocí standardního okna dotazu nebo pomocí příkazů QUERY nebo QUERY SELECTION.
• Třídění: Třídění které bylo vytvořeno ve standardním okně Třídění nebo s použitím příkazu QRDER BY.
• Matematické řady: Sum, Average, Min, Max, Std deviation, Variance, Sum square.
• Příkazy: RELATE ONE SELECTION, RELATE MANY SELECTION.

DŮLEŽITÉ: Pokud provádíte akci na více záznamech, trigger je volán pro každý záznam. Pokud provedete APPLY TO SELECTION na tabulku, která obsahuje 100 záznamů, bude trigger proveden 100x.

2. Vytvoření triggeru

Pro vytvoření triggeru použijte **Průzkumníka** nebo stiskněte klávesu Alt (Windows) nebo Option (Macintosh) a poklepejte na titul tabulky v okně struktury. Více informací najdete v *Přiručce návrháře 4th Dimension.*

Události databáze a triggery

Trigger může být proveden pro jednu ze čtyř **událostí databáze**. Uvnitř triggeru můžete tyto události rozeznat příkazem Database event. Toto funkce vrací číselnou hodnotu, která označuje událost databáze.

Obvykle budete psát trigger ve struktuře **Case of** podle výsledků funkce Database event:

```
   ` Trigger pro [maTabulka]
C_LONGINT($0)
$0:=0
Case of
   ÷ (Database event=On Saving New Record Event)
      ` Provést akci pro uložení nového záznamu
   ÷ (Database event=On Saving Existing Record Event)
      ` Provést akci pro uložení existujícího záznamu
```

```
    ÷ (Database event=On Deleting Record Event)
        ` Provést akci pro vymazání záznamu
    ÷ (Database event=On Loading Record Event)
        ` Provést akci pro načtení záznamu do paměti
End case
```

Triggery jsou funkce

Trigger má dva účely:

- Provést akci se záznamem před uložením nebo vymazáním a před načtením
- Uznání nebo odmítnutí akce databáze

1. Provedení akce

Pokaždé když je záznam uložen (přidán nebo upraven) do tabulky [Dokumenty], chcete uznačit záznam časovou značkou pro vytvoření a další pro poslední změnu. Můžete napsat následující trigger:

```
    ` Trigger pro tabulku [Dokumenty]
Case of
    ÷ (Database event=On Saving New Record Event)
        [Dokumenty]Kdy vytvoreno:=Časová značka
        [Dokumenty]Kdy upraveno:= Časová značka
    ÷ (Database event=On Saving Existing Record Event)
        [Dokumenty]Kdy upraveno:= Časová značka
End case
```

Poznámka: Funkce Časová značka je malá metoda která vrací počet sekund od pevného datumu.

Po napsání a aktivování tohoto triggeru už nezáleží na způsobu přidání nebo upravení záznamu v tabulce [Dokumenty], pole [Dokumenty]Kdy vytvoreno a [Dokumenty]Kdy upraveno budou vždy obnoveny.

Poznámka: Podívejte se na příklad u příkazu GET DATABASE PROPERTIES pro kompletní popis tohoto příkladu.

2. Uznání nebo odmítnutí akce databáze

K přijetí nebo odmítnutí akce databáze musí trigger vrátit **číslo chyby triggeru** do výsledku funkce $0.

Příklad

Vezměme si příklad tabulky [Zaměstnanci]. Během vstupu dat chcete vnutit pravidla o poli [Zaměstnanci]Sociální pojištění. Jakmile klepnete na tlačítko uložit, proběhne kontrola pole pomocí metody objektu:

```
    ` Metoda objektu bAccept tlačítko
If (Good SS number ([Zaměstnanci]Sociální pojištění))
    ACCEPT
Else
    BEEP
    ALERT ("Vložte číslo sociálního pojištění a znovu klepněte na OK.")
End if
```

Pokud je hodnota pole správná, přijmete záznam; pokud je hodnota pole nesprávná, zobrazí se upozornění a zůstanete ve vstupu dat.

Pokud vytvoříte záznam Zaměstnance programově, může být následující kousek kódu programové uložení záznamu,

ale může porušit pravidlo zadané v předchozím příkladu.

```
   ` Část metody projektu
   ` ...
CREATE RECORD ([Zaměstnanci])
[Zaměstnanci]Jméno :="DOE"
SAVE RECORD ([Zaměstnanci]) ` Porušení pravidel DB! Číslo SP nebylo zadáno!
   ` ...
```

S použitím triggeru pro tabulku [Zaměstnanci], můžete zajistit zadání pole [Zaměstnanci]Sociální pojištění ve všech úrovních databáze. Trigger může vypadat následovně:

```
   ` Trigger pro [Zaměstnanci]
$0:=0
$dbEvent:= Database event
Case of
   ÷ (($dbEvent=On Saving New Record Event) | ($dbEvent=On Saving Existing Record Event))
      If (Not(Good SS number ([Zaměstnanci]Sociální pojištění)))
         $0:=-15050
      Else
         ` ...
      End if
      ` ...
End case
```

Jakmile je tento trigger napsán a aktivován, řádek SAVE RECORD ([Zaměstnanci]) generuje chybu -15050 a záznam NEBUDE uložen.

Stejně tak 4D plug-in, který bude ukládat záznam Zaměstnance bez Sociálního pojištění. Trigger bude proveden a vrátí stejnou chybu a záznam nebude uložen.

Trigger zajišťuje že nikdo (uživatel, návrhář databáze, Plug-in, 4D Open klient se 4D Server) nemůže porušit pravidla daná o sociálním pojištění.

Poznamenejte si, že i když nemáte trigger pro tabulku, můžete dostávat chyby databáze při ukládání nebo mazání záznamů. Například pokud se pokusíte uložit duplikovanou hodnotu do jedinečného indexovaného pole, vznikne chyba -9998.

Proto triggery, které vrací chyby přidávají nové chyby databáze do vaší aplikace:

- 4D řídí normální chyby: jedinečný index, kontrola vztažených dat, atd.
- S použitím trigeru můžete řídit vlastní chyby jedinečné pro vaší databázi.

Důležité: Můžete vrátit hodnotu chyby podle vašeho výběru. Nicméně NEPOUŽÍVEJTE kódy chyb které již 4D používá. Doporučujeme vám používat chyby mezi -32000 a -15000. Kódy chyb nad -15000 jsou vyhrazeny pro Chyby databázového stroje.

Na úrovni procesu ovládáte chyby triggeru stejným způsobem jako u chyb "motoru" databáze:

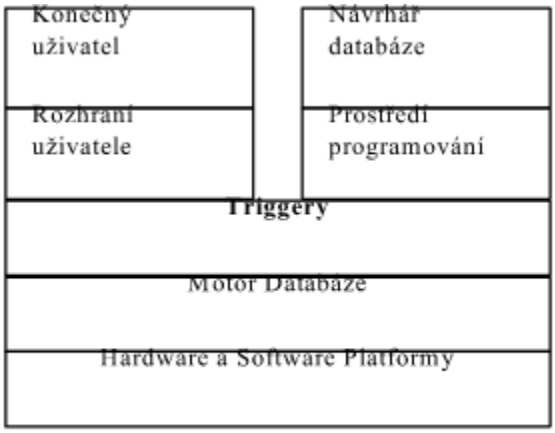
- Můžete nechat 4D zobrazit standardní dialogové okno chyby, pak bude metoda zastavena.
- Můžete použít metodu pro zachytávání chyb istalovanou pomocí příkazu ON ERR CALL a opravit chybu patřičným způsobem.

Poznámka: Pokud během vstupu dat je vrácena chyba triggeru pro mazání nebo potvrzení záznamu, je s chybou zacházeno stejně jako s chybou jedinečného indexovaného pole. Je zobrazen dialog chyby a vy zůstanete v zadávání dat. I když používáte databázi pouze v prostředí uživatele (ne vlastní nabídky), budete mít prospěch z používání triggerů.

I pokud trigger nevrátí žádnou chybu ($0:=0), neznamená to že operace databáze bude provedena - může se objevit komplikace jedinečných indexů. Pokud je akce obnovení záznamu, může být záznam zamčený, může se objevit chyba I/O, atd. Zkoušení je zakončeno po dokončení triggeru. Nicméně na vyšší úrovni provádění procesu jsou Chyby databázového stroje nebo chyby triggeru stejné - chyba triggeru je chyba motoru databáze.

Triggery a architektura 4D

Triggery se provádějí na úrovni enginu (motoru) databáze. Je to znázorněno v následujícím obrázku:



Triggery jsou prováděny na počítači kde je umístěn motor databáze. Toto platí pro 4D jednouživatskou. Na 4D Server jsou triggery prováděny v prováděném procesu na stroji serveru a ne na stroji klienta.

Jakmile je trigger zavolán, je proveden v kontextu procesu, ve kterém byla provedena událost databáze. Proces který vyvolal trigger je nazýván **vyvolávací proces**.

Trigger dále pracuje s platným výběrem, platným záznamem, módy tabulky pro číst/psát se zamykáním záznamů vyvolávacího procesu.

Upozornění: Trigger nemůže a nesmí měnit platný záznam tabulky, ke které je přiřazen. Pokud v triggeru potřebujete testovat jedinečnou hodnotu na více záznamů, použijte příkaz SET QUERY DESTINATION, který umožňuje provést dotaz bez změny platného výběru a platného záznamu tabulky.

Buďte opatrní při používání jiných databází nebo objektů jazyka prostředí 4D, protože trigger se může provádět na jiném počítači než je vyvolávací proces - to je případ 4D Serveru!

• **Meziprocení proměnné**: Trigger má přístup k meziprocesním proměnným počítače na kterém je prováděn. Na 4D Serveru to může být jiný počítač než je vyvolavacího procesu.
• **Procesní proměnné**: Nezávislá tabulka proměnných procesu je přenášena všemi triggery. Trigger nemá přístup k proměnným vyvolávacího procesu.
• **Místní proměnné**: V triggeru můžete použít místní proměnné. Jejich rozsah je pouze trigger; jsou vytvářeny/mazány při každém provedení triggeru.
• **Semafory**: Trigger může testovat a vytvořit globální semafor stejně jako místní semafor (na počítači kde je prováděn). Nicméně trigger musí být velmi rychlý a proto buďte opatrní při testování a nastavování semaforů z triggeru.
• **Sady a Pojmenované výběry**: Pokud v triggeru použijte sadu nebo pojmenovaný výběr, pracujete na počítači kde je trigger prováděn.

• **Rozhraní uživatele**: NEPOUŽÍVEJTE prvky rozhraní uživatele v triggeru (žádná upozornění, zprávy nebo dialogy). Měli by jste také omezit krokování v triggeru. Nezapomeňte že v klient/server jsou triggery prováděny na stroji 4D Server. Jakékoli upozornění na serveru nepomůže uživateli na klientovi. Rozhraní uživatele ovládajte z vyvolávacího procesu.

Triggery a Transakce

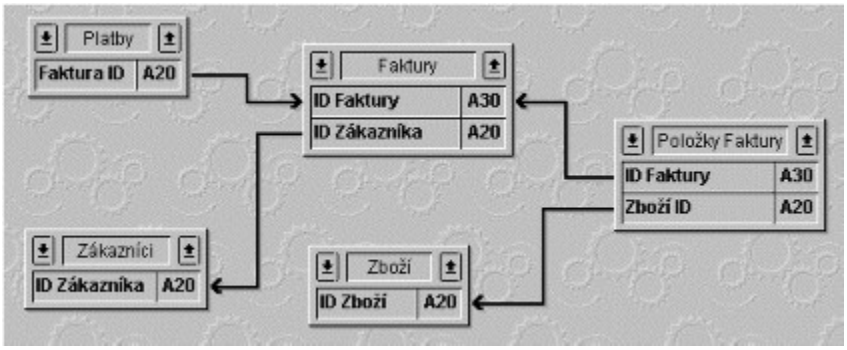
Transakce musíte ovládat na úrovni vyvolávacího procesu. Neřiďte transakce z triggeru. Během jednoho provedení triggeru, jestliže přidáváte, měníte nebo mažete více záznamů (přečtěte si další část), musíte nejdříve v triggeru použít příkaz In transaction aby jste zjistili jestli vyvolávací proces je již v transakci. Pokud není, trigger by mohl narazit na zamčené záznamy. Proto pokud není vyvolávací proces v transakci, nezačínejte v triggeru operace na záznamech. Pouze vraťte chybu do $0, která oznámí vyvolávacímu procesu že pro prováděné akce musí být v transakci.

Jinak pokud by se trigger setkal se zamčeným záznamem neměl by vyvolávací proces šanci akci provedenou triggerem vrátit.

Poznámka: Pro optimalizaci chování při kombinování triggerů a transakcí, 4D neprovádí trigger po provedení příkazu VALIDATE TRANSACTION. Toto zabrání triggeru aby byl proveden dvakrát.

Kaskádování triggerů

Předpokládejme následující strukturu:



Poznámka: Tabulky jsou neksou vidět celé; mají více polí než je zobrazno zde.

Řekněme že databáze kontroluje vymazávání faktur. Můžeme vyzkoušet jak by tato operace byla řízena na úrovni triggerů (mazání můžete řídit i na úrovni procesu).

Aby jsme zachovali integritu vztažených dat, vymazání faktury vyžaduje v triggeru [Faktury] provést následující akce:

• V záznamu tabulky [Zákazníci] zmenšit hodnotu pole Celkové prodeje o hodnotu faktury.
• Vymazat všechny vztažené záznamy z tabulky [Položky Faktury].
• Musíte také snížit hodnotu pole [Produkty]Prodáno o hodnotu v položkách faktur, které byly vymazány.
• Vymazat všechny vztažené záznamy z tabulky [Platby]

Za prvé, trigger pro [Faktury] musí provést tyto akce v transakci, tak aby se volající proces mohl vrátit zpět pokud by trigger narazil na zamčené záznamy.

Za druhé, trigger pro [Položky Faktury] je **kaskádován** triggerem [Faktury]. Trigger [Položky Faktury] je prováděn z triggeru [Faktury], protože mazání položek probíhá pomocí příkazu DELETE SELECTION z triggeru [Faktury].

Předpokládejme že všechny triggery z tohoto příkladu jsou aktivovány pro všechny události databáze. Kaskáda triggerů bude:

• Trigger [Faktury] je zavolán protože je vymazána faktura
• Trigger [Zákazníci] je zavolán protože trigger [Faktury] obnovuje pole Celkové prodeje
• Trigger [Položky Faktury] je zavolán protože trigger [Faktury] vymaže položku faktury (opakovaně)
• Trigger [Produkty] je zavolán protože trigger [Položky Faktury] obnovuje pole Prodáno
• Trigger [Platby] je zavolán protože trigger [Faktury] vymaže záznamy které patří k mazané faktuře (opakovaně)

V tomto vztahu je trigger [Faktury] prováděn v úrovni 1, triggery [Zákazníci], [Položky Faktury] a [Platby] v úrovni 2 a trigger [Produkty] je prováděn v úrovni 3.

Z triggeru můžete provést příkaz Trigger level k zjištění na které úrovni je trigger prováděn. Dále můžete použít příkaz TRIGGER PROPERTIES k získání informací o ostatních úrovních.

Například pokud bude záznam v tabulce [Produkty] vymazán na úrovni procesu, bude trigger [Produkty] v úrovni 1 a ne 3.

Používáním Trigger level a TRIGGER PROPERTIESmůžete detekovat důvod akce. V našem případě je faktura vymazána na úrovni procesu. Pokud vymažeme záznam v [Zákazníci] na úrovni procesu, pak se trigger [Zákazníci] může pokusit vymazat všechny faktury tohoto zákazníka. To znamená že trigger [Zákazníci] bude proveden stejně jako v předchozím případě, ale z jiného důvodu. V tiggeru [Faktury] můžete detekovat jestli je prováděn v úrovni 1 nebo 2. Pokud je v úrovni 2, můžete pak vynechat určitou část, protože záznam v tabulce [Zákazníci] je již vymazán. Nepotřebujete obnovovat pole Celkové prodeje.

Požití Pořadového čísla v Triggeru

Při ovládání události Při ukládání nového záznamu, můžete provést příkaz Sequence number k získání jedinečného ID pro záznam tabulky.

Příklad

```
   ` Trigger pro tabulku [Faktury]
Case of
   ÷ (Database event=On Saving new record)
      ` ...
      [Faktury]Faktura ID Number:=Sequence number ([Faktury])
      ` ...
End case
```

Dále si přečtěte

Database event, Metody, Record number, Trigger level, TRIGGER PROPERTIES.

Příkazy a odkazy pro Triggery

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Database event (Událost databáze)

Příkazy a odkazy pro Triggery

Verze 6.0

Database event → Číslo

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Číslo	←	0 vně všech akcí triggerů 1 ukládá nový záznam 2 ukládá existující záznam 3 maže záznam 4 zavádí záznam

Popis

Volaný z triggeru, vrací příkaz Database event číselnou hodnotu která označuje událost databáze. Jinými slovy důvod, proč je trigger prováděn.

Máte k dispozici následující předdefinované konstanty:

Konstatnta	***Typ***	***Hodnota***
On Saving New Record Event	*Long Integer*	*1*
On Saving Existing Record Event	*Long Integer*	*2*
On Deleting Record Event	*Long Integer*	*3*
On Loading Record Event	*Long Integer*	*4*

Pokud z triggeru provádíte operace na více záznamů, můžete narazit na podmínky (obvykle zamčené záznamy), které znemožní správné provedení triggeru. Příklad této situace je obnovování záznamů v tabulce [Produkty], když je zadáván záznam do tabulky [Faktury]. V tomto případě musíte zastavit provádění operace a vrátit chybu databáze, aby jste upozornili vyvolávací proces, že tato akce nemůže být provedena. Pak musíte mít možnost tuto nedokončenou akci zrušit, pomocí transakce. Jakmile nastane tato situace, potřebujete vědět, jestli je proces v transakci před provedením této akce, což můžete zjistit příkazem In transaction.

Při kaskádovém volání triggerů, nemá 4D žádná omezení kromě paměti. K optimalizaci provádění triggerů, můžete potřebovat triggery řídit nejen podle události databáze ale také podle úrovně ve které jsou provedeny. Například v databázi fakturace při mazání záznamu faktury, budete chtít obnovit pole [Zákazníci]Celkové prodeje a naopak při vymazání záznamu zákazníka budete chtít vymazat všechny faktury vztažené k tomuto zákazníkovi. K tomuto můžete použít příkazy Trigger level a TRIGGER PROPERTIES.

Příklad

Příkaz Database event použijete k následovnému strukturování triggeru:

```
   ` Trigger pro [maTabulka]
C_LONGINT($0)
$0:=0
Case of
   ÷ (Database event=On Saving New Record Event)
      ` Provést akci pro uložení nového záznamu
   ÷ (Database event=On Saving Existing Record Event)
```

```
      ` Provést akci pro uložení existujícího záznamu
   ÷ (Database event=On Deleting Record Event)
      ` Provést akci pro vymazání záznamu
   ÷ (Database event=On Loading Record Event)
      ` Provést akci pro načtení záznamu do paměti
End case
```

Dále si přečtěte

In transaction, Trigger level, TRIGGER PROPERTIES, Triggery.

Příkazy a odkazy pro Triggery

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Trigger level (Úroveň triggeru)

Příkazy a odkazy pro Triggery

Verze 6.0

Trigger level → Číslo

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Číslo	←	Úroveň provádění triggeru (0 je-li mimo jakýkoliv cyklus triggeru)

Popis

Příkaz Trigger level vrací úroveň triggeru na které je prováděn.

Pro podrobnější informace o úrovních provádění si přečtěte téma Kaskádování triggerů v části Triggery.

Dále si přečtěte

Database event, TRIGGER PROPERTIES, Triggery.

Příkazy a odkazy pro Triggery

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

TRIGGER PROPERTIES (VLASTNOSTI TRIGGERU)

Verze 6.0

TRIGGER PROPERTIES (úroveňTriggeru; dbUdálost; tabČíslo; záznamČíslo)

Parametr	Typ		Popis
úroveňTriggeru	číslo	→	Úroveň provádění triggeru
dbUdálost	číslo	←	událost databáze
tabČíslo	číslo	←	číslo dotčené tabulky
záznamČíslo	číslo	←	číslo dotčeného záznamu

Popis

Příkaz TRIGGER PROPERTIES vrací informace o úrovni provádění triggeru, kterou předáte do parametru *úroveňTriggeru*. TRIGGER PROPERTIES budete používat ve spojení s Trigger level pro provedení různých akcí podle úrovně provádění triggeru. Pro podrobnější informace o úrovních provádění si přečtěte téma Kaskádování triggerů v části Triggery.

Pokud předáte neexistující úroveň triggeru, příkaz vrátí 0 (nula) do všech parametrů.

Událost databáze pro kterou byl trigger prováděn je vrácena do parametru *tabUdálost*. Může obsahovat následující konstanty:

Konstatnta	*Typ*	*Hodnota*
On Saving New Record Event	*Long Integer*	*1*
On Saving Existing Record Event	*Long Integer*	*2*
On Deleting Record Event	*Long Integer*	*3*
On Loading Record Event	*Long Integer*	*4*

Číslo tabulky a číslo záznamu pro kterou je trigger prováděn je vrácen do parametrů *tabČíslo* a *záznamČíslo*.

Poznámka: Nezapomeňte že pokud je trigger prováděn v transakci tak mají nové záznamy dočasné číslo záznamu.

Dále si přečtěte

O číslech záznamů, Database event, Trigger level, Triggery.

Příkazy a odkazy pro Triggery

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

BEEP (PÍPNOUT)

Příkazy a odkazy pro Rozhraní uživatele

Verze 3

BEEP

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz BEEP donutí PC nebo Macintosh pípnout. Váš počítač (na Windows nebo Macintosh) může vydat jiný zvuk než pípnutí, podle nastavení na kontrolním panelu Zvuku.

Upozornění: Nepoužívejete BEEP v procesech Web spojení, protože pípnutí bude provedeno na stroji Web Serveru a ne na stroji klienta Web prohlížeče.

Příklad

Pokud v následujícím příkladu není nalezen žádný záznam je vytvořeno pípnutí a zobrazí se upozornění:

```
QUERY([Zákazníci];[Zákazníci]Jméno=$vsNameToLookFor)
If (Records in selection([Zákazníci])=0)
   BEEP
   ALERT("Není žádný zákazník s tímto jménem.")
End if
```

Dále si přečtěte

PLAY.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

PLAY (ZAHRÁT)

Příkazy a odkazy pro Rozhraní uživatele

Verze 3

PLAY (objektnázev{; kanál})

Parametr	Typ		Popis
objektnázev	řetězec	→	Název zvuku Windows: soubory.WAV, .MID nebo .AVI libovolná platforma: zdroje snd v res souboru MacOS
kanál	číslo	→	je-li určen, kanál syntetizátoru a asynchronně je-li vynechán, synchronně

Popis pro Windows

Na Windows příkaz PLAY zahraje zvuk (.WAV soubory), MIDI (.MID soubory) nebo Video (.AVI soubory) Windows soubor. Do parametru *objektnázev* předáte celou cestu k souboru který chcete zahrát.

Poznámka: Nemůžete hrát multimediální soubory nebo objekty v asynchronním módu. K tomu použijte OLE služby.

Na Macintoshi nebo na Windows (s určitými rozdíly, podívejte se na Důležitou poznámku) příkaz zahraje zvukový zdroj s názvem názevobjektu na Macintoshi.

Parametr *kanál* definuje Macintosh kanál sysntetizátoru. Pokud není *kanál* definován, je nastven kanál pro jednoduché digitální zvuky a je synchronní. Synchronní znamená, že budou zastaveny všechny akce dokud nebude zvuk dohrán. Pokud je kanál 0, je pro jednoduché digitální zvuky a je synchronní. Asynchronní znamená že akce budou probíhat dále a zvuk bude hrát na pozadí.

K zastavení hraní synchronního zvuku, použijte následující tvrzení:

```
PLAY ("";0)
```

Pokud pracujete s databází jak na Windows tak na Macintoshi, můžete hrát Macintosh zvuky i na Windows. K tomu:

• Na Macintoshi pomocí nástrojů ResEdit nebo Resourcerer, překopírujte požadované `snd ` zdroje do zdrojové části struktury.
• Převeďte databázi z Macintosh na Windows pomocí 4D Transporter.

Důležitá poznámka: Windows verze 4th Dimension nehraje Macintosh zvuky které byli kompromovány pomocí MACE. Pokud vaše Macintosh zvuky nehrají na Windows, zjistěte jestli vyhovuje následujícím požadavkům:

pole zvukového zdroje	*Hodnota (hexadecimal)*
Version	*0x0001*
NbSynth	*0x0001*
SynthResID	*0x0005*
SynthInitOptions	*0x000000A0*
NbSoundCommand	*0x0001*
FirstCommand	*0x8051*

Vnitřní data `snd ` zdroje můžete zjistit pomocí Resorcereru.

Příklady

1. Následující příklad ukazuje jak zahrát video soubor na Windows:

```
$DocRef := Open document ( ""; "AVI")
If (OK=1)
   CLOSE DOCUMENT($DocRef)
   PLAY (Document)
End if
```

2. Následující příklad bude umítěn do metody databáze Při spuštění. Přivítá uživatele zvukem Vítejte zvuk na Macintoshi:

```
PLAY ("Vítejte zvuk") ` Zahraje zvuk Vítejte zvuk
```

Dále si přečtěte

BEEP.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Get platform interface (Získat rozhraní platformy)

Příkazy a odkazy pro Rozhraní uživatele

Verze 6.0

Get platform interface → Číslo

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Číslo	←	Platné rozhraní platformy

Popis

Příkaz Get platform interface vrací číselnou hodnotu která určuje platné rozhraní platformy použité v zobrazení formuláře.

Funkce může vracet následující předdefinované konstanty:

Konstanta	***Typ***	***Hodnota***
Automatic interface	*Long Integer*	*-1*
Macintosh interface	*Long Integer*	*0*
Windows 3.1 interface	*Long Integer*	*1*
Windows 95 interface	*Long Integer*	*2*
Copland interface	*Long Integer*	*3*

Rozhraní platformy můžete změnit pomocí příkazu SET PLATFORM INTERFACE nebo v okně **Předvolby databáze** v prostředí návrháře.

Dále si přečtěte

SET PLATFORM INTERFACE.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET PLATFORM INTERFACE (NASTAVIT ROZHRANÍ PLATFORMY)

Příkazy a odkazy pro Rozhraní uživatele

Verze 3.5

SET PLATFORM INTERFACE (interface)

Parametr	Typ		Popis
interface	Číslo	→	nové nastavení rozhraní platformy: -1 Automaticky dle systému 0 MacOS (System7) 1 Windows 3.1 2 Windows 95 3 Copland

Popis

Příkaz SET PLATFORM INTERFACE nastaví rozhraní platformy použité pro zobrazení formulářů.

Do parametru *interface* vložte jednu z následujících konstant:

Konstanta	*Typ*	*Hodnota*
Automatic interface	*Long Integer*	*-1*
Macintosh interface	*Long Integer*	*0*
Windows 3.1 interface	*Long Integer*	*1*
Windows 95 interface	*Long Integer*	*2*
Copland interface	*Long Integer*	*3*

Příkaz neprovede nic pokud předáná hodnota nemění platné rozhraní platformy.

Poznámka: Rozhraní platformy můžete změnit také v okně **Předvolby databáze** v prostředí návrháře.

Příklad

V architektuře klient/server mohou jednotlivé stanice, Macintosh i Windows, používat rozdílné rozhraní platformy. K tomu můžete provést příkaz SET PLATFORM INTERFACE v metodě databáze Při spuštění:

```
   ` V tomto příkladu jsou nastavení uložená v tabulce [Předvolby]
   ` Nalezne se záznam pro platného uživatele
QUERY([Předvolby];[ Předvolby]Jméno Uživatele=Current User)
If (Records in selection([Předvolby])=0)
      ` Pokud nenalezeno, nalézt výchozí předvolby
      QUERY([Předvolby];[Předvolby]Jméno Uživatele ="Výchozí")
End if
   ` Nastavit rozhraní uživatele podle platného záznamu
SET PLATFORM INTERFACE ([Předvolby]Rozhraní platformy)
```

Dále si přečtěte

Get platform interface.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET TABLE TITLES (NASTAVIT NÁZVY TABULEK)

Příkazy a odkazy pro Rozhraní uživatele

Verze 6.0

SET TABLE TITLES (tabTituly; tabČísla)

Parametr	Typ		Popis
tabTituly	String Array	→	Názvy tabulek, jak se budou objevovat v dialogových oknech (Dotazy, Rychlé zprávy..)
TabČísla	Číselné Array	→	skutečná čísla tabulek v struktuře

Popis

Příkaz SET TABLE TITLES vám umožní zamaskovat, přejmenovat a přetřídit tabulky vaší databáze když se objevují ve standardních oknech 4th Dimension jako Editor dotazů v prostředí uživatele nebo vlastní nabídky.

Array tabTituly a tabČísla musí být synchonizované. Do array tabTituly zadáte názvy tabulek jak chcete aby se zobrazovali. Pokud nechcete některou tabulku zobrazovat, nevkládejte do array její název nebo nový titul. Tabulky se budou zobrazovat v pořadí v jakém je zadáte do tohoto array. Do array tabČíslo zadáte platná čísla tabulky odpovídající názvu tabulky nebo novému titulu v array tabTituly.

Máte například databázi s tabulkami A, B a C vytvořené v tomto pořadí. Chcete aby se zobrazovali jako X, Y a Z. Dále nechcete aby se objevovala tabulka B. Nakonec chcete aby se objevovali Z a X v tomto pořadí. Proto předáte Z a X do dvou-prvkového array tabTituly a předáte 3 a 1 do dvou-prvkového array tabČísla.

SET TABLE TITLES NEMĚNÍ strukturu vaší databáze. Upravuje pouze použití standardních dialogových oken 4th Dimension, jako je Editor dotazů, v prostředí uživatele a vlastní nabídky. Rozsah příkazu SET TABLE TITLES je pracovní stanice. Výhoda pro klient/server je, že každý klient může "vidět" tabulky databáze jiným způsobem. Příkaz SET TABLE TITLES můžete provést kolikrát potřebujete. Nicméně si poznamenejte, že ovlivňuje pouze další vzhled standardních oken 4th Dimension.

Příkaz SET TABLE TITLES použijte pro:

- Dynamické lokalizování databáze.
- Ukázání tabulek způsobem jaký chcete bez závislosti na aktuálním definování databáze.
- Ukázání tabulek podle identity a vlastního nastavení uživatele.

UPOZORNĚNÍ: SET TABLE TITLES NEZMĚNÍ vlastnost tabulky Neviditelné. Pokud je v prostředí návrháře tabulka označena jako neviditelná, použití v příkazu SET TABLE TITLES ji nemůže zobrazit.

Příklad

- Vytváříte 4D aplikaci kterou hodláte prodávat mezinárodně. Proto musíte dbát na lokalizaci. Co se týká standardních oken 4th Dimension, která se objevují v prostředí uživatele a vlastní nabídky, můžete adresovat lokalizační potřeby s použitím tabulky [Překlady] a mnoha metod projektu k vytvoření a použití polí lokalizovaných pro jakýkoliv počet jazyků.

- Do vaší databáze vložte následující tabulku:

- Pak vytvořte metodu projektu PŘELOZ TABULKY A POLE zobrazenou níže. Tato metoda projde strukturu vaší

databáze a vytvoří všechny potřebné záznamy v tabulce [Překlady] pro překlad do jazyka předáného jako parametr.

```
    ` Metoda projektu PŘELOZ TABULKY A POLE
    ` PŘELOZ TABULKY A POLE ( Řetězec )
    ` PŘELOZ TABULKY A POLE ( Jazyk )
C_STRING(31;$1)
C_LONGINT($vlTabulka;$vlPole)
For ($vlTabulka;1;Count tables) `Smyčka pro všechny tabulky
        ` Zkusit jestli je již založený překlad pro tuto tabulku do zadaného jazyka
   QUERY([Překlady];[Překlady]Aktuální název=Table name($vlTabulka);*)
   QUERY([Překlady]; & ;[Překlady]Jazyk=$1)
   If (Records in selection([Překlady])=0)
            ` Pokud ne, vytvořit záznam
      CREATE RECORD([Překlady])
      [Překlady]Aktuální název:=Table name($vlTabulka)
      [Překlady]Jazyk:=$1
            ` Přeložený název tabuky bude předán
      SAVE RECORD([Překlady])
   End if
   For ($vlPole;1;Count fields($vlTabulka))
            ` Zkusit jestli je již založený překlad pro toto pole do zadaného jazyka
      QUERY([Překlady];[Překlady]Aktuální název=Field name ($vlTabulka;$vlPole);*)
      QUERY([Překlady]; & ;[Překlady]Jazyk=$1)
      If (Records in selection([Překlady])=0)
               ` Pokud ne, založit záznam
         CREATE RECORD([Překlady])
         [Překlady]Aktuální název:=Field name($vlTabulka;$vlPole)
         [Překlady]Jazyk:=$1
               ` Přeložený název pole bude předán
         SAVE RECORD([Překlady])
      End if
   End for
End for
```

• Pokud v této chvíli provedete následující řádek, vytvoří se tolik záznamů pro překlad do zadaného jazyka kolik je názvů tabulek a polí.

PŘELOZ TABULKY A POLE ("Španělsky")

• Po provedení tohoto řádku můžete do pole [Překlad]Přeložený název vložit překlad pro každé pole a tabulku.

• Nakonec, pokud potřebujete v standardních oknech 4th Dimension zobrazit lokalizaci pro Španělský jazyk, napište:

LOKALIZOVANE TABULKY A POLE ("Španělsky")

Metoda projektu LOKALIZOVANE TABULKY A POLE:

```
    ` Metoda projektu LOKALIZOVANE TABULKY A POLE
    ` LOKALIZOVANE TABULKY A POLE ( Řetězec )
    ` LOKALIZOVANE TABULKY A POLE ( Jazyk )
C_STRING(63;$1)
C_LONGINT($vlTabulka;$vlNbTabulka;$vlPole;$vlNbPole)
$vlNbTabulka:=Count tables ` Získat počet tabulek v databázi
    ` Nastavit array které se vloží do SET TABLE TITLES
```

```
ARRAY STRING(31;$asTabulNazev;$vlNbTabulka)
ARRAY INTEGER($aiTableNumber;$vlNbTabulka)
For ($vlTabulka;1;$vlNbTabulka) ` Smyčka pro všechny tabulky
   $asTabulNazev{$vlTabulka}:=Table name($vlTabulka) ` Získat název tabulky
   $aiTableNumber{$vlTabulka}:=$vlTabulka ` Uložit platné číslo tabulky
      ` Najít překlad
   QUERY([Překlady];[Překlady]Aktuální název=$asTabulNazev{$vlTabulka};*)
   QUERY([Překlady]; & ;[Překlady]Jazyk=$1)
   If (Records in table([Překlady])>0)
         ` Pokud existuje, použít přeložený záznam
      $asTabulNazev{$vlTabulka}:=[Překlady]Přeložený název
   End if
   $vlNbPole:=Count fields($vlTabulka) ` Získat počet polí pro tabulku
      ` Nastavit array pro předání do SET FIELD TITLES
   ARRAY STRING(31;$asPoleNazev;$vlNbTabulka)
   ARRAY INTEGER($aiPoleCislo;$vlNbTabulka)
   For ($vlPole;1) ` Smyčka pro všechna pole
      $asPoleNazev{$vlPole}:=Field name($vlTabulka;$vlPole) ` Získat název pole
      $aiPoleCislo{$vlPole}:=$vlPole ` Uložit číslo pole
      QUERY([Překlady];[Překlady]Aktuální název=$asPoleNazev{$vlPole};*)
         ` Najít překlad
      QUERY([Překlady]; & ;[Překlady]Jazyk=$1)
      If (Records in table([Překlady])>0)
            ` Pokud existuje, použít přeložený název
         $asPoleNazev{$vlPole}:=[Překlady]Přeložený název
      End if
   End for
   SORT ARRAY($asPoleNazev;$aiPoleCislo;>)
   SET FIELD TITLES(Table($vlTabulka)→;$asPoleNazev;$aiPoleCislo)
End for
SORT ARRAY($asTabulNazev;$aiTableNumber;>)
SET TABLE TITLES($asTabulNazev;$aiTableNumber)
```

• Všimněte si, že nová lokalizace může být do databáze předána bez upravení nebo překompilování kódu.

Dále si přečtěte

Count tables, SET FIELD TITLES, Table name.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET FIELD TITLES (NASTAVIT NÁZVY POLÍ)

Verze 6.0

SET FIELD TITLES (tabulka | podtabulka; poleTituly; poleČísla)

Parametr	Typ		Popis
tabulka | podtabulka	tabulka | podtabulka	→	Tabulka nebo podtabulka, kde se budou přemapovávat názvy polí
poleTituly	Řetězec Array	→	Názvy polí, jak se budou objevovat v dialogových oknech (Dotazy, Rychlé zprávy..)
poleČísla	Číselný Array	→	skutečná čísla polí v struktuře

Popis

Příkaz SET FIELD TITLES vám umožní zamaskovat, přejmenovat a přetřídit pole tabulky nebo podtabulky když se objevují ve standardních oknech 4th Dimension jako Editor dotazů v prostředí uživatele nebo vlastní nabídky.

Array poleTituly a poleČísla musí být synchonizované. Do array poleTituly zadáte názvy polí jak chcete aby se zobrazovali. Pokud nechcete některé pole zobrazovat, nevkládejte do array jeho název nebo nový titul. Pole se budou zobrazovat v pořadí v jakém je zadáte do tohoto array. Do array poleČíslo zadáte platná čísla polí odpovídající názvu pole nebo novému titulu v array poleTituly.

Máte například tabulku s poli D, E a F vytvořené v tomto pořadí. Chcete aby se zobrazovali jako M, N a O. Dále nechcete aby se objevovala pole E. Nakonec chcete aby so bjevovali O a M v tomto pořadí. Proto předáte O a M do dvou-prvkového array poleTituly a předáte 3 a 1 do dvou-prvkového array poleČísla.

SET FIELD TITLES NEMĚNÍ strukturu vašich tabulek. Upravuje pouze použití standardních dialogových oken 4th Dimension, jako je Editor dotazů, v prostředí uživatele a vlastní nabídky. Rozsah příkazu SET FIELD TITLES je pracovní stanice. Výhoda pro klient/server je, že každý klient může "vidět" pole tabulky jiným způsobem. Příkaz SET FIELD TITLES můžete provést kolikrát potřebujete. Nicméně si poznamenejte, že ovlivňuje pouze další vzhled standardních oken 4th Dimension.

Příkaz SET FIELD TITLES použijte pro:

- Dynamické lokalizování databáze.
- Ukázání polí způsobem jaký chcete bez závislosti na aktuálním definování tabulky.
- Ukázání polí podle identity a vlastního nastavení uživatele.

UPOZORNĚNÍ: SET TABLE FIELD NEZMĚNÍ vlastnost pole Neviditelné. Pokud je v prostředí návrháře pole označeno jako neviditelné, použití v příkazu SET FIELD TITLES jej nemůže zobrazit.

Příklad

Podívejte se na příklad u příkazu SET TABLE TITLES.

Dále si přečtěte

Count fields, Field name, SET TABLE TITLES.

Shift down (Stisknut shift)

Příkazy a odkazy pro Rozhraní uživatele

Verze 6.0

Shift down → Logické

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Logické	←	Stav klávesy Shift

Popis

Následující metoda objektu pro tlačítko bTlačítko provede rozdílnou akci podle modifikátoru, který je stištěn při klepnutí na tlačítko:

```
` Metoda obejktu bTlačítko
Case of
   ` Může zde být testováno více kombinací kláves
   ` ...
   ÷ (Shift down & Windows Ctrl down)
      ` Stiskuny klávesy Shift a Windows Ctrl (nebo Macintosh Command)
      PROVED AKCI1
      ` ...
   ÷ (Shift down)
      ` Pouze klávesy Shift
      PROVED AKCI2
      ` ...
   ÷ (Windows Ctrl down)
      ` Stisknuto pouze Windows Ctrl (nebo Macintosh Command)
      PROVED AKCI3
      ` ...
      ` Jiné klávesy zde mohou být testovány
      ` ...
End case
```

Dále si přečtěte

Caps lock down, Macintosh command down, Macintosh control down, Macintosh option down, Windows Alt down, Windows Ctrl down.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Caps lock down (Stisknut zámek písmen)

Verze 6.0

Caps lock down → Logické

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Logické	←	Stav klávesy Zámek písmen

Popis

Caps lock down vrací True, pokud je stisknuta klávesa Zámek písmen.

Příklad

Podívejte se na příklad u příkazu Shift down.

Dále si přečtěte

Macintosh command down, Macintosh control down, Macintosh option down, Shift down, Windows Alt down, Windows Ctrl down.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Windows Ctrl down (Stisknut Windows CTRL)

Příkazy a odkazy pro Rozhraní uživatele

Verze 6.0

Windows Ctrl down → Logické

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Logické	←	Stav klávesy Windows Ctrl (Command na Macintosh)

Popis

Windows Ctrl down vrací True pokud je stisknuta klávesa Windows Ctrl.

Poznámka: Poku je použito na Macintoshi, vrací Windows Ctrl down True, když je stiskuta klávesa Command.

Příklad

Podívejte se na příklad u příkazu Shift down.

Dále si přečtěte

Caps lock down, Macintosh command down, Macintosh option down, Shift down, Windows Alt down, Windows Ctrl down.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Windows Alt down (Stisknut Windows Alt)

Příkazy a odkazy pro Rozhraní uživatele

Verze 6.0

Windows Alt down → Logické

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Logické	←	Stav klávesy Windows Alt (Option na Macintoshi)

Popis

Windows Alt down vrací True pokud je stisknuta klávesa Windows Alt.

Poznámka: Při použití tohoto příkazu na Macintoshi, Windows Alt down vrací True pokud je stiskuta klávesa Option.

Příklad

Podívejte se na příklad u příkazu Shift down.

Dále si přečtěte

Caps lock down, Macintosh command down, Macintosh control down, Macintosh option down, Shift down, Windows Ctrl down.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Macintosh command down (Stiskut Macintosh Command)

Verze 6.0

Macintosh command down → Logické

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Logické	←	Stav klávesy Macintosh Command (Ctrl na Windows)

Popis

Macintosh command down vrací True pokud je stisknuta klávesa Macintosh Command.

Poznámka: Pokud je tento příkaz proveden na Windows, vrátí True je stisknuta klávesa Windows Ctrl.

Příklad

Podívejte se na příklad u příkazu Shift down.

Dále si přečtěte

Caps lock down, Macintosh control down, Macintosh option down, Shift down, Windows Alt down, Windows Ctrl down.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Macintosh option down (Stisknut Macintosh Option)

Verze 6.0

Macintosh option down → Logické

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Logické	←	Stav klávesy Macintosh Option (Alt na Windows)

Popis

Macintosh option down vrací True pokud je stisknuta klávesa Macintosh Option.

Poznámka: Pokud je tento příkaz proveden na Windows, vrátí True je stisknuta klávesa Windows Alt.

Příklad

Podívejte se na příklad u příkazu Shift down.

Dále si přečtěte

Caps lock down, Macintosh command down, Macintosh control down, Shift down, Windows Alt down, Windows Ctrl down.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Macintosh control down (Stisknut Macintosh control)

Příkazy a odkazy pro Rozhraní uživatele

Verze 6.0

Macintosh control down → Logické

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Logické	←	Stav kávesy Macintosh control

Popis

Macintosh control down vrací True pokud je stisknuta klávesa Macintosh control.

Poznámka: Pokud je tento příkaz proveden na Windows, vrací vždy False. Tato klávesa nemá protějšek na Windows.

Příklad

Podívejte se na příklad u příkazu Shift down.

Dále si přečtěte

Caps lock down, Macintosh command down, Macintosh option down, Shift down, Windows Alt down, Windows Ctrl down.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GET MOUSE (ZÍSKAT MYŠ)

Verze 6.0

GET MOUSE (myšX; myšY; myšTlačítko{; *})

Parametr	Typ		Popis
myšX	číslo	←	horizontální koordináta myši
myšY	číslo	←	vertikální koordináta myši
myšTlačítko	číslo	←	stav tlačítka myši 0 = tlačítko nahoře (Win levé) 1 = tlačítko dole (Win levé) 2 = pravé talčítko dole (pouze Windows) 3 = obě tlačítka dole (pouze Windows)
*		→	je-li vynechána, používá systém místní souřadnice (okno formuláře) je-li, používá systém globální souřednice (obrazovka)

Popis

Příkat GET MOUSE vrací platný stav myši.

Vodorovné a svislé souřednice jsou vráceny do parametrů *myšX* a *myšY*. Pokud vynecháte parametr *, jsou vyjádřeny vzhledem k oknu na popředí platného procesu. Pokud předáte parametr *, jsou vyjádřeny vzhledem k obrazovce.

Parametr *myšTlačítko* vrací stav tlačítek myši, jak je zobrazeno výše.

Příklad

Podívejte se na příklad u příkazu Pop up menu.

Dále si přečtěte

Caps lock down, Macintosh command down, Macintosh control down, Macintosh option down, ON EVENT CALL, Shift down, Windows Alt down, Windows Ctrl down.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Pop up menu (Rozevírací nabídka)

Příkazy a odkazy pro Rozhraní uživatele

Verze 6.0

Pop up menu (obsah{; výchozí}) → Číslo

Parametr	Typ		Popis
obsah	Text	→	Textová definice nabídky, k zobrazení pod tlačítkem myši
výchozí	číslo	→	Výchozí vybraný prvek nabídky
Výsledek funkce	číslo	←	Číslo vybraného prvku nabídky

Popis

Příkaz Pop up menu zobrazí rozevírací nabídku na platném umístění myši.

Aby jste dodrželi pravidla rozhraní uživatele, budete tento příkaz volat obvykle při stisknutí tlačítka myši a pokud bude nadále drženo.

Položky rozevírací nabídky definujete do parametru obsah následovně:

- Jednotlivé položky oddělte znaménkem středník (;). Například "Pol1;Pol2;Pol3"
- K skrytí položky vložte otevřené závorky (() do textu položky.
- K definování oddělovací čáry vložte "(-" do textu položky
- K definování stylu písma, umístěte do textu položky znaménko menší než (<) následované jedním z těchto znaků:

<B	*Tučně*
<I	*Kurzívou*
<U	*Podtržené*
<O	*Obrysové (Macintosh pouze)*
<S	*Stínované (Macintosh pouze)*

- K předání znaménka označení k položce, vložte do textu znak vykřičník (!) následovaný znakem který chcete jako znaménko označení. Na Macintoshi je znak zobraze; na Windows je zobrazen znak zaškrtnutí, bez ohledu na znak jaký předáte.

- K předání ikony k položce, vložte do textu znak (^) následovaný znakem, jehož ASCII kód mínus 48 je ID MacOS zdroje ikony.

- K předání zkratky k položce, vložte do textu položky lomítko (/) následované znakem zkratky pro položku. Nezapomeňte že tato poslední možnost je pouze informační; žádná zkratka nevyvolá rozevírací nabídku. Nicméně můžete chtít vložit zkratku pokud položka rozevírací nabídky má shodnou položku v záhlaví nabídek aplikace.

Volitelný parametr výchozí vám umožní definovat výchozí označeou položku rozevírací nabídky. Vložte hodnotu mezi 1 a počtšm položek. Pokud tento parametr vynecháte, příkaz jako výchozí označí první položku rozevírací nabídky.

Pokud vyberete položku nabídky, příkaz vrátí její číslo. Jinak vrátí 0.

Poznámka: Použijte rozevírací nabídku která bude obsahovat rozumný počet položek. Pokud chcete použít více než 50 položek, uvažujte spíše o posuvné oblasti ve formuláři.

Příklad

Metoda projektu MA PLOVOUCI NABIDKA rozevře navigační rozevírací nabídku:

```
` Metoda projektu MA PLOVOUCI NABIDKA
```

```
GET MOUSE($vlMouseX;$vlMouseY;$vlButton)
If (Macintosh control down | ($vlButton=2))
   $vtItems:="O této databázi...<I;(-;!-Další vlastnosti;(-"
   For ($vlTabulka;1;Count tables)
      $vtItems:=$vtItems+";"+Table name($vlTabulka)
   End for
   $vlVýběrUživatele:=Pop up menu($vtItems)
   Case of
      ÷ ($vlVýběrUživatele=1)
         ` Zobrazit informace
      ÷ ($vlVýběrUživatele=2)
         ` Zobrazit vlastnosti
   Else
      If ($vlVýběrUživatele>0)
         ` Jít na tabulku jejíž číslo je $vlVýběrUživatele-4
      End if
   End case
End if
```

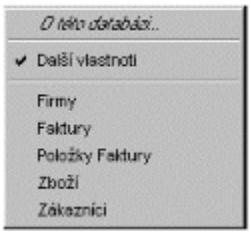
Tato metoda projektu může být volána z:

- Metody objektu formuláře který reaguje na tlačítko myši a nečeká až bude puštěno (tzn. neviditelné tlačítko)
- Procesu který sleduje události a komunikuje s jinými procesy
- Metody na ovládání událostí instalované příkazem ON EVENT CALL.

V posledních dvou případech nemusí být klepnutí provedeno v nějakém objektu formuláře. Je to jedno z rozšíření příkazu Pop up menu. Rozevírací nabídky vždy zobrazujete z objektů formuláře, ale pomocí tohoto příkazu můžete rozevírací nabídku zobrazit kdekoli.

Rozevírací nabídka je na Windows zobrazena pravým tlačítkem myši; na Macintoshi je to Control+Klepnutí. I když tato metoda netestuje klepnutí myši, dělá to volací metoda.

Následující obrázek ukazuje rozevírací nabídku jak se zobrazuje na Windows. Všimněte si standardního znaménka označení ve Windows verzi:



Dále si přečtěte

GET MOUSE.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

POST KEY (POSLAT KLÁVESU)

Příkazy a odkazy pro Rozhraní uživatele

Verze 6.0

POST KEY (kód{; modifikátor{; proces}})

Parametr	Typ		Popis
kód	číslo	→	ASCII kód znaku nebo kód klávesy funkce
modifikátor	číslo	→	stav klíčových modifikátorů
proces	číslo	→	číslo odkazu na cílový proces,
		→	kam se simulace stisku kláves odešle, nebo
		→	fronta událostí aplikace, je-li vynechán, dtto 0

Popis

Příkaz POST KEY simuluje úhoz na klávesu. Jeho efekt je stejný jako kdyby uživatel klepl na klávesu na klávesnici.

Do parametru *kód* předáte ASCII kód znaku.

Pokud předáte parametr *modifikátor*, předáte do něj jednu nebo kombinaci konstant událostí (modifikátorů). Například pro simulaci klávesy Shift můžete použít Shift key mask. Pokud nepředáte modifikátor, nebude simulován žádný modifikátor.

Pokud definujete parametr proces, je úhoz na klávesu poslán do procesu jehož číslo zadáte. Pokud předáte 0 nebo tento parametr vynecháte je úhoz na klávesu poslán na úrovni aplikace a zobrezí se potřičném procesu.

Příklad

Podívejte se na příklad u příkazu Process number.

Dále si přečtěte

POST CLICK, POST EVENT.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

POST CLICK (POSLAT KLEPNUTÍ)

Příkazy a odkazy pro Rozhraní uživatele

Verze 6.0

POST CLICK (myšX; myšY{; proces}{; *})

Parametr	Typ		Popis
myšX	číslo	→	Horizontální koordináta myši
myšY	číslo	→	Vertikální koordináta myši
proces	číslo	→	Číslo odkazu na cílový proces, kam se klepnutí myši odešle, nebo fronta událostí aplikace, je-li vynechán, dtto 0
*		→	je-li vynechána, používá systém místní souřadnice (obrazovka) je-li, používá systém globální souřednice (okno formuláře)

Popis

Příkaz POST CLICK pošle klepnutí na tlačítko myši. Jeho efekt je stejný jako když uživatel klepne na tlačítko myši.

Do parametrů *myšX* a *myšY* předáte vodorovné a svislé souřadnice. Pokud předáte parametr *, napíšete souřadnice vzhledem k oknu na popředí procesu, jehož číslo zadáte. Pokud vynecháte tento parametr, napíšete souřadnice vzhledem k obrazovce (oknu aplikace).

Pokud definujete parametr *proces*, je úhoz na klávesu poslán do procesu jehož číslo zadáte. Pokud předáte 0 nebo tento parametr vynecháte je úhoz na klávesu poslán na úrovni aplikace a zobrazí se v patřičném procesu.

Dále si přečtěte

POST EVENT, POST KEY.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

POST EVENT (POSLAT UDÁLOST)

Příkazy a odkazy pro Rozhraní uživatele

Verze 6.0

POST EVENT (co; zpráva; kdy; myšX; myšY; modifikátor{; proces})

Parametr	Typ		Popis
co	číslo	→	typ události
zpráva	číslo	→	zpráva události
kdy	číslo	→	čas události v tricích
myšX	číslo	→	horizontální souřadnice myši
myšY	číslo	→	vertikální souřadnice myši
modifikátor	číslo	→	stav klíčových modifikátorů
proces	číslo	→	číslo odkazu na cílový proces, kam se má událost odeslat, nebo fronta událostí aplikace, je-li vynechán, dtto 0

Popis

Příkaz POST EVENT simuluje událost klávesy nebo myši. Jeho efekt je stejný jako když uživatel pracuje s klávesnicí nebo myší.

Parametrem *co* předáte jednu z následujících hodnot:

Konstanta	***Typ***	***Hodnota***
Mouse down event	*Long Integer*	*1*
Mouse up event	*Long Integer*	*2*
Key down event	*Long Integer*	*3*
Key up event	*Long Integer*	*4*
Auto key event	*Long Integer*	*5*

Pokud je simulována událost myši, do parametru zpráva předáte 0. Pokud je simulována událost na klávesnici, předáte do zprávy ASCII kód znaku.

Parametrem *kdy* ovykle předáte hodnotu vracenou Tickcount.

Pokud je událost událostí myši, předáte do *myšX* a *myšY* vodorovné a svislé souřednice klepnutí myši. Pokud předáte parametr *, napíšete souřadnice vzhledem k oknu na popředí procesu, jehož číslo zadáte. Pokud vynecháte tento parametr, napíšete souřednice vzhledem k obrazovce.

Do parametru *modifikátor*, předáte do jeden nebo kombinaci konstant událostí (modifikátorů). Například pro simulaci klávesy Shift můžete použít Shift key bit.

Pokud definujete parametr proces, je událost poslána do procesu jehož číslo zadáte. Pokud předáte 0 nebo tento parametr vynecháte je úhoz na klávesu poslán na úrovni aplikace a objeví se v patřičném procesu.

Dále si přečtěte

POST CLICK, POST KEY.

GET HIGHLIGHT (ZÍSKAT ZVÝRAZNĚNÉ)

Příkazy a odkazy pro Rozhraní uživatele

Verze 3

GET HIGHLIGHT (oblast; počvyb; konecvyb)

Parametr	Typ		Popis
oblast	pole | proměnná	→	dostupné pole nebo proměnná
počvyb	číslo	←	zjištěná počáteční pozice vybraného textu
konecvyb	číslo	←	zjištěná konečná pozice vybraného textu

Popis

Příkaz GET HIGHLIGHT se používá k určení toho, jaký text je označen.

Upozornění: I když předáte do GET HIGHLIGHT pole nebo dostupnou proměnnou, tento příkaz vrací pozici výběru pouze když je použit na právě upravovanou oblast.

Poznámka: Tento příkaz nemůže být použit na pole ve výstupním formuláři nebo podformuláři.

Text může být označen příkazem HIGHLIGHT TEXT nebo uživatelem.

Parametr *počvyb* vrací pozici prvního označeného znaku. Parametr *konecvyb* vrací pozici posledního označeného znaku plus jedna.

Pokud jsou *počvyb* a *konecvyb* vráceny stejné, je kurzor umístěn před znakem definovaným *počvyb*. Uživatel nemá označený žádný text a nejsou zvýrazněny žádné znaky.

Příklad

1. Následující příklad vezme zvýrazněný výběr z pole [Produkty]Komentář:

```
GET HIGHLIGHT ([Produkty]Komentář;vFirst;vLast)
If (vFirst<vLast)
   ALERT("Vybraný text je: "+Substring([Produkty]Komentář;vFirst;vLast-vFirst))
End if
```

2. Podívejte se na příklad u příkazu FILTER KEYSTROKE.

Dále si přečtěte

FILTER KEYSTROKE, HIGHLIGHT TEXT, Keystroke.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

HIGHLIGHT TEXT (ZVÝRAZNIT TEXT)

Příkazy a odkazy pro Rozhraní uživatele

Verze 3

HIGHLIGHT TEXT (oblast; počvyb; konecvyb)

Parametr	Typ		Popis
oblast	pole | proměnná	→	dostupné pole nebo proměnná
počvyb	číslo	→	počáteční pozice textu k vybrání
konecvyb	číslo	→	konečná pozice textu k vybrání

Popis

Příkaz HIGHLIGHT TEXT zvýrazní výběr v textové oblasti.

Jestliže oblast není pravě upravovaný objekt, focus je přusunut na tuto oblast.

Poznámka: Tento příkaz nemůže být použit na pole ve výstupním formuláři nebo podformuláři.

Parametr počvyb je pozice prvního označeného znaku. Parametr konecvyb je pozice posledního označeného znaku plus jedna. Pokud jsou počvyb a konecvyb jsou stejné, je kurzor umístěn před znakem definovaným počvyba není označený žádný text.

Pokud je konecvyb větší než počet znaků v oblasti, pak jsou označeny všechny znaky od počvyb do konce textu.

Příklad

1. Následující příklad vybere všechy znaky dostupného pole [Produkty]Komentář:

```
HIGHLIGHT TEXT([Produkty]Komentář;1;Length([Produkty]Komentář)+1)
```

2. Následující příklad posune kurzor na začátek textu pole [Produkty]Komentář:

```
HIGHLIGHT TEXT([Produkty]Komentář;1;1)
```

3. Následující příklad posune kurzor na konec testu pole [Produkty]Komentář:

```
$vLen:=Length([Produkty]Komentář)+1
HIGHLIGHT TEXT([Produkty]Komentář;$vLen;$vLen)
```

4. Podívejte se na příklad u příkazu FILTER KEYSTROKE.

Dále si přečtěte

GET HIGHLIGHT.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET CURSOR (NASTAVIT KURZOR)

Verze 6.0

SET CURSOR ({kurzor})

Parametr	Typ		Popis
kurzor	Číslo	→	Číslo MacOS zdroje kurzoru

Popis

Příkaz SET CURSOR nastaví kuzor myši na MacOS zdroj `CURS` jehož ID předáte jako parametr.

Pokud vybecháte parametr, bude kurzor nastaven na výchozí šipku.

Použijte příkaz RESOURCE LIST pro zobrazení seznamu dostupných kurzotů.

Dále si přečtěte

RESOURCE LIST.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Last object (Poslední objekt)

Příkazy a odkazy pro Rozhraní uživatele

Verze 3

Last object → Ukazatel

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Ukazatel	←	Ukazatel na poslední nebo platnou dostupnou oblast

Popis

Last object vrací ukazatel na poslední nebo platné dostupné oblasti. Jinými slovy objekt ve kterém je kuzor nebo jej právě opustil. Můžete použít Last object k provedení akce v oblasti formuláři bez toho, aby jste věděli který objekt je označený. Před provedením akce se pomocí příkazu Type ujistěte, že objekt je správného typu. Tento příkaz nemůže být použit na pole v podformuláři.

Poznámka: Příkaz je určen pro použití ve vstupu dat, jinak bude vracet chyby.

Příklad

Následující příklad je metoda objektu pro tlačítko. Metoda objektu mění data v platném výběru na velká písmena. Objektu musí být typu text nebo řetězec (typ 0 nebo 24):

```
$vp := Last object ` Uložit ukazatel na poslední oblasti
If ((Type ($vp→) = Is Alpha field) | (Type($vp→) = Is String var))
      ` JE-LI TO ŘETĚZEC NEBO TEXT
   $vp→ := Uppercase ($vp→) ` Změnit oblast na velká písmena
End if
```

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

REDRAW (PŘEKRESLIT)

Příkazy a odkazy pro Rozhraní uživatele

Verze 3

REDRAW (objekt)

Parametr	Typ		Popis
objekt	Objekt	→	podtabulka, pro kterou se překreslí předány formulář, nebo Tabulka, pro kterou se překreslí předány formulář, nebo Pole, pro které se překreslí oblast, nebo Proměnná, pro kterou se překreslí oblast, nebo Formulář, který se překreslí ve Web prohlížeči při časování

Popis

Pokud použijete metodu k upravení hodnoty pole nebo podpole zobrazeného ve formuláři, musíte použít příkaz REDRAW aby jste měli jistotu že bude formulář obnoven.

Web Server: Při použití po události formuláře Při časování (On Timer), REDRAW může být použit k opakovanému obnovení formuláře 4D poslaného na Web prohlížeč. Jestli chcete vědět více informací, přečtěte si příkaz SET TIMER.

Dále si přečtěte

SET TIMER.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

INVERT BACKGROUND (INVERTOVAT POZADÍ)

Verze 3

INVERT BACKGROUND ({*;} textProm | textPole)

Parametr	Typ		Popis
*		→	dovolí vstup proměnné nebo název objektu
textProm | textPole	proměnná | pole	→	textová proměnná nebo textové pole k inverzi

Popis

Příkaz INVERT BACKGROUND je použit k inverzi textProm nebo textPole.

Rozsah příkazu je používaný formulář.

INVERT BACKGROUND můžete použít k zobrazení na obrazovku nebo k tisku na bodovou maticovou tiskárnu. Postscriptová tiskárna nebude tisknou invertované pozadí.

Příklad

Tento příklad je metoda objektu pro proměnnou ve vstupním formuláři. Testuje hodnotu pole. Pokud je pole kladné, metoda neprovede nic. Pokud je pole záporné, metoda objektu invertuje zobrazení objektu ve formuláři:

```
vAmount:=[Accounts]Amount ` Vložit hodnotu pole do proměnné
If (vAmount < 0) ` Pokud je pole záporné…
   INVERT BACKGROUND (vAmount) ` Invertovat pozadí
End if
```

Poznámka: Tento příkaz, původně vytvořen pro černobílé rozhraní uživatele, je nyní používán zřídka. Nyní používáte spíše barvy k zvýraznění pole nebo proměnné.

Dále si přečtěte

SET COLOR, SET RGB COLORS.

Příkazy a odkazy pro Rozhraní uživatele

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

EDIT ACCESS (UPRAVIT PŘÍSTUP)

Příkazy a odkazy pro Uživatelé a skupiny

Verze 3

EDIT ACCESS

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz EDIT ACCESS umožní uživateli upravit systém hesel. K upravení přístupu se používá okno Hesla z prostředí návrháře.

Skupiny mohou být uporavovány Návrhářem, Administrátorem a vlastníky skupin. Návrhář a Administrátor mohou upravovat všechny skupiny. Vlastník skupiny může upravovat svojí skupinu. Jednotlivý uživatelé mohou být přidání a odstraněni ze skupiny. Příkaz neprovede nic, pokud nejsou definovány žádné skupiny.

Návrhář a Administrátor mohou přidávat nové uživatele a stejně tak je přiřadit k nějaké skupině.

Příklad

Následující příklad zobrazí okno Hesla:

```
EDIT ACCESS
```

Dále si přečtěte

CHANGE ACCESS, CHANGE PASSWORD.

Příkazy a odkazy pro Uživatelé a skupiny

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CHANGE ACCESS (ZMĚNIT PŘÍSTUP)

Verze 3

CHANGE ACCESS

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz CHANGE ACCESS umožní uživateli změnit úroveň přístupu bez opuštění databáze. Je zobrazeno stejné okno jako když uživatel spouští databázi a může vybrat vstup pod jiným uživatelem.

Zobrazené okno bude závislé na nastavení v Předvolbách databáze v prostředí návrháře.

Příklad

Následující příklad zobrazí okno hesel pro vstup:

```
CHANGE ACCESS
```

Dále si přečtěte

CHANGE PASSWORD.

Příkazy a odkazy pro Uživatelé a skupiny

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CHANGE PASSWORD (ZMĚNIT HESLO)

Příkazy a odkazy pro Uživatelé a skupiny

Verze 3

CHANGE PASSWORD (heslo)

Parametr	Typ		Popis
heslo	Řetězec	→	Nové heslo

Popis

Příkaz CHANGE PASSWORD změní heslo platného uživatele. Tento příkaz nahradí staré heslo novým heslem zadaným do parametru heslo.

Upozornění: Heslo je citlivé na malé a velké znaky.

Příklad

Následující příklad umožní uživateli změnit heslo:

```
CHANGE ACCESS ` Zobrazí uživateli okno hesel pro vstup
If (OK=1)
   $pw1:=Request("Zadejte nové heslo pro "+Current user)
      ` Heslo by mělo mít alespoň pět znaků
   If (((OK=1) & ($pw1#"")) & (Length($pw1)>5))
         ` Ujistěte se, že heslo bylo zadáno správně
      $pw2:=Request("Zadejte heslo znovu")
      If ((OK=1) & ($pw1=$pw2))
         CHANGE PASSWORD($pw2) ` Změnit heslo
      End if
   End if
End if
```

Dále si přečtěte

CHANGE ACCESS.

Příkazy a odkazy pro Uživatelé a skupiny

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Validate password (Potvrdit heslo)

Příkazy a odkazy pro Uživatelé a skupiny

Verze 6.0.2

Validate password (uživID; heslo) → Logické

Parametr	Typ		Popis
uživID	číslo	→	jedinečné číslo ID uživatele
heslo	řetězec	→	heslo v nekódované formě
Výsledek funkce	Logické	←	True = platné heslo False = neplatné heslo

Popis

Validate password vrací True, pokud heslo zadané do heslo je heslo uživatele zadaného do uživID.

Příklad

Tento příklad testuje jestli heslo uživatele "Hardy" je "Laurel":

```
GET USER LIST(atUserName;alUserID)
$vlElem:=Find in array(atUserName;"Hardy")
If ($vlElem>0)
   If (Validate password(alUserID{$vlElem};"Laurel")>0)
      ALERT("Správně!")
   Else
      ALERT("Špatně!")
   End if
Else
   ALERT("Neznámý uživatel")
End if
```

Dále si přečtěte

GET USER PROPERTIES , SET USER PROPERTIES.

Příkazy a odkazy pro Uživatelé a skupiny

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Current user (Platný uživatel)

Verze 3

Current user → Řetězec

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Řetězec	←	Jméno platného uživatele

Popis

Current user vrací jméno platného uživatele.

Příklad

Podívejte se na příklad u příkazu User in group.

Dále si přečtěte

CHANGE ACCESS, CHANGE PASSWORD, User in group.

Příkazy a odkazy pro Uživatelé a skupiny

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

User in group (Uživatel ve skupině)

Příkazy a odkazy pro Uživatelé a skupiny

Verze 3

User in Group (uživatel; skuipna) → Logické

Parametr	Typ		Popis
uživatel	Řetězec	→	Jméno uživatele
skupina	Řetězec	→	Název skupiny
Výsledek funkce	Logické	←	TRUE = uživatel je ve skupině FALSE = uživatel není ve skupině

Popis

User in group vrací True pokud je uživatel ve skupině.

Příklad

Následující příklad vyhledá určiré faktury. Pokud je platný uživatel ve skupině Vedení, je oprávněn zobrazit faktury ve formuláři obsahujícím důvěrné informace. Pokud uživatel v této skupině není, je zobrazen jiný formulář:

```
QUERY([Faktury];[Faktury]Cena >10000)
If (User in group(Current user;"Vedení"))
   OUTPUT FORM([Faktury];"Vedení Výstup")
   INPUT FORM([Faktury];"Vedení Vstup")
Else
   OUTPUT FORM([Faktury];"Standard Output")
   INPUT FORM([Faktury];"Standard Vstup")
End if
MODIFY SELECTION([Faktury];*)
```

Dále si přečtěte

Current user.

Příkazy a odkazy pro Uživatelé a skupiny

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DELETE USER (VYMAZAT UŽIVATELE)

Příkazy a odkazy pro Uživatelé a skupiny

Verze 6.0

DELETE USER (uživID)

Parametr	Typ		Popis
uživID	Číslo	→	ID číslo uživatele k vymazání

Popis

Příkaz DELETE USER vymaže uživatele, jehož jedinečné ID předáte do uživID. Musíte zadat platné ID uživatele vrácené příkazem GET USER LIST.

Pokud uživatel neexistuje nebo byl již vymazán, je vrácena chyba -9979. Tuto chybu můžete zachytit pomocí metody instalované příkazem ON ERR CALL.

Jméno vymazaného uživatele se nebude objevovat v okně hesel při vstupu do databáze nebo při provedení příkazu CHANGE ACCESS. Nicméně pro zachování jedinečnosti ID uživatelů, je vymazaný uživatel ponechán v systému hesel. Jména vymazaných uživatelů se budou objevovat zeleně v okně Hesla v prostředí návrháře.

Dále si přečtěte

GET USER LIST, GET USER PROPERTIES, Is user deleted, SET USER PROPERTIES.

Ovládání chyb

Pokud nemáte práva na provedení příkazu DELETE USER nebo jsou již hesla upravována jiným uživatelem, je zobrazena chyba přístupu. Tuto chybu můžete zachytit pomocí Metody na ovládání chyb instalované příkazem ON ERR CALL.

Příkazy a odkazy pro Uživatelé a skupiny

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Is user deleted (Je uživatel vymazán)

Příkazy a odkazy pro Uživatelé a skupiny

Verze 6.0

Is user deleted (uživatelČíslo) → Logické

Parametr	Typ		Popis
uživatelČíslo	Číslo	→	ID číslo uživatele
Výsledek funkce	Logické	←	TRUE = účet uživatele byl vymazán a neexistuje FALSE = účet uživatele je aktivní

Popis

Příkaz Is user deleted testuje účet uživatele podle jedinečného ID které zadáte do parametru uživatelČíslo.

Pokud účet uživatele neexistuje nebo byl již vymazán, příkaz vrátí True. Jinak vrací False.

Dále si přečtěte

DELETE USER, GET USER PROPERTIES, SET USER PROPERTIES.

Ovládání chyb

Pokud nemáte práva na provedení příkazu Is user deleted nebo jsou již hesla upravována jiným uživatelem, je zobrazena chyba přístupu. Tuto chybu můžete zachytit pomocí Metody na ovládání chyb instalované příkazem ON ERR CALL.

Příkazy a odkazy pro Uživatelé a skupiny

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GET USER LIST (ZÍSKAT SEZNAM UŽIVATELŮ)

Verze 6.0

GET USER LIST (JménaUživatelů; ČíslaUživatelů)

Parametr	Typ		Popis
JménaUživatelů	String Array	←	Jména uživatelů jak se objeví v okně editoru hesel a přístupu
ČíslaUživatelů	Integer Array	←	Odpovídající jedinečné číslo ID uživatele

Popis

GET USER LIST naplní jména a jedinečná ID uživatelů do array *JménaUživatelů* a *ČíslaUživatelů*.

Array *JménaUživatelů* obsahuje jména uživatelů jak se objevují v okně hesel a to včetně uživatelů jejichž účet byl skryt (jména uživatelů zobrazená zeleně v okně Hesla).

Poznámka: Použijte příkaz Is user deleted k detekování uživatelů.

Array *ČísloUživatelů*, synchronizované s *JménaUživatelů*, obsahuje odpovídající jedinečná ID uživatelů. Tato čísla mohou mít následující hodnoty nebo rozsah:

ID uživatele	***Popis uživatele***
1	*Návrhář*
2	*Administrátor*
3 až 15000	*Uživatelé vytvoření návrhářem databáze (uživatel #3 je první vytvořený, uživatel #4 je druhý, atd.)*
-11 až -15000	*Uživatelé vytvoření administrátorem databáze (uživatel #-11 je první vytvořený, uživatel #-12 je druhý, atd.)*

Dále si přečtěte

GET GROUP LIST, GET USER PROPERTIES, SET USER PROPERTIES.

Ovládání chyb

Pokud nemáte práva na provedení příkazu GET USER LIST nebo jsou již hesla upravována jiným uživatelem, je zobrazena chyba přístupu. Tuto chybu můžete zachytit pomocí Metody na ovládání chyb instalované příkazem ON ERR CALL.

GET USER PROPERTIES (ZÍSKAT VLASTNOSTI UŽIVATELE)

Příkazy a odkazy pro Uživatelé a skupiny

Verze 6.0

GET USER PROPERTIES (uživID; jméno; startup; heslo; početPřihl; poslPřihl{; členem})

Parametr	Typ		Popis
uživID	číslo	→	jedinečné číslo ID uživatele
jméno	řetězec	←	Jméno / název uživatele
startup	řetězec	←	název metody při spuštění
heslo	řetězec	←	vždy prázdný řetězec
početPřihl	číslo	←	počet přihlášení do systému
poslPřihl	datum	←	Datum posledního přihlášení do systému
členem	Integer Array	←	čísla ID skupin, do kerých uživatel patří

Popis

GET USER PROPERTIES vrací informace o uživateli jehož ID zadáte do uživID. Musíte vložit platné ID uživatele vrácené příkazem GET USER LIST.

Pokud účet uživatele neexistuje nebo již byl vymazán, je vrácena chyba -9979. Tuto chybu můžete zachytit pomocí metody instalované příkazem ON ERR CALL. Jinak můžete před GET USER PROPERTIES provést Is user deleted k testování jestli účet uživatele existuje.

ID uživatele mohou mít následující hodnoty nebo rozsah:

ID uživatele	***Popis uživatele***
1	*Návrhář*
2	*Administrátor*
3 až 15000	*Uživatelé vytvoření návrhářem databáze (uživatel #3 je první vytvořený, uživatel #4 je druhý, atd.)*
-11 až -15000	*Uživatelé vytvoření administrátorem databáze (uživatel #-11 je první vytvořený, uživatel #-12 je druhý, atd.)*

Po provedení příkazu obdržíte jméno, metodu spuštění, kódované heslo, počet připojení a datum posledního připojení uživatele do parametrů jméno, startup, heslo, početPřihl a poslPřihl.

Poznámka: GET USER PROPERTIES nevrací heslo do parametru heslo. Od verze 6.0.2 je vždy vracen prázdný řetězec. *Příručka jazyka 4th Dimension* dodávaná s verzí 6.0.2 nepopisuje tuto změnu.

Pokud předáte volitelný parametr členem, jsou do něj vrácena jedinečná ID skupin ke kterým uživatel patří. ID skupin mohou být v následujícím rozsahu:

ID skupiny	***Popis skupiny***
15001 až 32767	*Skupina vytvořená Návrhářem nebo přičleněným Vlastníkem skupiny (skupina #15001 je první vytvořená, skupina #15002 je vytvořena druhá, atd.)*
15001 až 32767	*Skupina vytvořená Administrátorem nebo přičleněným Vlastníkem skupiny (skupina #15001 je první vytvořená, skupina #15002 je vytvořena druhá, atd.)*

Dále si přečtěte

GET GROUP LIST, GET USER LIST, SET USER PROPERTIES, Validate password (6.0.2).

Ovládání chyb

Pokud nemáte práva na provedení příkazu GET USER PROPERTIES nebo jsou již hesla upravována jiným uživatelem, je zobrazena chyba přístupu. Tuto chybu můžete zachytit pomocí Metody na ovládání chyb instalované příkazem ON ERR CALL.

Příkazy a odkazy pro Uživatelé a skupiny

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Set user properties (Nastavit vlastnosti uživatele)

Příkazy a odkazy pro Uživatelé a skupiny

Verze 6.0.2

Set user properties (uživID; jméno; startup; heslo; početPřihl; poslPřihl{; členem}) → Číslo

Parametr	Typ		Popis
uživID	číslo	→	jedinečné číslo ID účtu uživatele, nebo pro nové -1 pro přidání uživatele spojeného s Návrhářem, nebo -2 pro přidání uživatele spojeného s Administrátorem
jméno	řetězec	→	Nové jméno / název uživatele
startup	řetězec	→	název metody Startup (Při připojení)
heslo	řetězec	→	nové nekódované heslo, nebo * k ponechání hesla původního
početPřihl	číslo	→	nově nastavený počet přihlášení do systému
poslPřihl	datum	→	nově nastavené datum posledního přihlášení do systému
členem	Integer Array	→	čísla ID skupin, do kerých uživatel patří
Výsledek funkce	číslo	←	jedinečné číslo ID nového uživatele

Popis

Příkaz Set user properties vám umožní změnit a obnovit vlastnosti účtu existujícího uživatele, jehož ID předáte do parametru *uživID* nebo k předání nového uživatele přiřazeného Návrháři nebo Administrátorovi.

Pokud měníte vlastnosti existujícího uživatele, musíte vložit platné ID uživatele vrácené příkazem GET USER LIST.

Pokud účet uživatele neexistuje nebo již byl vymazán, je vrácena chyba -9979. Tuto chybu můžete zachytit pomocí metody instalované příkazem ON ERR CALL. Jinak můžete před Set user properties provést Is user deleted k testování jestli účet uživatele existuje.

ID uživatele mohou mít následující hodnoty nebo rozsah:

ID uživatele	***Popis uživatele***
1	*Návrhář*
2	*Administrátor*
3 až 15000	*Uživatelé vytvoření návrhářem databáze* *(uživatel #3 je první vytvořený, uživatel #4 je druhý, atd.)*
-11 až -15000	*Uživatelé vytvoření administrátorem databáze* *(uživatel #-11 je první vytvořený, uživatel #-12 je druhý, atd.)*

K přidání nového uživatele připojeného k Návrháři, vložte do *uživID* -1 a k vytvoření uživatele připojeného k Administrátorovi vložte do *uživID* -2. Po provedení příkazu je přidán nový uživatel a jeho jedinečné ID je vráceno do *uživID*.

Pokud nepředáte do *uživID* -1, -2 nebo plané ID uživatele, příkaz neprovede nic.

Před provedením příkazu předáte nové jméno, metodu spuštění, heslo, počet připojení a datum posledního připojení do parametrů jméno, *startup, heslo, početPřihl a poslPřihl*. Do parametru heslo předáte nezakódované heslo. 4D jej zakóduje před předáním do účtu uživatele.

Pokud nové jméno uživatele není jedinečné (již existuje uživatel s tímto jménem), příkaz neprovede nic a vrátí chybu -9979. Tuto chybu můžete zachytit pomocí metody instalované příkazem ON ERR CALL.

Pokud nechcete měnit všechny vlastnosti uživatele, proveďte nejdříve GET USER PROPERTIES a vložte vrácené hodnoty místo parametrů které nechcete měnit.

Pokud nechcete měnit *heslo* uživatele, vložte * jako hodnotu parametru *heslo*. Toto vám umožní změnit ostatní vlastnosti mimo hesla.

Pokud nepředáte volitelný parametr *členem*, zůstane nadále uživatel členem všech původních skupin. Pokud nepoužijete tento parametr při zakládání nového uživatele, uživatel nebude členem žádné skupiny.

Pokud použijete volitelný parametr *členem*, změníte všechna členství uživatele. Před provedením příkazu musíte naplnit array členem, ID skupin ke kterým bude uživatel patřit. ID skupin mohou být v následujícím rozsahu:

ID skupiny	***Popis skupiny***
15001 až 32767	*Skupina vytvořená Návrhářem nebo přičleněným Vlastníkem skupiny (skupina #15001 je první vytvořená, skupina #15002 je vytvořena druhá, atd.)*
15001 až 32767	*Skupina vytvořená Administrátorem nebo přičleněným Vlastníkem skupiny (skupina #15001 je první vytvořená, skupina #15002 je vytvořena druhá, atd.)*

K odstranění uživatele ze všech skupin, vložte prázdný array členem.

Dále si přečtěte

DELETE USER, GET GROUP LIST, GET USER LIST, GET USER PROPERTIES, Is user deleted, Validate password.

Ovládání chyb

Pokud nemáte práva na provedení příkazu Set user properties nebo jsou již hesla upravována jiným uživatelem, je zobrazena chyba přístupu. Tuto chybu můžete zachytit pomocí Metody na ovládání chyb instalované příkazem ON ERR CALL.

Příkazy a odkazy pro Uživatelé a skupiny

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GET GROUP LIST (ZÍSKAT SEZNAM SKUPIN)

Verze 6.0

GET GROUP LIST (NázvySkupin; číslaSkupin)

Parametr	Typ		Popis
NázvySkupin	String Array	←	Názvy skupin jak se objeví v okně editoru hesel a přístupu
číslaSkupin	Integer Array	←	Odpovídající jedinečné číslo ID skupin

Popis

GET GROUP LIST naplní array *NázvySkupin* a *číslaSkupin* názvy a jedinečnými ID skupin jak se objevují v okně Editoru hesel v prostředí návrháře.

Array *NázvySkupin* je naplněn názvy všech skupin které jsou v okně Editor hesel a to včetně skrytých skupin (názvy skupin které se v Editoru hesel objevují červeně).

Poznámka: Použijte příkaz GET GROUP PROPERTIES k detekování skupiny která je skrytá.

Array *číslaSkupin*, synchonizované s *NázvySkupin*, je naplněn odpovídajícímy čísli ID skupiny. Tato čísla mohou být v následujícím rozsahu:

ID skupiny	***Popis skupiny***
15001 až 32767	*Skupina vytvořená Návrhářem nebo přičleněným Vlastníkem skupiny (skupina #15001 je první vytvořená, skupina #15002 je vytvořena druhá, atd.)*
15001 až 32767	*Skupina vytvořená Administrátorem nebo přičleněným Vlastníkem skupiny (skupina #15001 je první vytvořená, skupina #15002 je vytvořena druhá, atd.)*

Dále si přečtěte

GET GROUP PROPERTIES, GET USER LIST, SET GROUP PROPERTIES.

Ovládání chyb

Pokud nemáte práva na provedení příkazu GET GROUP LIST nebo jsou již hesla upravována jiným uživatelem, je zobrazena chyba přístupu. Tuto chybu můžete zachytit pomocí Metody na ovládání chyb instalované příkazem ON ERR CALL.

Příkazy a odkazy pro Uživatelé a skupiny

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GET GROUP PROPERTIES (ZÍSKAT VLASTNOSTI SKUPINY)

Příkazy a odkazy pro Uživatelé a skupiny

Verze 6.0

GET GROUP PROPERTIES (skupinaID; název; vlastník{; členi})

Parametr	Typ		Popis
skupinaID	číslo	→	jedinečné číslo ID skupiny
název	řetězec	←	Název skupiny
vlastník	číslo	←	číslo ID uživatele, který vlastní skupinu
členi	číselné Array	←	array s čísly ID jednotlivých členů skupiny

Popis

GET GROUP PROPERTIES vrací vlastnosti skupiny jejíž jedinečné ID zadáte do parametru skupinaID. Musíte vložit platné ID vrácené příkazem GET GROUP LIST. ID skupiny mohou být v následujícím rozsahu:

ID skupiny	***Popis skupiny***
15001 až 32767	*Skupina vytvořená Návrhářem nebo přičleněným Vlastníkem skupiny (skupina #15001 je první vytvořená, skupina #15002 je vytvořena druhá, atd.)*
15001 až 32767	*Skupina vytvořená Administrátorem nebo přičleněným Vlastníkem skupiny (skupina #15001 je první vytvořená, skupina #15002 je vytvořena druhá, atd.)*

Pokud nepředáte správné ID skupiny, příkaz vrátí prázdné parametry.

Po provedení příkazu obdržíte název a vlastníka skupiny do parametrů název a vlastník.

Pokud předáte volitelný parametr členi, budou vrácena ID uživatelů a skupin které patří k této skupině. ID členů mohou být následující:

ID člena	***Popis člena***
1	*Návrhář*
2	*Administrátor*
3 až 15000	*Uživatelé vytvoření návrhářem databáze (uživatel #3 je první vytvořený, uživatel #4 je druhý, atd.)*
-11 až -15000	*Uživatelé vytvoření administrátorem databáze (uživatel #-11 je první vytvořený, uživatel #-12 je druhý, atd.)*
15001 až 32767	*Skupina vytvořená Návrhářem nebo přičleněným Vlastníkem skupiny (skupina #15001 je první vytvořená, skupina #15002 je vytvořena druhá, atd.)*
15001 až 32767	*Skupina vytvořená Administrátorem nebo přičleněným Vlastníkem skupiny (skupina #15001 je první vytvořená, skupina #15002 je vytvořena druhá, atd.)*

Dále si přečtěte

GET GROUP LIST, GET USER LIST, SET GROUP PROPERTIES.

Ovládání chyb

Pokud nemáte práva na provedení příkazu GET GROUP PROPERTIES nebo jsou již hesla upravována jiným uživatelem, je zobrazena chyba přístupu. Tuto chybu můžete zachytit pomocí Metody na ovládání chyb instalované příkazem ON ERR CALL.

Příkazy a odkazy pro Uživatelé a skupiny

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Set group properties (Nastavit vlastnosti skupiny)

Příkazy a odkazy pro Uživatelé a skupiny

Verze 6.0

Set group properties (skupinaID; název; vlastník{; členi}) → Číslo

Parametr	Typ	Popis	
skupinaID	číslo	→	jedinečné číslo ID skupiny, nebo pro nové -1 pro přidání skupiny spojené s Návrhářem, nebo -2 pro přidání skupiny spojené s Administrátorem
název	řetězec	→	Název skupiny
vlastník	číslo	→	číslo ID uživatele, který bude vlastnit skupinu
členi	číselné Array	→	array s čísly ID jednotlivých členů skupiny
Výsledek funkce	číslo	←	jedinečné číslo ID nově založené skupiny

Popis

Set group properties vám umožní nastavit a obnovit vlastnosti existující skupiny jejíž ID zadáte do parametru *skupinaID* nebo vytvořit novou skupinu přiřazenou k Návrháři nebo Administrátorovi.

Pokud měníte vlastnosti existující skupiny, musíte vložit platné ID vrácené příkazem GET GROUP LIST. ID skupiny mohou být v následujícím rozsahu:

ID skupiny	***Popis skupiny***
15001 až 32767	*Skupina vytvořená Návrhářem nebo přičleněným Vlastníkem skupiny (skupina #15001 je první vytvořená, skupina #15002 je vytvořena druhá, atd.)*
15001 až 32767	*Skupina vytvořená Administrátorem nebo přičleněným Vlastníkem skupiny (skupina #15001 je první vytvořená, skupina #15002 je vytvořena druhá, atd.)*

K předání nové skupiny přiřazené k Návrháři, vložte -1 do *skupinaID*. K přidání skupiny přiřazené k Administrátorovi vložte -2 do *skupinaID*. V obou případech, při vkládání nové skupiny se 4th Dimension pokusí znovu použít první dostupný skrytý účet; vytvoří nový účet pouze v případě, že není žádný skrytý. Po provedení příkazu je skupina předána a ID skupiny je vráceno do parametru skupinaID.

Pokud nepředáte -1, -2 nebo správné ID skupiny, příkaz neproveden nic.

Před provedením příkazu předáte nový název a vlastníka skupiny do paramatrů název a vlastník. Pokud nechcete změnit všechny vlastnosti skupiny (mimo členi, čtěte dále), nejdříve proveďte příkaz GET GROUP PROPERTIES a vložte vrácené hodnoty jako parametry které nechcete měnit.

Pokud nepředáte parametr *členi*, zůstane seznam členů skupiny nezměněný. Pokud nepředáte parametr členi při vytváření nové skupiny, nebude mít skupina žádné členy.

Poznámka: Vlastník skupiny není automaticky nastaven jako člen skupiny kterou vlastní. Je pouze na vás jestli chcete vložit vlastníka skupiny jako člena skupiny.

Pokud předáte volitelný parametr *členi*, změníte celý seznam členů skupiny. Před provedením příkazu musíte naplnit array členi s ID uživatelů a skupin kteří budou členy skupiny. ID členů mohou být v následujícím rozsahu:

ID člena	***Popis člena***

1	*Návrhář*
2	*Administrátor*
3 až 15000	*Uživatelé vytvoření návrhářem databáze* *(uživatel #3 je první vytvořený, uživatel #4 je druhý, atd.)*
-11 až -15000	*Uživatelé vytvoření administrátorem databáze* *(uživatel #-11 je první vytvořený, uživatel #-12 je druhý, atd.)*
15001 až 32767	*Skupina vytvořená Návrhářem nebo přičleněným Vlastníkem skupiny* *(skupina #15001 je první vytvořená,* *skupina #15002 je vytvořena druhá, atd.)*
15001 až 32767	*Skupina vytvořená Administrátorem nebo přičleněným* *Vlastníkem skupiny* *(skupina #15001 je první vytvořená,* *skupina #15002 je vytvořena druhá, atd.)*

K ostranění všech členů skupiny, vložte prázdný array *členi*.

Dále si přečtěte

GET GROUP LIST, GET GROUP PROPERTIES, GET USER LIST.

Ovládání chyb

Pokud nemáte práva na provedení příkazu Set group properties nebo jsou již hesla upravována jiným uživatelem, je zobrazena chyba přístupu. Tuto chybu můžete zachytit pomocí Metody na ovládání chyb instalované příkazem ON ERR CALL.

Příkazy a odkazy pro Uživatelé a skupiny

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CHANGE WEB LICENSE (ZMĚNIT WEB LICENCI)

Příkazy a odkazy pro Uživatelé a skupiny

Verze 6.0.2

CHANGE WEB LICENSE

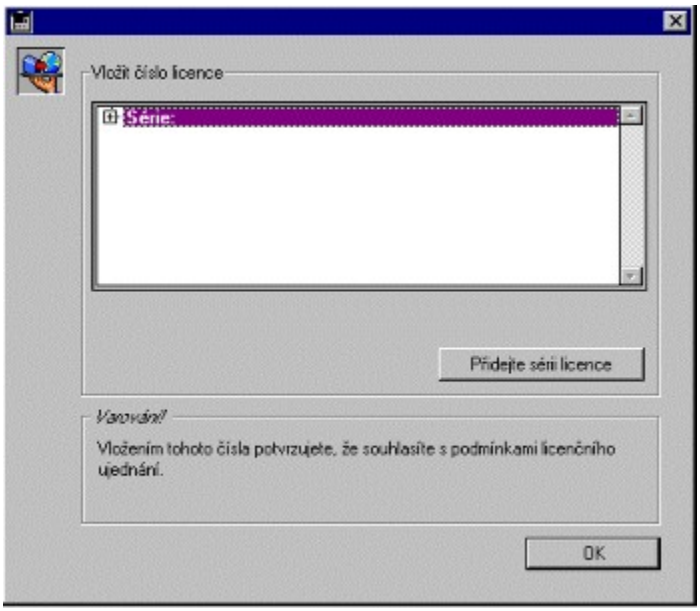
Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz CHANGE WEB LICENSE zobrazí dialogové okno ACI licence, které umožní uživateli vložit ACI Expansion (pouze 4D Server) nebo Sériovou licenci.

Dialogové okno ACI licence:



Poznámka: Jestli chcete vědět více informací, přečtěte si *Instalační příručku*.

V prostředí návrháře zobrazíte toto okno klepnutím na tlačítko Licence v okně Předvolby databáze. S použitm příkazu CHANGE WEB LICENSE můžete zobrazit okno licence i z prostředí uživatele a vlastní nabídky.

CHANGE WEB LICENSE je vhodný způsob jak umožnit změnit licenci ve zkompilované a spojené aplikaci distribudované u zákazníků. Vývojáří 4D a IS manažeři mohou použít tento příkaz k distribuování aplikací 4D a umožnit uživatelům zadat jejich licenci bez posílání obnovené aplikace při každé příležitosti.

Příklad

Ve vlastním nastavení nebo předvolbách dialogového okna, můžete vložit tlařítko jehož metoda bude:

```
` Metoda objektu bLicence
CHANGE WEB LICENSE
```

Příkazy a odkazy pro Uživatelé a skupiny

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SAVE VARIABLES (ULOŽIT PROMĚNNÉ)

Příkazy a odkazy pro Proměnné

Verze 3

SAVE VARIABLES (dokument; prom{; prom2; ...; promN})

Parametr	Typ		Popis
dokument	řetězec	→	Dokument, do kterého uložit 4D proměnné
promx	proměnná	→	proměnné k uložení

Popis

Příkaz SAVE VARIABLES uloží jednu nebo více proměnných do dokumentu jehož název předáte v *dokument*.

Proměnné nemusí být stejného typu, ale musí být Řetězec, Text, Real, Integer, Long Integer, Datum, Čas, Logické nebo Obrázek.

Pokud do *dokument* předáte prázdný řetězec, zobrazí se standardní okno Uložit soubor; uživatel může sám nazvat dokument do kterého se bude ukládat. V tomto případě bude systémová proměnná Document obsahovat název vytvořeného dokumentu.

Pokud jsou proměnné správně uloženy, je proměnná OK nastavena na 1, jinak je nastavena na 0.

Poznámka: Pokud zapisujete proměnné do dokumentu pomocí SAVE VARIABLES, používá 4D vnitřní formát dat. Tyto proměnné může zpět načíst pouze příkazem LOAD VARIABLES. Nepoužívejte příkazy RECEIVE VARIABLE nebo RECEIVE PACKET na dokument uložený pomocí SAVE VARIABLES.

UPOZORNĚNÍ: Tento příkaz nepodporuje proměnné typu array. Místo toho použijte příkazy BLOB.

Příklad

Následující příklad uloží tři proměnné do dokumentu s názvem PředUživ:

```
SAVE VARIABLES ("PředUživ";vsJméno;vlKód;vgIkonObr)
```

Systémové proměnné a Sady

Pokud jsou proměnné správně uloženy, je proměnná OK nastavena na 1, jinak je nastavena na 0.

Dále si přečtěte

BLOB TO DOCUMENT, BLOB TO VARIABLE, DOCUMENT TO BLOB, LOAD VARIABLES, VARIABLE TO BLOB.

Příkazy a odkazy pro Proměnné

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

LOAD VARIABLES (NAČÍST PROMĚNNÉ)

Příkazy a odkazy pro Proměnné

Verze 3

LOAD VARIABLES (dokument; prom{; prom2; ...; promN})

Parametr	Typ		Popis
dokument	řetězec	→	Dokument obsahující 4D proměnné
promx	proměnná	→	proměnné, které mají obdržet hodnotu

Popis

Příkaz LOAD VARIABLES načte jednu nebo více proměnných z dokumentu definovaného v *dokument*. Dokument musí být vytvořen příkazem SAVE VARIABLES.

Proměnné *prom, prom2 ... promN* jsou vytvořeny; pokud existují tak jsou přepsány.

Pokud předáte prázdný řetězec do *dokument*, je zobrazeno standardní okno Otevřít soubor a uživatel může sám vybrat dokument k otevření. Pokud je dokument vybrán, je proměnná Document nastavena na název dokumentu.

Ve zkompilované verzi musí být proměnné stejného typu jako jsou načítané proměnné.

UPOZORNĚNÍ: Tento příkaz nepodporuje proměnné array. Místo toho použijte příkazy BLOB.

Příklad

Následující příklad načte tři proměnné z dokumentu s názvem PředUživ

```
LOAD VARIABLES ("PředUživ";vsJméno;vlKód;vgIkonObr)
```

Systémové proměnné a Sady

Pokud jsou proměnné správně načteny, je proměnná OK nastavena na 1, jinak je nastavena na 0.

Dále si přečtěte

BLOB TO DOCUMENT, BLOB TO VARIABLE, DOCUMENT TO BLOB, RECEIVE VARIABLE, VARIABLE TO BLOB.

Příkazy a odkazy pro Proměnné

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

CLEAR VARIABLE (VYMAZAT PROMĚNNOU)

Příkazy a odkazy pro Proměnné

Verze 3

CLEAR VARIABLE (proměnná)

Parametr	Typ		Popis
proměnná	Proměnná	→	Proměnná k uvolnění a vymazání

Popis

Tento příkaz pracuje odlišně ve zkompiloné a v nezkompilované databázi.

V nezkompilované verzi

CLEAR VARIABLE odstraní proměnnou z paměti. Tím proměnná bude Nedefinovaná; při pokusu o přečtění její hodnoty vznikne chyba syntaxe.Když do této proměnné přiřadíte nějakou hodnotu, 4D ji vytvoří za "chodu". Po vymazání proměnné, vrátí Undefined True při použití na tuto proměnnou.

Ve zkompilované verzi CLEAR VARIABLE nastaví pouze roměnnou na její výchozí hodnotu (tzn. prázdný řetězec u proměnných Řetězec a Text, 0 pro číselné proměnné, žádný prvek pro array, atd.). Proměnná stále existuje - ve zkompilované verzi nikdy nemůže být nedefinovaná.

Proměnná, kterou mažete musí být procesní nebo meziprocesní.

Poznámka: Pokud proces končí, nepotřebujete procesní proměnné mazat. 4D je vymaže automaticky.

Místní proměnné, začínající znakem dolaru ($), nemohou být vymazány příkazem CLEAR VARIABLE. Jsou automaticky vymazány jakmile skončí metoda, kde byly vytvořeny.

Příklad

Ve formuláři používáte rozevírací seznam, jehož jediným účelem je rozhraní uživatele. Jinými slovy používáte array během vstupu dat, ale jakmile formulář zavřete, již jej nepotřebujete. Vytvoříte jej během zavedení formuláře:

```
   ` Metoda objektu asMyDropDown
Case of
÷ (Form event=On Load)
      ` Vytvořit array jedním ze způsobů
   ARRAY STRING(63;asMyDropDown;...)
      ` ...
:(Form event=On Unload)
      ` Již array nepotřebujeme
   CLEAR VARIABLE (asMyDropDown)
      ` ...
End case
```

Dále si přečtěte

Undefined.

Příkazy a odkazy pro Proměnné

Undefined (Nedefinováno)

Příkazy a odkazy pro Proměnné

Verze 3

Undefined (proměnná) → Logické

Parametr	Typ		Popis
proměnná	Proměnná	→	Proměnná k testování
Výsledek funkce	Logické	←	True = Proměnná je nedefinovaná False = Proměnná je definovaná

Popis

Undefined vrací True, pokud proměnná nebyla definována a False jestliže proměnná již byla definována. Proměnná je definovaná když je do ní přiřazena hodnota. Proměnná je nedefinovaná, jestliže do ní nebyla přiřazena hodnota a nebo byla vymazána pomocí CLEAR VARIABLE.

Pokud je databáze kompilovaná pomocí 4D Compiler, Undefined vrací False pro všechny proměnné.

Příklad

1. Do verze 6 byl dobrý způsob testování jestli jste ve zkompilované verzi následovný:

```
anyVar:="Hello"
CLEAR VARIABLE(anyVar)
If (Undefined(anyVar))
   ` Jste v nezkompilované verzi
Else
   ` Jste ve zkompilované verzi
End if
```

Od verze 6 je výhodnější používat příkaz Compiled application.

2. Následující kód řídí vytváření procesů při vybrání položky nabídky patřičného modulu vaší aplikace. Pokud proces již existuje, přenese jej na popředí; pokud neexistuje tak jej vytvoří. K tomu v metodě databáze Při spuštění vytvoříte meziprocesní proměnnou <>PID_.. pro každý modul vaší aplikace.

Během vývoje aplikace přidáváte nové moduly. Místo upravování metody databáze Při spuštění (k předání a nastavení nové proměnné <>PID_...) a pak znovu otevření databáze k přeinicializování všeho při každém přidání modulu, použijete příkaz Undefined k řízení přidání nového modulu za běhu:

```
   ` Globální procedura M_PRIDAT_ZAKAZN
   ` Tento řádek se stará o pokročilejší stádia vývoje
If (Undefined(<>PID_PRIDAT_ZAKAZN))
   C_LONGINT(<>PID_PRIDAT_ZAKAZN)
   <>PID_PRIDAT_ZAKAZN:=0
End if
If (<>PID_PRIDAT_ZAKAZN=0)
   <>PID_PRIDAT_ZAKAZN:=New process("P_PRIDAT_ZAKAZN";
                                    64*1024;"P_PRIDAT_ZAKAZN")
Else
```

```
    SHOW PROCESS(<>PID_PRIDAT_ZAKAZN)
    BRING TO FRONT(<>PID_PRIDAT_ZAKAZN)
End if
   ` Poznámka: P_PRIDAT_ZAKAZN, metoda procesu, nastaví
   ` <>PID_PRIDAT_ZAKAZN na 0 pokud skončí.
```

Dále si přečtěte

CLEAR VARIABLE.

Příkazy a odkazy pro Proměnné

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Web služby, Popis

Příkazy **a odkazy pro Web server**

Verze 6.0

Jak 4th Dimension tak 4D Server obsahují Web server engine, který vám umožňuje publikovat vaše databáze na Webu. Jedinečné a zatím nepřekonané možnosti 4D Web serveru jsou:

• **Přímé Web služby**

Vaše databáze jsou rovnou publikovány na Webu. Nepotřebujete vyvíjet systém databáze, Web stránky ani žádný CGI interface mezi nimi. Vaše databáze je vaše Web stránka.

• **Bezpečnost připojení**

Mnoho nastavitelných možností a Metoda databáze Při ověření WEB vám umožňuje regulovat přístup do vaší Web databáze. Vlastnost "Generický uživatel Web" zjednodušuje řízení přístupu uvnitř databáze. Nadto možnost definovat výchozí složku HTML vám umožní omezit přístup k diskovým souborům.

• **On-line, transparentní překlad HTML**

4th Dimension on-line dynamicky překládá vaše formuláře a komponenty návrhu do HTML stránek. Tyto nové HTML stránky jsou okamžitě dostupné pro Web prohlížeč, pokud je právě připojen k databázi. Dnes je většina systémů Web databází založená na systémech CGI nebo statických HTML stránkách.

CGI systém vyžaduje vývoj databáze, Web stánek a CGI. Systémy založené na statických HTML stránkách vyžadují program který bude vaše úpravy překládat do HTML stránek pokaždé, když změníte prvky návrhu ve vaší databázi. V obou případech jsou komponenty Web vytvářeny off-line a vyžadují ruční zásah od vývojáře databáze a/nebo Webu. Ve 4th Dimension můžete vaše komponenty návrhu měnit kolikrát chcete a kdy chcete. Jakmile uložíte změny v prostředí návrháře, jsou okamžitě dostupné pro Web prohlížeč. Tak můžete během vývoje a testování vaší aplikace ihned testovat výsledek na právě připojeném Web prohlížeči.

• **Dynamický přístup k záznamům a datům**

4D zachází s Web prohlížečen jako se standardním klientem databáze 4D. Pokud například změníte některé záznamy v místní databázi 4D nebo z klienta 4D Serveru, jsou tyto změny ihned dostupné na Web prohlížeči. Nepotřebujete znovu zpracovat záznamy pro HTML publikování jako v mnoha jiných aplikacích.

• **Udržování sekce a souvislostí databáze**

Web prohlížeč, jak již jeho název napovídá, vám umožňuje prohlížet jednotlivé Web stránky v náhodném pořadí - můžete skákat z jedné stránky na jinou, z jedné adresy na druhou, atd. Při použití Web prohlížeče pro klient/server databázi, potřebujete aby se prohlížeč pohyboval podle transakcí databáze. Pokud například vkládáte nový záznam, potřebujete vnutit, že záznam musí být potvrzen nebo zrušen. Uživatel nesmí mít možnost opustit zadávaný záznam pomocí navigace Web prohlížeče a opustit tak nedokončený záznam.

Web server 4D obsahuje vestavěné udržování sekce a souvislotí databáze. Přes URL Web stránek ovládá jedinečné kontextové a podkontextové ID čísla, která zaručují kompletní synchronizaci mezi platnou Web stránkou zobrazenou prohlížečem a kontextu připojené 4D databáze.

• **Mód bez souvislostí / nekontextní mód**

V tomto módu se Web server 4S stává normálním HTTP serverem; statické stránky jsou posílány bez nutnosti nějakého pořadí. Vždy můžete použít rozšířených možností 4D s použitím "semidynamických" stránek a specielních URL.

Web server 4D můžete použít v jakémkoli módu chcete: kontextní mód, nekontextní mód nebo je ze chodu přepínat podle vašich potřeb.

• **Udržování více uživatelů**

Jakmile se stanou klienty 4D, je s Web prohlížeči zacházeno jako s kompletními klienty databáze. Pokud například budete upravovat záznam na Web prohlížeči, 4D tento záznam automaticky uzamkne a zabrání tak ostatním klientům tento záznam měnit. Po potrzení nebo zrušení vstupu, 4D automaticky odemkne záznam. 4D vám umožňuje zadávat data v transakci, jak je to možné ve 4th Dimension a 4D Client.

• **Udržování Web procesů**

4D obsahuje několik procesů pro řízení architektury Web klient/server. Vestavěný hlavní Web Server proce řídí a umožňuje připojení přes Web. Jakmile je Web prohlížeč připojen, je ovládán ze samostatného procesu, vytvořeného pro tento účel. Jako výsledek plně integrované více-dotazové architektury 4D, mohou normální nebo Web klienti provádět více databázových požadavků (jako dotazy), které budou najednou prováděny enginem databáze. Klientům je garantováno, že požadavky určitého Web klienta nebudou zasahovat do kontextu jiných procesů.

• **Optimalizovaná architektura Web serveru**

Engin Web Serveru 4D má stejné možnosti jako engin databáze 4D. Pokud například načtete sérii hodnot záznamů do array je operace provedena místně na stroji Web Serveru. Výsledek je pak poslán jako celek na na Web klienta.

Vzhledem k tomu, že Web spojení je normální a plně funkční 4D proces na straně Web serveru, můžete mít spuštěny všechny vaše oblíbené algoritmy 4D. Jsou prováděny místně na straně Web serveru a pouze výsledek, pokud je nějaký, je posílán na Web prohlížeč. Můžete například provést dotaz, který potřebuje vztahy, sady a vypočítat statistické údaje a na Web prohlížeč se pošle pouze výsledek. Komplexní systém Web databáze můžete vytvořit bez toho, aby jste byli expertem na Web a vše pouze úpravou a kompilací vaší současné aplikace.

• **Zahrnutí HTML a JavaScript**

Ačkoli 4D většinou provede vše co potřebujete k publikování databáze na Webu, můžete použít HTML a JavaScript kód k dotvoření vašeho vývoje ve 4D. Například můžete vytvořit domovskou stránku vaší databáze pomocí stránky HTML.

Můžete vytvořit vlastní HTML stránky a vložit je na Web pomocí příkazů SEND HTML FILE nebo SEND HTML BLOB. Můžete také použít HTML ve formuláři jehož vzhled na Web prohlížeči je složený z formuláře a HTML objektu obsaženého ve formuláři. Do HTML pak můžete vložit JavaScript kód, který bude provádět akce a vstup dat na straně Web prohlížeče bez poslání dotazu zpět na server.

• **Svázání mezi HTML a objekty 4D**

Pokud použijete HTML stránky a kód ve vašem vývoji, potřebujete na straně 4D mít možnost získat hodnoty a data která byla zadána do objektů HTML. Místo psaní složitých metod, 4D obsahuje jednoduchý systém svázání objektů HTML do proměnných 4D - vaše objekty musí mít pouze stejný název. Důsledkem je, že implementace anylýz a odpovědí na HTML žádosti je snadná. Musíte pouze napsat kód, který bude zpracovávat automaticky plněné proměnné posílané Web prohlížečem.

Kam jít nyní?

• K nastavení vašeho stroje a databáze pro publikování na Webu si přečtěte Web služby, Nastavení.
• Pro naučení se jak publikovat databázi na Webu si přečtěte Web služby, Poprvé (část I) a Web služby, Poprvé (část II).
• Pro naučení se více o systému řízení přístupu si přečtěte Web služby, Bezpečnost připojení.
• Pro informace o zahrnutí HTML si přečtěte Web služby, Zahrnutí HTML a JavaScriptu.
• Pro informace o vzájemném působení Web a Procesů si přečtěte Web služby, Procesy Web spojení.
• Pro informace o nekontextním módu si přečtěte Web služby, Nekontextní mód.

Dále si přečtěte

SEND HTML FILE, SET HOME PAGE, SET HTML ROOT, SET HTML ROOT, SET WEB DISPLAY LIMITS, SET WEB TIMEOUT, STOP WEB SERVER, Web služby, Bezpečnost připojení, Web služby, Nekontextní mód, Web služby, Web server nastavení.

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Web služby, Nastavení

Příkazy a odkazy pro Web server

Verze 6.0

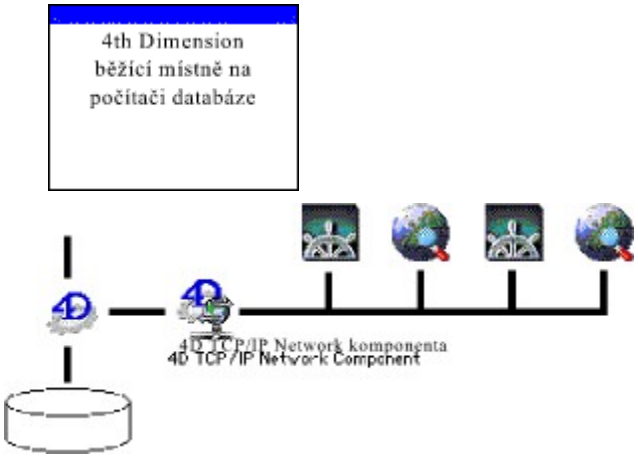
4th Dimension a 4D Server obsahují Web server,pomocí kterého můžete publikovat vaše databáze na Webu, a to jak dynamické tak statické.

4th Dimension a Web

Pokud publikujete vaší databázi na Webu pomocí 4th Dimension, můžete najednou:

- Používat databázi v samotné 4D
- Připojit se k databázi pomocí Web prohlížeče

Je to ukázáno v následujícím obrázku:



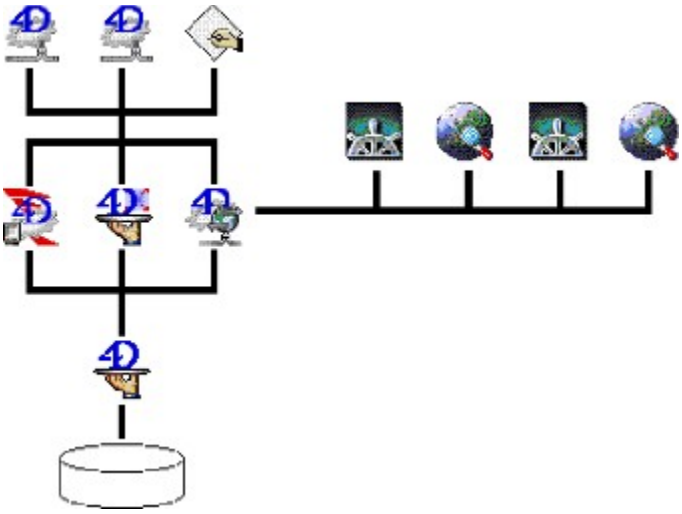
Místní databáze

4D Server a Web

Pokud publikujete vaší databázi na Webu pomocí 4D Server, můžete se současně připojit k databázi a pracovat s ní pomocí:

- Stanice 4d Client
- Aplikace založené na 4D Open
- Web prohlížeč.

Je to znázorněno v následujícím obrázku:



Obsluha databáze 4D na Webu

K obsluhování 4D databáze na Webu pomocí 4th Dimension nebo 4D Server, musíte mít patřičné rozšiřující licence Web a síťové komponenty:

• Požadované rozšíření Web musí být instalováno ve vaší aplikaci. Jestli chcete vědět více informací, podívejte se do Příručky instalace 4D.

• Web spojení jsou prováděna přes Síť pomocí protokolu TCP/IP. Proto:

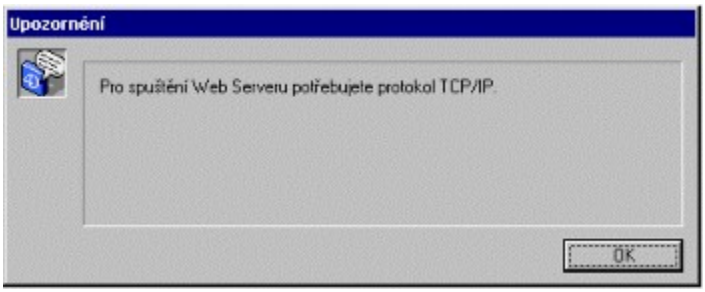
- Musíte mít na vašem počítači instalovaný a správně instalovaný TCP/IP. Informace najdete v příručce k vašemu počítači nebo Operačnímu systému.

- Musíte mít instalované síťové komponenty 4D pro TCP/IP; na Macintoshi musí být síťové komponenty instalovány přímo do 4th Dimension, 4D Server nebo vlastních připojených aplikací 4D. Na Windows musí být síťové komponenty ve složce ACI umístěné v aktivní složce WINDOWS.

Poznámka: V obou případech najdete podrobnější informace v manuálu "Network Components for 4D Server".

• Po instalování nebo vyzkoušení TCP/IP, potřebujete zapnout Web služby uvnitř 4D. Tento bod je dále popsán.

Pokud se pokusíte zapnout Web server bez instalovaného protokolu TCP/IP nebo bez síťových komponentů 4D pro TCP/IP, 4th Dimension zobrazí následující upozornění:



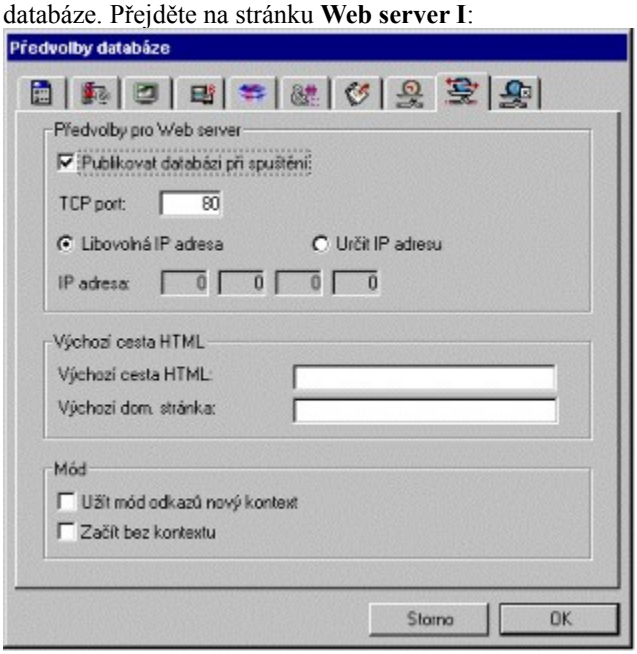
Pokud se objeví tato zpráva, proveďte instalaci jak je popsáno nebo opravte nastavení TCP/IP.

Zapnutí Web služeb

Web služby 4D mohou být zapnuty třemi způsoby:

• S použitím nabídky Web Server v hlavním záhlaví nabídek prostředí uživatele 4D Serveru nebo 4th Dimension. Nabídka Web Server vám umožňuje zapnout a vypnout Web služby podle vašich požadavků.

• Automatické publikování databáze při každém spuštění. Pro automatické publikování použijte položku **Předvoby databáze** z nabídky **Soubor** v prostředí návrháře 4th Dimension nebo 4D Server. Objeví se okno Předvolby databáze. Přejděte na stránku **Web server I**:



Na této straně zaškrtněte možnost **Publikovat databázi při spuštění** a klepněte na **OK**. Jakmile to uděláte, bude databáze automaticky publikována na Web při každém spuštění pomocí 4th Dimension nebo 4D Server.

• Programově provedením příkazu START WEB SERVER.

Tip: Pro zapnutí a vypnutí Web služeb 4D nepotřebujete databázi vypínat a znovu zapínat. Můžete přerušit nebo zapnout publikování kolikrát chcete pomocí nabídky Web server nebo příkazů START WEB SERVER a STOP WEB SERVER.

Připojení k databázi 4D publikované na Webu

Po publikování databáze na Webu se můžete připojit pomocí Web prohlížeče. K tomu:

• Pokud má vaše Web stránka registrovaný název (např. "Kytky Mezinárodní"), vložte tento název do oblasti Otevřít, Adresa nebo Umístění ve vašem prohlížeči.

• Pokud nemáte zadán žádný název, použijte IP adresu vašeho počítače (např. 123.4.567.89) v oblasti Otevřít, Adresa nebo Umístění ve vašem prohlížeči.

V této chvíli se můžete připojit, pokud je vše v pořádku. Nemůžete se připojit pokud nastane jedna z následujících situací:

1. Obdržíte zprávu jako "Aplikace...nemůže otevřít server, Spojení nebylo navázáno.".

V tomto případě otestujte následující:

• Ověřte jestli je název zadané adresy správný.

• Ověřte jestli je spuštěna 4th Dimension nebo 4D Server a jestli je zapnut jejich Web Server.

• Ověřte jestli není databáze nastavena na jiný TCP port než výchozí Web TCP port (viz situaci 3)

• Ověřte jestli je TCP/IP správně nastaveno na stroji serveru a prohlížeče. Oba počítače musí být na stejné síti a podsíti nebo vaše routery musí být správně nastaveny.

• Ověřte hardware připojení.

• Pokud netestujete vaše místní připojení, ale připojujete se na Web server na Internetu nebo Intranetu k někomu jinému, může být zpráva pravdivá; server může být vypnut nebo přetížený. Zkuste se připojit později a nebo kontaktujte poskytovatele Web služeb.

2. Připojíte se, ale objeví se Web zpráva s touto zprávou ""Tato databáze nebyla ještě nastavena pro WEB". Toto zpráva znamená že Web služby jsou zapnuty a vy jste se správně připojili k databázi. Nicméně databáze potřebuje vložit minimální požadavky aby byla použitelná na Webu. Jestli chcete vědět více informací, přečtěte si Web služby, Poprvé (část I).

3. Připojíte se, ale nedostanete stránku kterou jste očekávali! Toto se může stát pokud máte více Web serverů běžících současně na stejném stroji. Příklady:

• Máte spuštěnu jednu 4D Web databázi na Windows NT 4.0, který má spuštěné své vlastní Web služby.
• Máte spuštěno více 4D Web databází na jednom počítači.

V tomto případě potřebujete vyměnit číslo TCP portu na kterém je publikována vaše Web databáze. K tomu si přečtěte následující část.

Poznámka: Pokud je vaše databáze chráněna sytémem hesel, můžete vložit správné jméno uživatele a heslo (více informací najdete v části Web služby, Bezpečnost připojení).

Nastavení TCP portu na specifickou hodnotu

Jako výchozí publikuje 4D Web databázi na standardním TCP portu, jehož číslo je 80. Pokud je port použit jinou Web službou, potřebujete změnit TCP port používaný databází 4D. K tomu vyberte položku **Předvolby databáze** z nabídky **Soubor** v prostředí návrháře 4D Server nebo 4th Dimension. Jděte na dostupnou oblast **TCP port** a změňte platnou hodnotu (TCP port nepoužívaný jinou TCP/IP službou běžící na stejném počítači).

Poznámka: Pokud zadáte 0, 4D použije výchozí port 80.

Z Web prohlížeče potřebujete zadat nevýchozí TCP porty k přihlášení k databázi. Adresa musí mít příponu odpovídající začínající dvojtečkou pro číslo portu. Pokud máte například TCP port 700, musíte adresu zadávat "123.4.567.89:700".

UPOZORNĚNÍ: Pokud používáte jiný TCP port než výchozí 80, dávejte si pozor na použití čísel, která jsou výchozí pro jiné Web služby které chcete mít spuštěné společně. Pokud například chcete použít FTP protokol na stroji Web serveru, nepoužívejte TCP porty 20 a 21 (pokud si nejste jisti co děláte), které jsou výchozí pro tento protokol. Jestli chcete vědět více informací o číslech TCP portů a o protokolech, kupte si nějakou knížku o TCP/IP protokolu a podívejte se na tabulku RFC 1700 standardní přiřazená čísla. Čísla portů pod 256 jsou rezervované pro známé služby, čísla portů od 256 do 1024 jsou reservované pro specifické služby platformy UNIX. Pokud použijete čísla portu v několika tisících, bude vše v pořádku.

Dále si přečtěte

SEND HTML FILE, SET HTML ROOT, SET WEB DISPLAY LIMITS, SET WEB TIMEOUT, STOP WEB SERVER.

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Web služby, Poprvé (Část I)

Příkazy a odkazy pro Web server

Verze 6.0

Příklad v kontextním módu s udržováním souvislostí

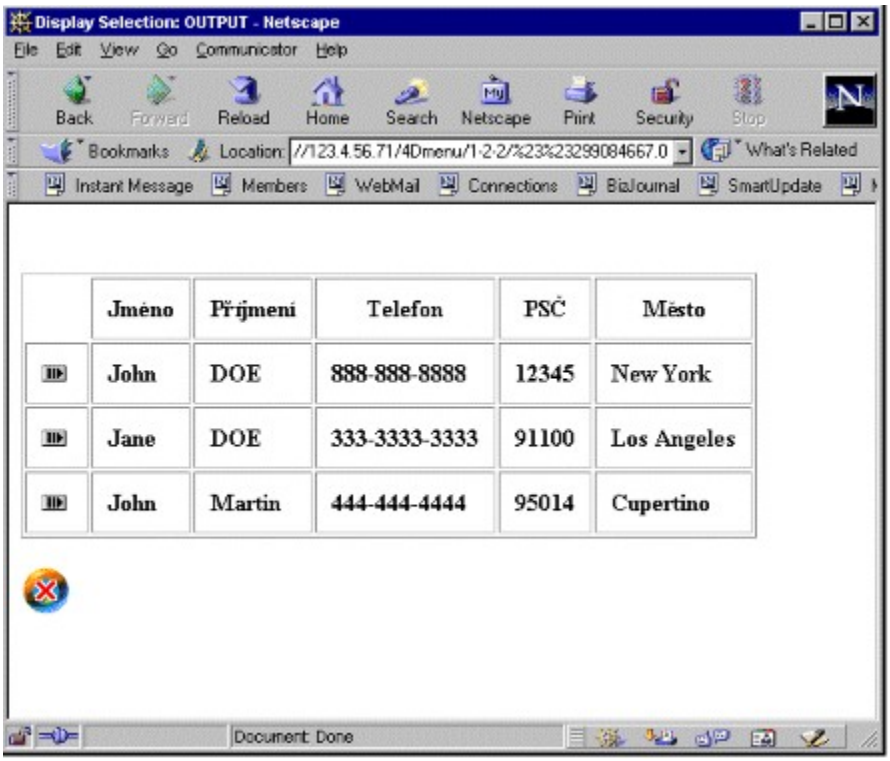
Podívejme se na příklad databáze 4D automaticky publikované na Webu v standardním kontextním módu. Struktura této databáze (ukázaná na konci této části) je jednoduchá; databáze je složena z jedné tabulky, vstupního formuláře, výstupního formuláře a záhlaví nabídek. Domovská stránka je vytvořena. Databáze je publikována jako Web server.

1. Připojení k Web serveru databáze.

Připojte se k Web serveru otevřením Web prohlížeče. Dostanete následující Domovskou Web stránku, která je získána Netscapem na Windows.

2. Zobrazení a procházení záznamů.

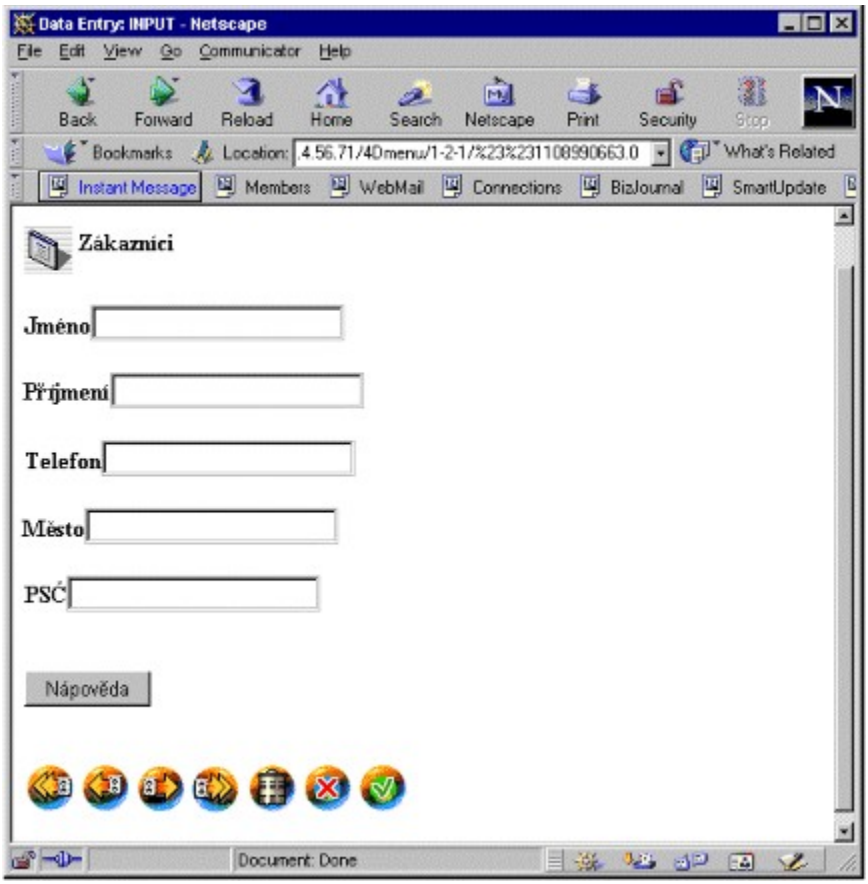
Klepněte na podtržený text Seznam existujících záznamů. Provede se Web ekvivalent zobrazení záznamů 4D na obrazovku:



V této chvíli můžete prohlížet záznamy podle vašich potřeb. Po klepnutí na tlačítko Hotovo (označené červeným X), se dostanete zpět na Domovskou stránku.

3. Přidání záznamů.

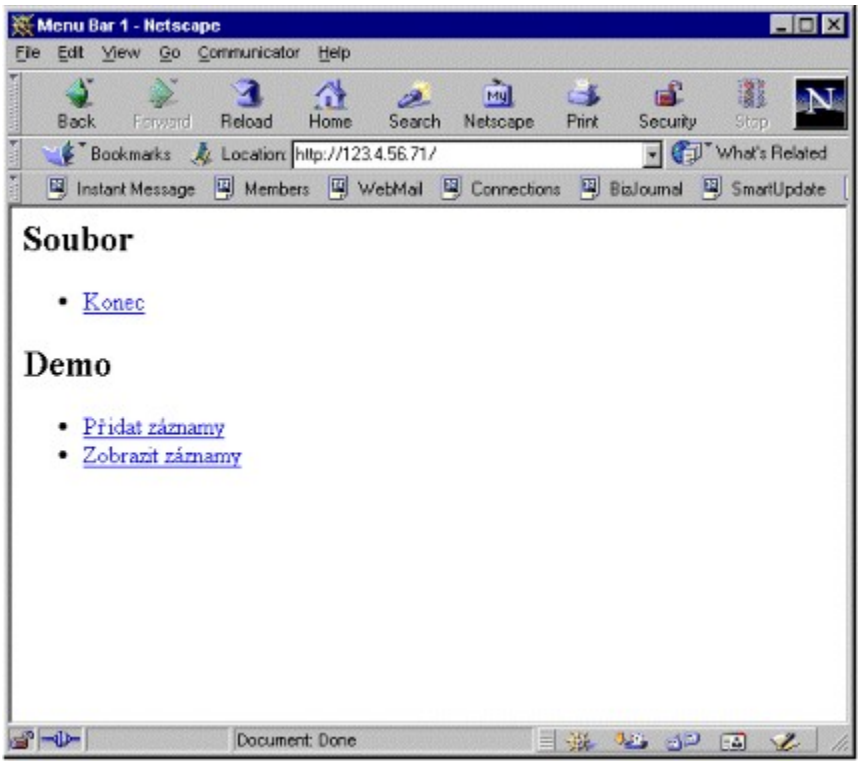
Na domovské stránce klepněte na podtržený text Přidat záznamy k zobrazení Web stránky pro přidání záznamů:



Můžete přidat tolik záznamů kolik potřebujete. Jakmile jste hotovi, klepněte na tlačítko Zrušit (to s červeným křížem) a vraťte se na Domovskou stránku.

4. Seznam nebo přídání záznamu v Hlavní nabídce

Na Domovské stránce klepněte na Jít na hlavní záhlaví nabídek. Opustíte tak Domovskou stránku a dostanete Web ekvivalent Vlastní nabídky 4D:



V této chvíli vám klepnutí na položky nabídek umožní Zobrazit nebo Přidat záznamy; stejné metody 4D, které jsou použity na Domovské stránce jsou přiřazeny k položkám nabídek.

5. Ukončení spojení

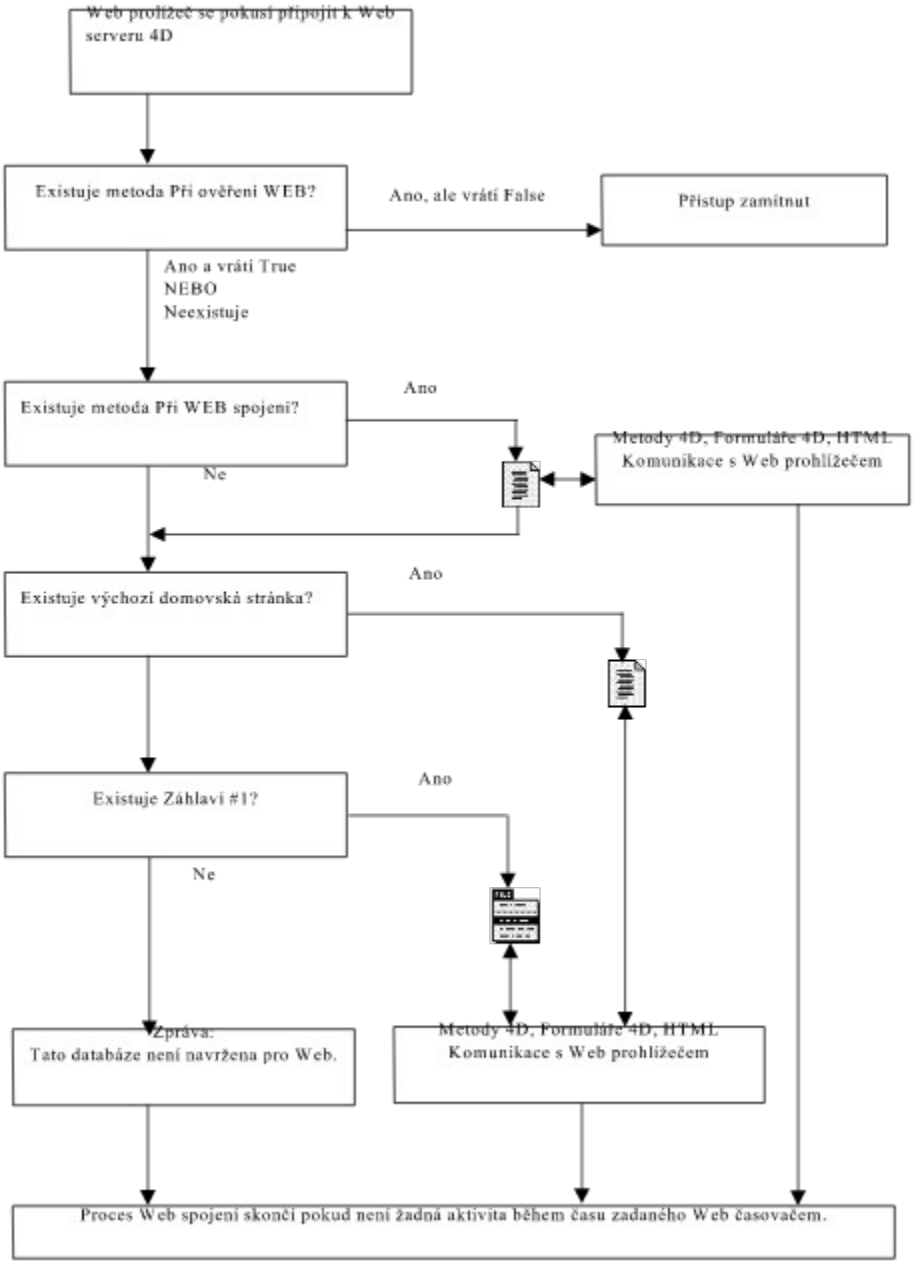
Jakmile jste hotovi, vypněte váš prohlížeč a 4D ukončí proces Web spojení jakmile skončí časový limit.

Inicializace Web spojení

Pokaždé, když se Web prohlížeč připojí k databázi 4D publikované jako Web Server, 4D provede následující akce:

• Provede metodu Při ověření WEB, pokud existuje
• Pokud tato metoda vrátí True nebo neexistuje, spustí se metoda Při WEB spojení, pokud existuje.
• Pokud nejsou tyto metody nebo jsou dokončeny, 4D zobrazí výchozí Domovskou stránku, která je definovaná v Předvolbách databáze, pokud je nějaká.
• Pokud není Domovská stránka, 4D zobrazí platné záhlaví nabídek (jako výchozí záhlaví #1), pokud existuje.
• Pokud není ani Domovská stránka ani záhlaví nabídek, 4th Dimension zobrazí výchozí Web stránku s obsahem: "Tato databáze není nastavena na Web".

Následující obrázek ukazuje tyto akce:



Poznámka: Tento obrázek popisuje výchozí mód Web serveru 4D, **kontextní mód**. Pokud databáze začíná v **nekontextním módu**, tato sekvence skončí po volání k Domovské stránce, pokud existuje (přečtěte si část Web služby, Nekontextní mód) Metoda databáze Při Web spojení může volat jakoukoli metodu nebo formulář stejně jako HTML stránku. Metoda databáze může ovládat celou sekci.
Web připojení k 4D nebo 4D Server není stejné jako klient/server připojení. HTTP protokol, který podporuje HTML a Web není protokol "založený na sekci"; spíše to je protokol "založený na žádostech". V klient/server se připojujete, pracujete v sekci a odpojíte se od serveru. S HTTP, si každá akce vyžaduje potvrzení nebo akci na Web serveru, dotaz je poslílán na server. HTTP žádost může být znázorněna "Připojení+Žádost+Očekávání odpovědi+Odpojení."

Ke spuštění klient/server sekce přes HTTP, ovládá 4D jako výchozí, přes kódování URL, kontext, který jedinečně identifikuje vaše Web spojení a ve stejnou chvíli přiřadí spojení k procesu 4D který spojení ovládá; to je Kontextní mód.

Nicméně v tomto módu 4D nemá způsob jak simulovat ekvivalent ukončení klient/server spojení které ukončí sekci.

Toto je důvod proč ukončení klient/server sekce je prováděno přes časové schéma. 4D proces, který ovládá spojení skončí pokud není žádná aktivita po dobu definovaného časového limitu Web.

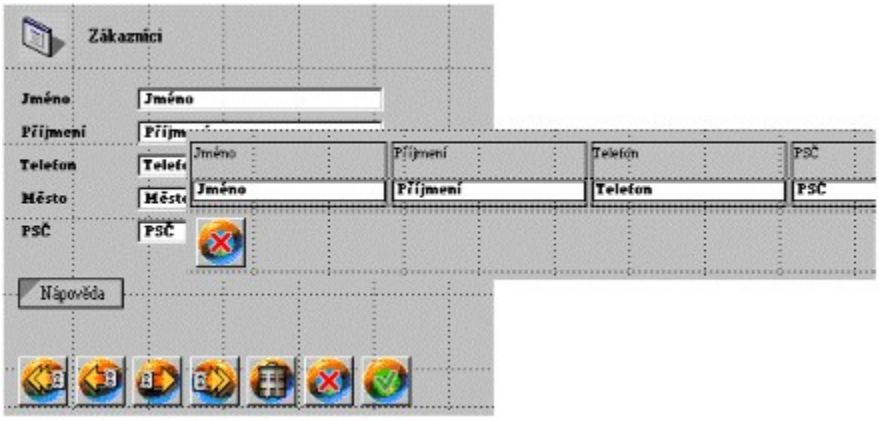
Databáze a Web server v jednom

Web server 4D můžete kompletně ovládat pomocí záhlaví nabídek 4D, formulářů a metod. V předchozím příkladu, zobrazení seznamu a přidání záznamů, bylo vše provedeno jednoduchými metodami a formuláři. Pokud nemáte použitu žádnou HTML stránku, Web prohlížeč může po připojení obdržet záhlaví nabídek #1, jak bylo ukázáno.

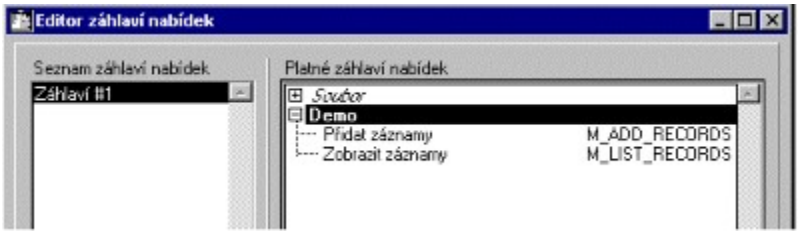
Pokud vynecháme domovskou HTML stránku, naprogramování Web serveru, podporujícího transakce klient/server databáze, je shodné s vytvářením databáze 4D na Windows nebo Macintoshi, pro jednoho nebo více uživatelů. Následující kroky popisují proces vytvoření takovéto příkladové databáze.

1. Zde je struktura příkladové databáze.

2. Přidají se Vstupní a výstupní formuláře k umožnění práce se záznamy.



3. Je přidáné Záhlaví #1 pro umožnění práce s Vlastními nabídkami a k podporování Web spojení.



4. Jsou napsány dvě metody projektu.

A je hotovo!

V méně než pěti minutách máte vytvořenu databázi která může pracovat místně na 4D i na Intranetu nebo Internetu.

Přečtěte si nyní Web služby, Poprvé (část II).

Dále si přečtěte

SEND HTML FILE, SET WEB DISPLAY LIMITS, SET WEB DISPLAY LIMITS, SET WEB TIMEOUT, START WEB SERVER, STOP WEB SERVER.

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Web služby, Poprvé (Část II)

Příkazy a odkazy pro Web server

Verze 6.0

Přidání HTML vlastností do databáze (kontextní mód)

Pokud chcete Web uživateli zobrazit více než pouze záhlaví nabídek vaší databáze, můžete do databáze Při WEB spojení vložit metodu která zobrazí formulář 4D nebo HTML stránku nebo nastavit Web server na zobrazení výchozí domovské stránky. Můžete použít existující HTML stránky vytvořené nějakým HTML nástrojem.

Vaše 4D Web stránka může být kompletně udělána v 4D systému nebo kombinace formulářů 4D a HTML stránek. Zajímavé na použití HTML stránek z databáze 4D je to, že můžete těžit z výhod vývojového prostředí 4D a HTML. Zapamatujte si, že pokud nechcete, nemusíte HTML stránky používat!

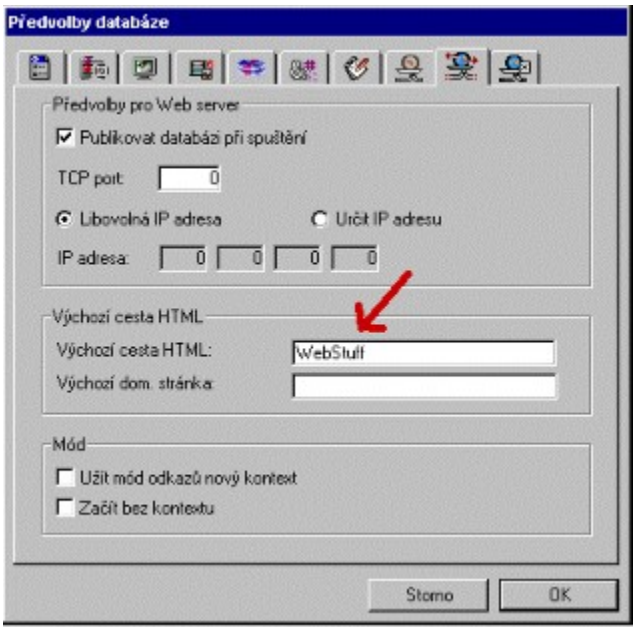
Příklad

V tomto příkladu přidáte existující HTML stránky do databáze. Následující obrázek ukazuje složku databáze.

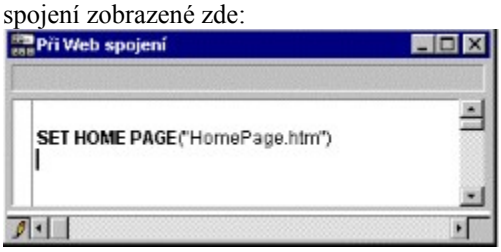
Použijeme HTML dokument HomePage.htm jako domovskou stránku databáze. Dokument 4DV6Logo.gif je obrázek použitý v HTML dokumentu.

Patřičné nastavení Web serveru 4D je následující:

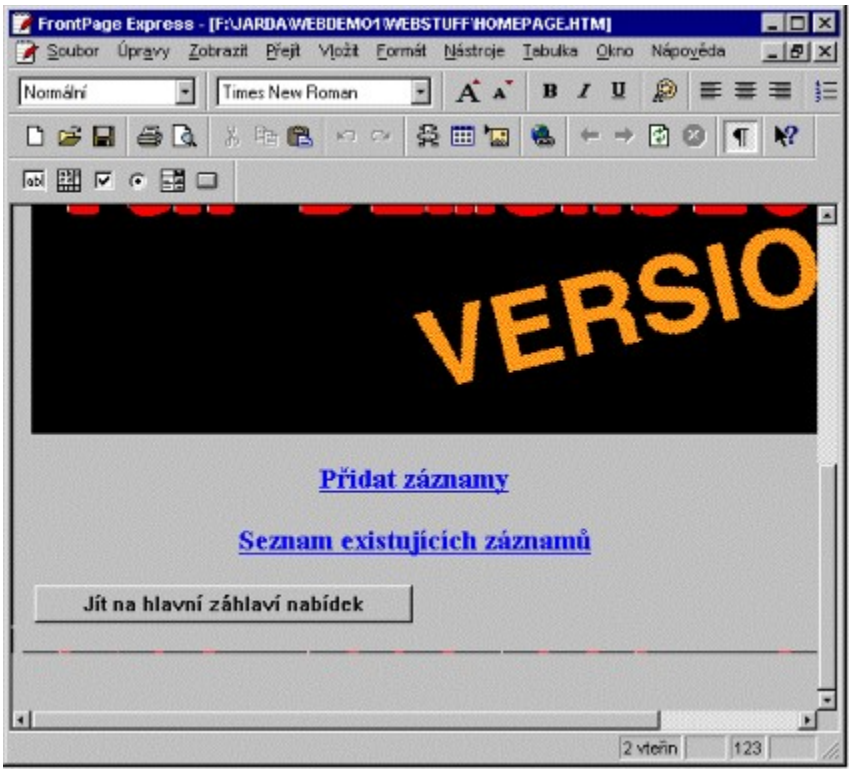
• Na straně "Web server I" v Předvolbách databáze předáte novou Výchozí cestu HTML:



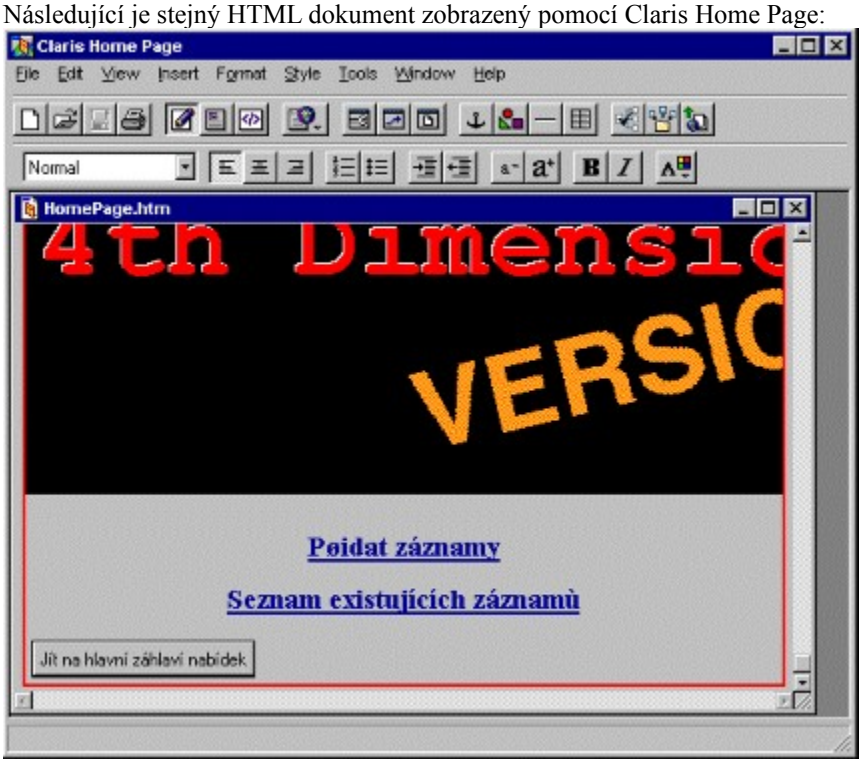
Oblast Výchozí cesta HTML řekne 4th Dimension kde má hledat HTML dokumenty.
Poznámka: Jestli chcete vědět více informací o tomto dialogovém okně, přečtěte si Web služby, Web server nastavení.
• HTML dokument je pak deklarován jako domovská stránka pro platný Web proces v metodě databáze Při WEB spojení zobrazené zde:



Následující je HomePage.htm zobrazená v Microsoft Front Page:



Následující je stejný HTML dokument zobrazený pomocí Claris Home Page:

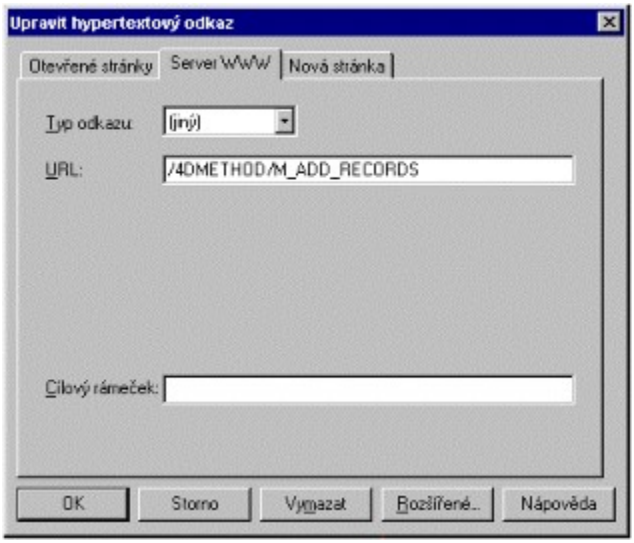


Odkazující URL

Dvě podtržené textové položky "Přidat záznamy" a "Seznam existujících záznamů" provádějí spuštění metody projektu 4D M_ADD_RECORDS a M_LIST_RECORDS přes jejich URL. Konvence je velmi jednoduchá: jakýkoli HTML objekt se může odkazovat na metodu databáze s pomocí URL "/4DMETHOD/Název_vaší_metody".

Zde je URL pro text "Přidat záznamy" v Claris Home Page:

Zde je stejný URL v Microsoft Front Page:



Po definování těchto odkazů, jakmile Web prohlížeč pošle zpět URL, 4D provede metodu projektu definovanou za klíčovým slovem /4DMETHOD/. Pak, po skončení běhu metody, pošle pak zpět do HTML obah jejího provedení. Nezapomeňte že metoda projektu může zobrazit formuláře 4D, jiné HTML stránky, atd.

Tlačítka

HTML dokument v tomto příkladu obsahuje tlačítko použité k potvrzení záznamu. Jsou tři typy tlačítek HTML: normální, potvrzení a reset.

• Normální - K normálním tlačítkům mohou být přiřazeny URL které se odkazují na metody 4D pomocí klíčového slova /4DMETHOD/. Normální tlačítka jsou používána pro navigaci.

• Potvrzení - Tlačítka Potvrzení potvrdí formulář s hodnotami zadanými uživatelem (pokud jsou nějaké) na Web server. Jsou používány pro ovládání vstupu dat který zadáte přes HTML stránku a ne přes formulář 4D.

• Reset - Tato tlačítka nejsou příliš použitelná ve vývoji 4D; odstraní hodnoty zadané uživatelem (pokud nějaké) a neposílá dotaz na server.

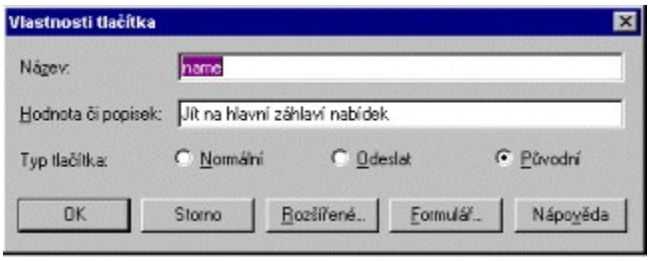
Při zahrnování HTML stránek do 4D, budete obvykle používat normální a potvrzovací tlačítka.

Definování metody 4D ke spuštění

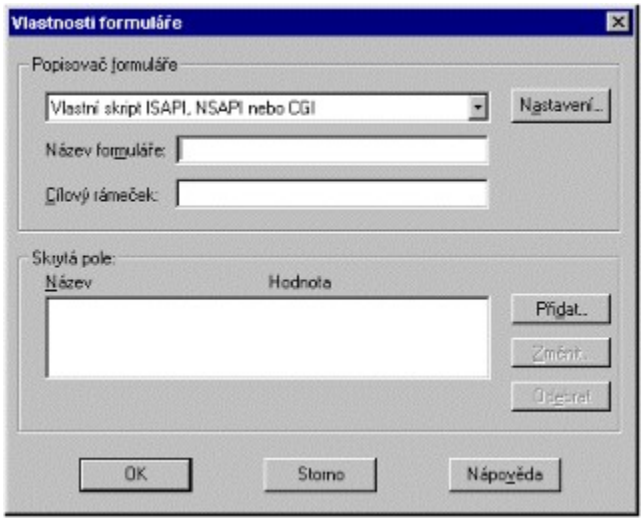
K potvrzení HTML formuláře na straně 4D, potřebujete definovat POST akci metody 4D která bude spuštěna po potvrzení formuláře.

Definování POST akce metody 4D s použitím Microsoft Front Page:

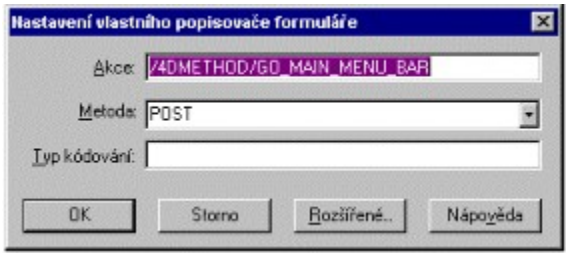
1. Prohlédněte si vlastnosti tlačítka potvrdit. Zobrazí se následující dialogové okno:



2. Klepněte na tlačítko Formulář... Objeví se okno vlastností formuláře.



3. Klepněte na tlačítko Nastavení. Objeví se následující dialogové okno:

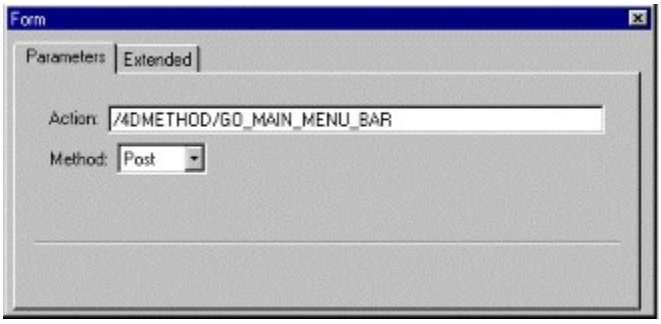


4. Jako akci definuje URL odkazu "/4DMETHOD/Název_vaší_metody". Zde vložíme GO_MAIN_MENU_BAR jako metodu 4D. Tato metoda se spustí při klepnutí na tlačítko.

5. Vybertte **POST** z rozevíracího seznamu Metoda.

Definování POST akce metody 4D s použitím Claris Home Page:

1. Vyberte položku **Object Editor** z nabídky View. Objeví se následující dialogové okno:



2. Vyberte **POST** z rozevírací nabídky.

3. Vložte "/4DMETHOD/Název_vaší_metody" jako **Form Action**.

Metoda projektu

Metoda projektu GO_MAIN_MENU_BAR je ukázána zde:

V tomto příkladu má tato metoda pouze jeden účel; opustit platnou výchozí domovskou stránku zobrazenou na Web prohlížeči a poslat platné záhlaví nabídek 4D. 4D přepne na záhlaví nabídek #1.

A je to! V pěti minutách, navržením Web stránky a přidáním metody projektu GO_MAIN_MENU_BAR, máte databázi a Web server který kombinuje možnosti klient/server s HTML vývojem.

Kam jít nyní?

- Pro informace o nastavení Web serveru si přečtěte Web služby, Web server nastavení
- K nastavení vašeho stroje a databáze pro publikování na Webu si přečtěte Web služby, Nastavení.
- Pro informace o zahrnutí HTML si přečtěte Web služby, Zahrnutí HTML a JavaScriptu.

Dále si přečtěte

SEND HTML FILE, SET HTML ROOT, SET WEB TIMEOUT, START WEB SERVER, STOP WEB SERVER.

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Web služby, Bezpečnost připojení

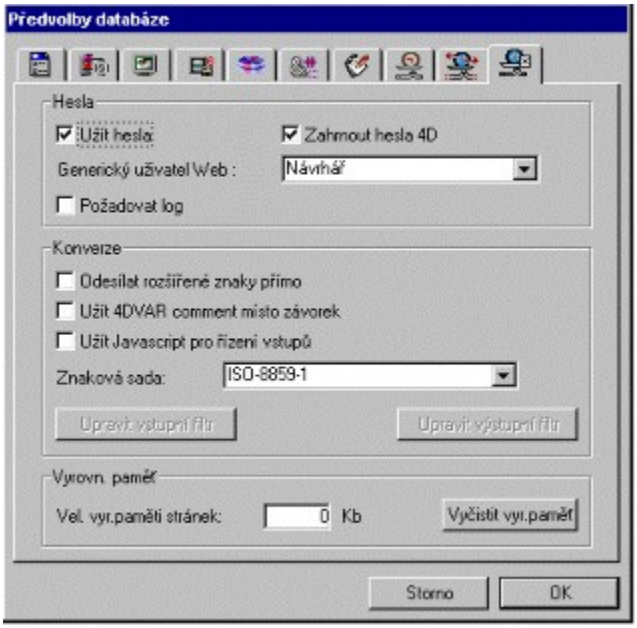
Příkazy a odkazy pro Web server

Verze 6.5

Bezpečnost vašeho Web serveru 4D je založena na následujících prvcích:
• kombinaci možnosti používat při přihášení heslo, zapínané v předvolbách databáze, a metody databáze Při ověření WEB.
• definování Generického uživatele Web.
• definování Výchozí cesty HTML.

Systém řízení hesel pro přístup Web

V Předvolbách databáze můžete definovat systém přístupu který chcete použít na váš Web server. K tomu v Předvolbách databáze přejděte na stránku **Web server II**. Zobrazí se následující okno:



V oblasti "Heslo" jsou pro vás dvě přístupné možnosti: **Užít hesla** a **Zahrnout hesla 4D**. Druhé zaškrtávací políčko je dostupné pouze když je zaškrtnuto první.

• **Užít hesla**: aktivuje systém hesel Web serveru. Pro každé spojení se na Web prohlížeči objeví dialogové okno kam může uživatel zadat své jméno a heslo. Obě tyto hodnoty, stejně jako parametry připojení (IP adresa a port,URL...) jsou poslány do metody databáze Při ověření WEB a proto s nimi můžete dále pracovat.

Poznámka: V tomto případě, jestliže metoda databáze Při ověření WEB neexistuje, spojení je odmítnuto.

• **Zahrnout hesla 4D**: vám umožní použít, místo nebo spolu s vaším systémem hesel, systém hesel 4D (jak je definovaný ve 4D).

Přehled systému přístupu Web serveru 4D

Systém který filtruje připojení k Web serveru 4D záleží na kombinaci dvou parametrů:
• Nastavení hesel pro Web v Předvolbách databáze,
• Existenci metody databáze Při ověření WEB.

Zde jsou rozdílné výsledné systémy:

Žádná označená možnost

• Pokud metoda Při ověření WEB existuje, je spuštěna, vedle $1 a $2 je vrácena pouze IP adresa prohlížeče a serveru ($3 a $4), jméno uživatele a heslo ($5 a $6) zůstanou prázdné. V tomto případě můžete filtrovat připojení podle IP adresy prohlížeče a/nebo IP adresy serveru.
• Pokud metoda databáze Při ověření WEB neexistuje, je připojení automaticky přijato.

Je označena vlastnost "Užít hesla", ale není označena vlastnost "Zahrnout hesla 4D"

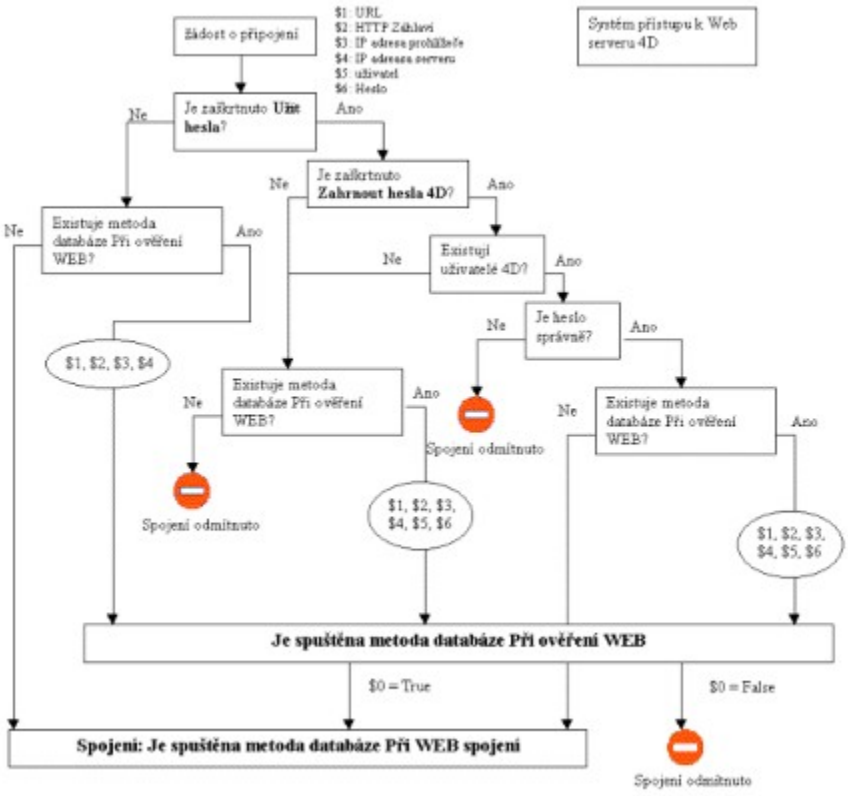
• Pokud metoda datbáze Při ověření WEB existuje, je spuštěna a všechny její parametry jsou vráceny. Proto můžete podrobněji filtrovat vstup podle jména uživatele, hesla a/nebo IP adresy prohlížeče nebo Web serveru.
• Pokud metoda databáze Při ověření WEB neexistuje, je spojení automaticky odmítnuto a na prohlížeč je poslána zpráva, že metoda ověření neexistuje.

Poznámka: Pokud je jméno uživatele prázdný řetězec a metoda databáze Při ověření WEB neexistuje, je na prohlížeč posláno okno hesel.

Jsou zaškrtnuty možnosti "Užít hesla" a "Zahrnout hesla 4D"

• Pokud zadané jméno uživatele existuje v seznamu uživatelů 4D a heslo je správně, je spojení provedeno. Pokud je heslo špatně, je spojení odmítnuto.
• Pokud jméno uživatele, poslané z prohlížeče neexistuje ve 4D, jsou možné dva výsledky:
- Pokud existuje Metoda databáze Při ověření WEB, jsou parametry $1, $2, $3, $4, $5 a $6 vráceny. Pak můžete filtrovat spojení podle jména uživatele, hesla a/nebo IP adresy prohlížeče nebo Web serveru.
- Pokud Při ověření WEB neexistuje, je spojení odmítnuto.

Systém přístupu Web serveru 4D je zobrazen v následujícím obrázku:



Bezpečnostní poznámka o robotech

Některé roboty (vyhledávací programy, spider..) procházejí přes Web servery a statické stránky. Pokud chcete aby měly přístup na váš server, můžete definovat, které URL nemají zpřístupněné.

K tomu umístěte soubor ROBOTS.TXT na cestu serveru. Soubor musí být strukturován následujícím způsobem:

User Agent: <Jméno>

Disallow: <URL> nebo <počátek URL>

Například:

User Agent: *

Disallow: /4D

Disallow: /%23%23

Disallow: /GIFS/

"User Agent: *" znamená, že platí pro všechny roboty

"Disallow: /4D" znamená že roboty nemají přístup na URL začínající /4D.

"Disallow: /%23%23" znamená že roboty nemají přístup na URL začínající /%23%23

"Disallow: /GIFS/" znamená že roboty nemají přístup ke složce /GIFS/ a jejím podsložkám.

Další příklad:

User Agent: *

Disallow: /

V tomto případě nemají roboty vůbec přístup k počítači Web serveru.

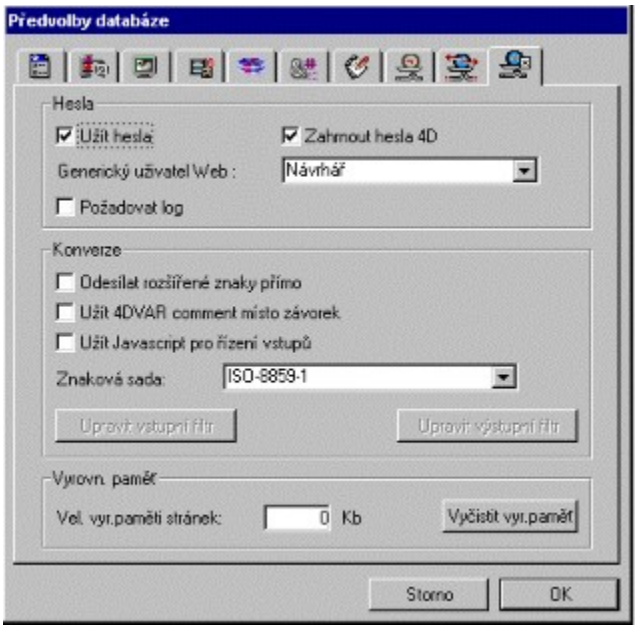
Generický uživatel Web

Můžete definovat uživatele, předtím definovaného v systému hesel 4D, jako "Generického uživatele Web". V tomto případě, **každý** prohlížeč připojený k databázi **použije** přistupová práva a omezení přiřazená tomuto generickému uživateli. Proto můžete jednoduše kontrolovat přístup Web prohlížeče k různým částem databáze.

Poznámka: Nespleťte si tuto vlastnost, která umožňuje omezit přístup prohlížeče k různým částem databáze (tabulkám, nabídkám, atd.) se systémem kontroly připojení k Web serveru, řízené systémem hesel a metodou databáze Při ověření WEB.

K definování generického uživatele Web:
1. V prostředí návrháře v Editoru hesel vytvořte uživatele.
Pokud chcete, můžete k němu přiřadit heslo.
2. V různých editorech 4D umožněte nebo zakažte přístup tohoto uživatele.
3. V okně Předvolby databáze přejděte na stránku "Web server II".

Objeví se následující okno:



Jako výchozí je nastaven Návrhář jako Generický uživatel a prohlížeč má plný přístup do databáze.

4. Vyberte uživatele ze seznamu "Generický uživatel Web" a potvrďte dialogové okno. Všechny Web prohlížeče které se připojí k databázi budou mít přistupová práva a omezení tohoto uživatele (kromě situace kdy není zaškrtnuta možnost "Zahrnout hesla 4D" a uživatel který se připojí není v seznamu uživatelů 4D, to viz dále).

Vzájemné ovlivňování systému hesel Web serveru

Vlastnost "Užít hesla" neovlivňuje funkci generického uživatele Web. Ať už je stav této vlastnosti jakýkoli, přístupová omezení a práva přiřazená ke "Generickému uživateli Web" jsou použita na všechny připojené prohlížeče.

Pokud je označena možnost "Zahrnout hesla 4D", jsou možné dva výsledky:

• Jméno uživatele a heslo neexistuje v tabulce hesel 4D. V tomto případě, pokud bylo spojení přijato metodou databáze Při ověření WEB, budou pro prohlížeč použita přístupová práva Generického uživatele Web.
• Pokud jméno a heslo uživatele existuje v tabulce hesel 4D, parametr "Generický uživatel Web" je ignorován. Uživatel se připojí s vlastními přístupovými právy.

Definování cesty ke složce HTML

Tato možnost v Předvolbách databáze vám umožní nastavit složku, ve které bude 4D hledat statické HTML stránky a obrázky k poslání na prohlížeč.

Navíc, cesta ke složce HTML definuje hierarchickou úroveň, nad kterou soubory nebudou přístupné.

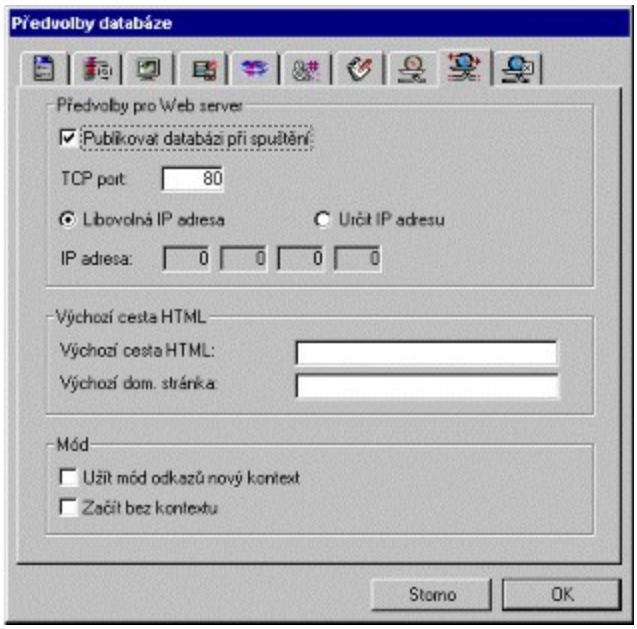
Toto omezení přístupu platí jak pro URL posílané z prohlížeče, tak pro příkazy Web serveru 4D, jako SEND HTML FILE. Pokud se URL poslané z prohlížeče nebo příkaz pokusí otevřít soubory uložené nad složkou HTML, bude vrácena chyba značící že soubor nebyl nalezen.

Jako výchozí je složka HTML stejná jako složka struktury. **Buďte opatrní, protože v tomto případě nejsou žádná omezení přístupu** (uživatelé budou mít přístup na všechny jednotky).

K nastavení výchozí HTML složky:

1. V dialogovém okně Předvolby databáze přejděte na stránku Web server I.

Objeví se následující stránka:



2. V dostupné oblasti "Výchozí cesta HTML", zadejte cestu ke složce kterou chcete definovat.

Cesta přístupu předáná do tohoto okna je relativní; je ustanovena ze složky obsahující strukturu databáze. Pro zachování kompatibility vaší databáze mezi platformami, Web server 4D používá následující koncence zápisu pro zapsání cesty. Pravidla zápisu jsou následující:

• složky jsou odděleny lomítkem ("/")
• cesta přístupu končí lomítkem ("/")
• k přechodu v hierarchii složek o jednu úroveň výše, zadejte ".." (dvě tečky) před název složky
• cesta přístupu nemusí začínat lomítkem ("/") (kromě pokud nechcete složku databáze jako složku HTML, čtěte dále)

Pokud například chcete mít HTML cestu k podsložce "Web" umístěné ve složce "4Ddatabáze", předáte 4DDatabáze/Web/

Pokud chcete mít složku databáze i složkou HTML, ale mít zakázaný přistup k vyšší úrovni, vložte do zadávací oblasti "/" . Pro neomezený přístup na všechny jednotky nechte tuto oblast prázdnou.

3. Potvrďte dialogové okno.

Poznámka: Jakmile je v Předvolbách databáze změněna cesta ke složce HTML, cache je vymazána tak, že nebude obsahovat soubory ke kterým již není přístup.

Předvolby databáze a SET HTML ROOT

Jakmile je složka HTML nastavena, můžete jí kdykoli změnit použitím příkazu SET HTML ROOT. Úprava je platná pouze pro Web proces pracovní stanice, pro který byla spuštěna. Cache HTML stránek je pak vyčištěna.

Příkaz SET HTML ROOT bere však do úvahy složku Výchozí HTML cesty pokud je definována v Předvolbách databáze. Jestliže bude složka definovaná v Předvolbách databáze "/WebStr" a vy předáte instrukci SET HTML ROOT("Složka"), bude výchozí HTML složka "WebStr/Složka/". V tomto případě můžete ovládat pouze složky umístěné ve složce "WebStr/".

Poznámka: Příkaz SET HTML ROOT neprovede nic pokud je Web server v nekontextním módu. (Více informací najdete v části Web služby, Nekontextní mód).

Dále si přečtěte

Metoda databáze Při ověření WEB, Metoda databáze Při Web spojení.

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Metoda databáze Při ověření WEB

Příkazy a odkazy pro Web server

Verze 6.5

Metoda databáze Při ověření WEB je pro ovládání přístupu k Web serveru. Je provedena pokaždé, když se prohlížeč připojí k databázi.

Tato metoda přijímá šest parametrů: $1, $2, $3, $4, $5 a $6. Obsah těchto parametrů je následovný:

Parametr	*Typ*	*Popis*
$1	*Text*	*URL*
$2	*Text*	*HTTP záhlaví*
$3	*Text*	*IP adresa klienta (prohlížeče)*
$4	*Text*	*IP adresa serveru*
$5	*Text*	*Jméno uživatele*
$6	*Text*	*Heslo*

Parametry musíte deklarovat následovně:

```
` Metoda databáze Při ověření WEB
C_TEXT ($1;$2;$3;$4;$5;$6)
` Kód metody
```

Poznámka: Všechny parametry nemusí být naplněny. Informace, které obdrží metoda jsou závislé na nastavení, které provedete v Předvolbách databáze (Přečtěte si kapitolu Web služby, Bezpečnost připojení).

• **URL**

První parametr ($1) je URL zadané uživatelem do oblasti Adresa v prohlížeči, zkrácené o adresu hostitele.

Vezměme si příklad Intranet spojení. Předpokládejme že IP adresa stroje Web serveru 4D je 123.4.567.89. Následující tabulka ukazuje hodnotu $1 podle URL zadaného do prohlížeče:

URL zadané do oblasti Adresa v Web prohlížeči	*Hodnota parametru $1*
123.4.567.89	*/*
http://123.4.567.89	*/*
123.4.567.89/Zákazníci	*/Zákazníci*
http://123.4.567.89/Zákazníci	*/Zákazníci*
123.4.567.89/Zákazníci/Přidat	*/Zákazníci/Přidat*
http://123.4.567.89/Provést/Pokud_OK/Něco	*/Provést/Pokud_OK/Něco*

Poznámka: Jestli chcete vědět více informací o tomto parametru, přečtěte si kapitolu Metoda databáze Při Web spojení.

• **Záhlaví HTTP žádosti**

Druhý parametr ($2) je záhlaví HTTP žádosti poslané Web prohlížečem. Nezapomeňte že toto záhlaví je předáno metoda databáze Při ověření WEB tak jak je. Jeho obsah je závislý na typu prohlížeče který provádí připojení.

Pokud vaše aplikace pracuje s těmito informacemi, je pouze na vás jak využijete záhlaví.

Poznámka: Jestli chcete vědět více informací o tomto parametru si přečtěte popis k metodě databáze Při Web spojení.

• **IP adresa Web klienta**

Parametr $3 bude obsahovat IP adresu stroje prohlížeče. Tato informace vám umožní rozlišit spojení Internet a Intranet.

• **IP adresa serveru**

Parametr $4 bude obsahovat IP adresu Webserveru 4D. 4D verze 6.5 vám umožňuje "multi-homing", který umožní využívat stroj pro více než jednu IP adresu. Podrobnější informace najdete v kapitole Web služby, Web server nastavení.

• **Jméno uživatele a heslo**

Parametry $5 a $6 obsahují jméno uživatele a heslo zadané uživatelem do standardního okna hesla zobrazeného prohlížečem. Jestliže je zaškrtnuta možnost **Užít hesla**, toto okno se zobrazí pro každé spojení (podívejte se na Web služby, Bezpečnost připojení).

Poznámka: Pokud jméno uživatele existuje ve 4D, parametr $6 (heslo uživatele) není vrácen z důvodů bezpečnosti.

Metoda databáze Při ověření WEB vrací do $0 logickou hodnotu:

• pokud je $0 True, spojení je přijato.
• pokud je $0 False, spojení je odmítnuto.

Metoda databáze Při Web spojení je provedena pouze pokud bylo spojení metodou databáze Při ověření WEB přijato.

UPOZORNĚNÍ: Nevolejte prvky rozhraní v metodě databáze Při ověření WEB (ALERT, DIALOG, atd.), jinak bude metoda přerušena a spojení bude odmítnuto. Stejná věc se stane pokud se vyskytne chyba při provádění metody databáze.

Systém který filtruje spojení s Web serverem je závislý na kombinaci možností Web hesla v Předvolbách databáze a metodě databáze Při ověření WEB. Tato část je popsaná v kapitole Web služby, Bezpečnost připojení.

Příklad

Zde je typický příklad metody databáze Při ověření WEB, která filtruje připojení pomocí tabulky Uživatelů a hesel.

```
   ` Metoda databáze Při ověření WEB
C_TEXT($5;$6;$3;$4)
C_TEXT($user;$password;$BrowserIP;$ServerIP)
C_BOOLEAN($4Duser)
ARRAY TEXT($users;0)
ARRAY TEXT($nums;0)
C_LONGINT($upos)
C_BOOLEAN($0)
$0:=False
$user:=$5
$password:=$6
$BrowserIP:=$3
$ServerIP:=$4
   `Z důvodů bezpečnosti odmítnou jména obsahující @
If (SeZavináčem($user) | SeZavináčem($password))
   $0:=False
      `Metoda SeZavináčem je popsána dále
Else
      `Podívat se, jestli je to uživatel 4D
   GET USER LIST($users;$nums)
   $upos:=Find in array($users;$user)
```

```
      If ($upos > 0)
         $4Duser:=Not(Is user deleted($nums{$upos}))
      Else
         $4Duser:=False
      End if
      If (Not($4Duser))
            `Není to uživatel 4D, podívat se do tabulky uživatelů Web
         QUERY([WebUzivatelé];[WebUzivatelé]Uživatel=$user;*)
         QUERY([WebUzivatelé]; & [WebUzivatelé]Heslo=$password)
         $0:=(Records in selection([WebUzivatelé]) = 1)
      Else
         $0:=True
      End if
   End if
      `Je to intranet spojení?
   If (Substring($BrowserIP;1;7) # "192.100.")
      $0:=False
   End if
```

Zde je metoda projektu SeZavináčem:

```
      ` Metoda SeZavináčem
      ` SeZavináčem ( Řetězec ) → Logické
      ` SeZavináčem ( Jméno ) → Obsahuje znak zavináč
   C_INTEGER($i)
   C_BOOLEAN($0)
   C_TEXT($1)
   $0:=False
   For($i;1;Length($1))
      If (Ascii(Substring($1;$i;1)) = Ascii("@"))
         $0:=True
      End if
   End for
```

Dále si přečtěte

Metoda databáze Při Web spojení, Textové parametry předávané do 4D Metod volaných přes URL, Web služby, Bezpečnost připojení.

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Web služby, Procesy Web spojení

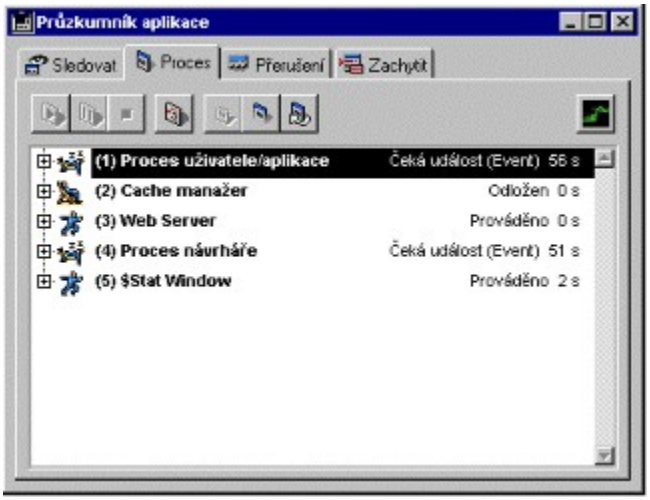
Příkazy a odkazy pro Web server

Verze 6.0

Proces Web server

Proces **Web server** běží a je prováděn od chvíle kdy je databáze publikována na Webu.

Na straně **Procesy** v Průzkumníku provádění, ukázaném zde, je Web server třetí proces který byl spuštěn:



Je to základní proces 4D: nemůžete jej odstranit pomocí tlačítka **Odstranit**. Nemůžete na něj použít meziprocesní komunikaci jako CALL PROCESS. Nezapomeňte že proces Web server nemá žádné prvky rozhraní (okna, nabídky, atd.).

Proces Web server můžete zapnout následujícími způsoby:
• Vybrat **Start Web Server** z nabídky **Web Server** v prostředí uživatele.
• Provést příkaz START WEB SERVER.
• Otevřít databáze s označenou možností **Publikovat databázi při spuštění** v Předvolbách databáze.

Proces Web server můžete zastavit následujícími způsoby:
• Vybrat položku **Stop Web Server** z nabídky **Web server** v Prostředí uživatele.
• Provést příkaz STOP WEB SERVER.
• Vypnout právě publikovanou databázi.

Proces Web server má za účel pouze ovládat pokusy o připojení k databázi. Zapnutí procesu Web Server neznamená že otevřete spojení, znamená to, že pouze umožňujete Web uživatelům navázat spojení. Zastavení procesu Web Server neznamená, že přerušíte platné procesy Web spojení (pokud jsou nějaké), znamá to, že již nechcete další připojené uživatele.

Pokud zastavíte proces Web Server ve chvíli, kdy jsou nějaké procesy Web spojení, pokračují tyto procesy dále, dokud Web uživatel nepřestane posílat dotazy nebo dokud se neozve déle než čas zadaný pro přerušení spojení (nastavené v okně Předvolby databáze, nebo příkazem SET WEB TIMEOUT). Proto čas pozdržení může být nutný pro přerušení procesu Web server.

Procesy Web spojení

Pokaždé když se prohlížeč připojí k databázi, žádost je řízena procesem Web Server, který provede následující kroky:

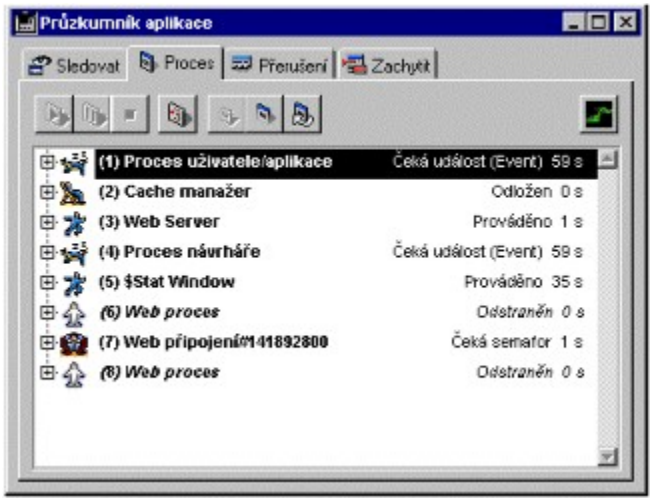
• Zaprvé vytvoří jeden nebo několik místních dočasných procesů 4D k řízení spojení s Web prohlížečem.

Poznámka: Tyto dočasné procesy řídí každý HTTP dotaz. Jsou rychle spuštěny a pak odstraněny nebo zastaveny. Aby Web server mohl reagovat v nekontextním módu, 4D pozdrží Web proces na 5 sekund a pak jej znovu použije na nějaký další HTTP dotaz. Toto chování můžete upravit příkazem SET DATABASE PARAMETER.

• Jestliže dotaz vyžaduje založení kontextního módu, otestuje se jestli jsou nějaké volné zdroje pro nové spojení. Jestliže to není případ, pošle na Web prohlížeč následující zprávu: "Tato databáze nebyla navržena pro Web."

Pokud je Web spojení správně založeno, pak se zapne **Proces Web spojení**. Je to proces, který bude ovládat vnitřní stránku Web spojení.

Následující seznam procesů ukazuje proces Web spojení "Web spojení#141892800," zapnutý po připojení Web prohlížeče:



Všimněte si odstraněného šestého a osmého procesu, které byly zapnuty a zastaveny procesem Web server; tyto procesy ovládají inicializaci Web spojení.
Poznámka: Jestli chcete vědět více informací o řízení kontextu přečtěte si následující část "Řízení kontextu Web připojení: Kontextní mód".
• Pokud žádost nevyžaduje založení kontextního módu (například databáze je zapnuta v nekontextním módu), není založen kontext; žádost je vyřízena a případná odpověď je poslána na Web prohlížeč. Dočasný proces je pak odstraněn nebo pozdržen (čtěte dále).

• Pokud během sekce je spojení přepnuto z kontextního na nekontextní mód, proces Web spojení je odstraněn.

Stejně tak pokud se spojení přepne z nekontextního na kontextní mód, je založen číslovaný proces Web spojení.

Poznámka: Jestli chcete vědět více informací o nekontextním módu přečtěte si kapitolu Web služby, Nekontextní mód.

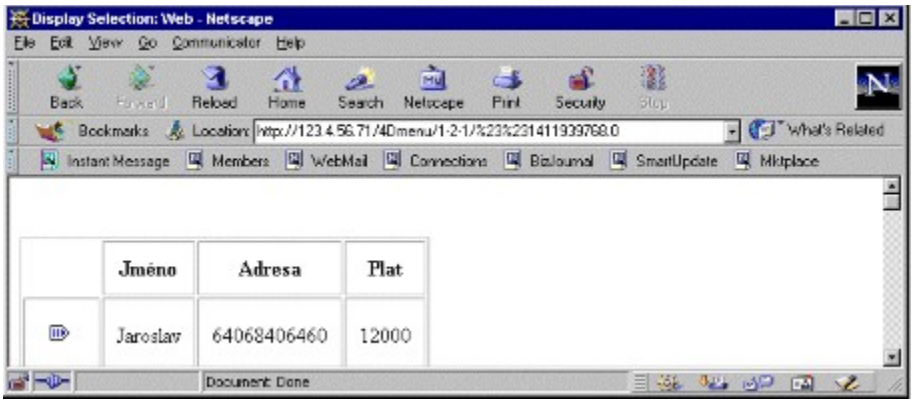
Řízení kontextu Web připojení: Kontextní mód

ID kontextu Web spojení

Číslo obsažené v názvu procesu Web spojení je nazýváno **ID kontextu** a je vytvářeno náhodně a jedinečně definuje každé Web spojení. ID kontextu je během spojení udržováno jak ve 4D tak na prohlížeči. V tomto případě

je ID kontextu 231411939768.

V následujícím okně Web prohlížeče můžete vidět toto číslo v URL zobrazeném v oblasti Adresa/Location:



URL jsou automaticky ovládány 4D během celé práce v kontextním módu. Pokaždé když TCP/IP síťové komponenty 4D obdrží žádost HTTP, 4D získá z URL ID kontextu a proto může žádost směřovat do správného Web spojení.

ID kontextu:
• Umožňují 4D ovládat jak část Web tak Databáze pro každé Web spojení.
• Průhledně umožní ovládat více Web spojení.
• Zabrání budoucím nežádoucím spojením při použití kopií stránek (typu bookmark, oblíbené), protože pro každé spojení je vytvořeno rozdílné ID kontextu.

Synchronizace sekcí Web a Databáze: ID podkontextu Web spojení

V obrázku nahoře si všiměte, že ID kontextu je následováno tečkou a jiným číslem. Toto číslo se nazývá **ID podkontextu**. 4D automaticky mění tato čísla pokaždé, když je na prohlížeč poslána další HTML stránka založená na 4D v kontextním módu. ID podkontextu je důležité k řízení sekce databáze.

Obvykle prohlížeč obsahuje navigační tlačítka jako Zpět, Dopředu, atd. Tato tlačítka jsou použitelná pokud prohlížíte dokumenty, zprávy, aj. Méně používané jsou provádění transakcí databáze.

Pokud například Web uživatel zadává nový záznam, musíte si být jisti že byl vstup potvrzen, to je, že uživatel ve formuláři 4D klepne na tlačítko Storno nebo Přijmout. Pokud uživatel během zadávání použije navigační tlačítka, zůstane vstup dat nedokončený. Aby tomu zabránila, používá 4D ID podkontextu k sysnchronizování Web sekce na straně prohlížeče se sekcí databáze na straně 4D.

Pokaždé když je přijmut formulář nebo je poslána z prohlížeče žádost na 4D, proběhne kontrola a pokud je nějaká nesynchronizace mezi Web a databází, 4D pošle zprávu "*S použitím navigace prohlížeče jste opustili formulář vyžadující potvrzení dat. 4D server se nyní vrátí do toho formuláře a vy jej můžete potvrdit nebo zrušit.*" Pomocí ID podkontextu pak 4D přejde na stránku vstupu dat.

Tato synchronizace je také důležitá pro procesy Web spojení. Například pro správné pokračování Vašeho kódu potřebujete správně odejít z, ADD RECORD([...])...

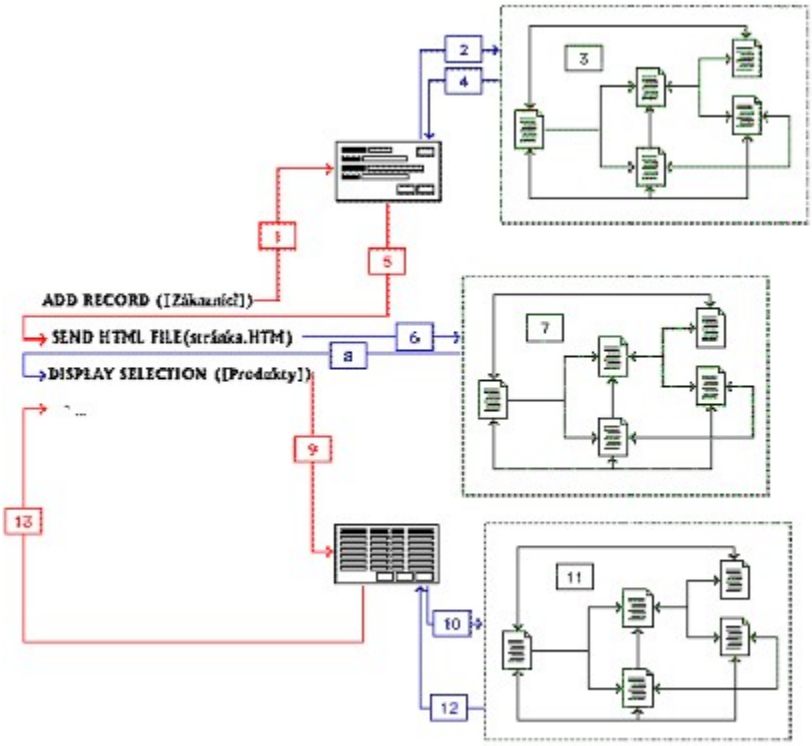
Synchronizace je výběrová. Pokud je Web stránka zobrazená na prohlížeči formulář 4D (ADD RECORD, DISPLAY SELECTION, DIALOG, atd.), synchronizace se použije.

Pokud je platná Web stránka HTML stránka poslaná z jiné Web stránka (poslaná příkazem SEND HTML FILE), pak můžete použít navigaci po stránkách, synchronizace se nepoužije.
Vezměme si následující výsek kódu:

```
ADD RECORD ([Zákazníci])
SEND HTML FILE ("anyPage.HTM")
```

DISPLAY SELECTION ([Produkty])

Následující obrázek ukazuje co se bude dít na straně Web a na straně 4D během provádění.
• Čáry červeně znamenají překlad a odevzdání formuláře 4D.
• Modré čáry značí přepnutí mezi 4D HTML stránkami a normálními HTML stránkami.
• Oblasti v zelené označují HTML stránky založené 4D.



Popis kroků
1. Příkaz ADD RECORD. 4D přeloží platný vstupní formulář tabulky do HTML stránky a pošle jej na Web prohlížeč. Pokud je formulář vícestránkový, standardní navigační tlačítka ve formuláři vám umožní posunovat se po stránkách formuláře. Tato 4D navigace je implementována a používána 4D Web Serverem (jako potvrzení Web formuláře).
2. Během vstupu dat (zde ve volání ADD RECORD), je klepnuto na tlačítko a jeho metoda objektu pošle SEND HTML FILE.

3. Ve volání SEND HTML FILE, jestliže HTML stránka obsahuje odkazy, je možné listovat ve více stránkách. Pokud je předáno SEND HTML FILE (""), HTML mód je opuštěn.

4. Metoda objektu tlačítka které bylo stisknuto a pokračuje se ve vstupu dat byl započatém příkazem ADD RECORD. Všiměte si, že kroky 2 a 3 se mohou několikrát opakovat.

5. Nakonec je vstup dat přijmutý nebo stornovaný a proces Web spojení pokračuje.

6. Další je provedení SEND HTML FILE.

7. Tento krok je podobný jako krok 3. Pokud HTML stránka obsahuje odkazy na jiné stránky můžete se pohybovat po několika stránkách. Pokud je provedeno SEND HTML FILE(""), HTML mód je opuštěn.

8. Proces Web spojení pokračuje.

9. Je proveden příkaz DISPLAY SELECTION. 4D přeloží platný výstupní formulář do HTML stránky a pošle jej na Web prohlížeč. Během DISPLAY SELECTION, 4D umožňuje pohyb mezi zobrazenou stránkou se seznamem a zobrazením jednotlivých záznamů. 4D také může použít MODIFY SELECTION k řízení vstupu dat a zamykání záznamů, a pro úpravy záznamů pomocí potvrzení Web formuláře.

10. Během pohybování se po výběru, je klepnuto na tlačítko v zápatí a jeho metoda objektu provede SEND HTML FILE.

11. Tento krok je podobný jako kroky 7 a 3. Pokud HTML stránka obsahuje odkazy na jiné stránky můžete se pohybovat po více stránkách. Pokud je provedeno SEND HTML FILE(""), HTML mód je opuštěn.

12. Metoda objektu tlačítka které bylo stisknuto pokračuje a je proveden návrat k zobrazení výběru předtím započaté příkazem DISPLAY SELECTION. Všimněte si, že kroky 10 a 11 se mohou při pohybu ve výběru několikrát opakovat.

13. Nakonec je zobrazení výběru opuštěno a proces Web spojení je pokračuje.

A tak dále...

Volná navigace po Webu (například klepnutí na Zpět a Dopředu) je možné v SEND HTML FILE (zelené oblasti v obrázku nahoře). Na druhou stranu, pokud jsou jakékoli HTML stránky založené na 4D (vstup dat, zobrazení výběru.. obsahující standrdní dialogová okna CONFIRM nebo Request) opuštěny pomocí navigace Web prohlížeče, 4D bude synchronizovat sekci Web a sekci databáze tím, že se vrátí zpět na Web stránku jejíž ID podkontextu odpovídá tomu, které bylo poslané příkazem 4D spuštěným na straně procesu Web spojení.

Proces Web spojení a Web sekce

Z pohledu uživatele ovládají akce na Web prohlížeči sekci Web. Z pohledu programu, proces Web spojení ovládá Web sekci a ne obráceně. Web prohlížeč zobrazí stránky poslané procesem Web spojení, které:

• Provede kód 4D nebo
• Počká na potvrzení z platné Web stránky prohlížeče.

Z pohledu návrháře, může proces Web spojení vypadat jako proces 4D jehož doménou spuštění je 4th Dimension nebo 4D Server, ale jehož rozhraní uživatele je na Web prohlížeči.

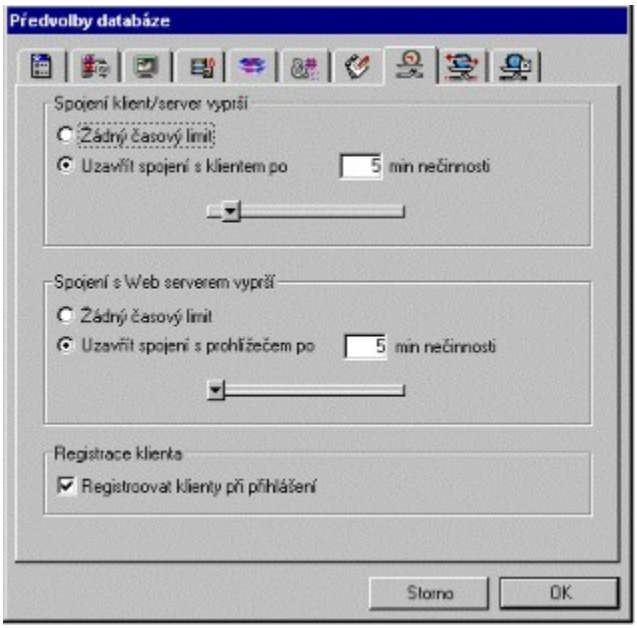
Při navrhování aplikace Web databáze v kontextním módu, mějte vždy na paměti tuto dvojakost procesu Web spojení. Například

• Během vstupu dat je hlavní nabídka Web prohlížeče a ne 4D. Ve formuláři 4D se nespoléhejte na nabídky 4D; jsou na stroji Web serveru ne na prohlížeči.
• Při navrhování formulářů pro Web prohlížeč, nezapomeňte že nastavení formuláře 4D je omezeno možnostmi HTML (a někdy vlastnostmi 4D při převodu). Nespoléhejte se pouze na nastavení formuláře 4D (tzn. typy objektů a události formuláře). Jestli chcete vědět více informací, přečtěte si část Web služby, Zahrnutí HTML a JavaScriptu.

• Meziprocesní komunikace. Pokud je CALL PROCESS použit na proces Web spojení, nemá žádná efekt, protože jeho aktivní okno je na Web prohlížeči. Na druhou stranu však může proces Web spojení použít CALL PROCESS pro volání jiných procesů 4D a zde může být meziprocesní komunikace použita dokonce oboustranně s použitím příkazů GET PROCESS VARIABLE a SET PROCESS VARIABLE, které nevyžadují prvky s rozhraním uživatele.

Časovač Web spojení

Jak bylo ukázáno dříve, proces Web spojení provádí kód 4D nebo čeká na potvrzení stránky zobrazené na prohlížeči. V druhém případě, proces Web spojení bude čekat na potvrzení delší nebo stejně dlouhý čas jaký je zadaný v Předvolbách databáze **Spojení s Web serverem vyprší za** (zobrazeno níže) nebo nastavený programově příkazem SET WEB TIMEOUT.



Rozsah platnosti nastavení časovače spojení Web serveru je sekce databáze. Všechny procesy Web spojení jsou nastaveny na tuto hodnotu; jako výchozí je nastaveno 5 minut.

Poznámka: Příkaz SET WEB TIMEOUT vám umožňuje nastavit rozdílné hodnoty pro každý proces.

Tuto hodnotu můžete zvětšit nebo zmenšit podle vašich požadavků. Například můžete tuto hodnotu zvětšit pokud vaše aplikace umožňuje Web uživatelům prohlížet jiné Web stránky jako HTML odkazy na stranách posílaných vaší databází. Prodloužením časového limitu umožníte uživatelům prohlížet déle tyto jiné stránky bez přerušení spojení s vaší databází.

UPOZORNĚNÍ: Není způsob jak zastavit proces Web spojení programově. Pokud nastavíte dlouhý časový limit, proces bude čekat i když uživatel skončí práci s Web spojením po určeném čase. Pokud nedefinujete žádný limit, proces Web spojení je zrušen pouze když skončí databáze. Proces Web spojení je však automaticky ukončen když je Web server přepnut na nekontextní mód.

Tip: Narozdíl od procesu Web server, může být proces Web spojení přerušen příkazem Abort (dostupným v Průzkumníku provádění na straně Procesy).

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Metoda databáze Při Web spojení

Příkazy a odkazy pro Web server

Verze 6.0

Metoda databáze Při Web spojení je volána 4th Dimension nebo 4D Serverem pokaždé, když Web prohlížeč začne spojení s databází v kontextním módu nebo když je potřeba založení kontextu (přečtěte si další sekci "Volání metody databáze Při Web spojení").

Spojení samozřejmě musí být nejdříve přijato metodou databáze Při ověření WEB (pokud existuje) a databáze musí být publikována na Webu.

Metoda databáze Při Web spojení přijímá šest parametrů. Obsah těchto parametrů je následující:

Parametr	***Typ***	***Popis***
$1	*Text*	*URL*
$2	*Text*	*HTTP záhlaví*
$3	*Text*	*IP adresa klienta (prohlížeče)*
$4	*Text*	*IP adresa serveru*
$5	*Text*	*Jméno uživatele*
$6	*Text*	*Heslo*

Parametry musíte deklarovat následovně:

```
` Metoda databáze Při Web spojení
C_TEXT ($1;$2;$3;$4;$5;$6)
` Kód metody
```

• **URL**

První parametr ($1) je URL zadané uživatelem do oblasti Adresa prohlížeče, ze které je odstraněna adresa hostitele.

Vezměme si příklad Intranet spojení. Předpokládejme že IP adresa stroje Web serveru 4D je 123.4.567.89. Následující tabulka ukazuje hodnotu $1 podle URL zadaného do prohlížeče:

URL zadané do oblasti Adresa v Web prohlížeči	***Hodnota parametru $1***
123.4.567.89	*/*
http://123.4.567.89	*/*
123.4.567.89/Zákazníci	*/Zákazníci*
http://123.4.567.89/Zákazníci	*/Zákazníci*
123.4.567.89/Zákazníci/Přidat	*/Zákazníci/Přidat*
http://123.4.567.89/Provést/Pokud_OK/Něco	*/Provést/Pokud_OK/Něco*

Použití tohoto parametru je pouze na vás. 4D ignoruje hodnotu předánou do URL za částí hostitele.

Například chcete stanovit konvenci kde hodnota "/Zákazníci/Přidat" znamená "přidat nový záznam do tabulky [Zákazníci]." Web uživateli můžete předat seznam možných hodnot a/nebo výchozí odkazy na elektronických dokumentech (typu bookmark či oblíbené položky), kam můžete vložit zkratky k rozdílným částem vaší aplikace. Tímto způsobem můžete Web uživateli rychle zpřístupnit části vaší Web stránky bez toho aby musel projít celou přistupovou cestou při každém připojení k vaší databázi.

UPOZORNĚNÍ: Aby 4D zabránila uživateli znovu vstoupit do databáze pomocí odkazů vytvořených při předchozí práci, 4D odřízne jakékoli URL které odpovídá jednomu ze standardních URL 4D.

• **Záhlaví HTTP žádosti**

Druhý parametr ($2) je záhlaví HTTP žádosti poslané Web prohlížečem. Nezapomeňte že toto záhlaví je předáno do metody databáze Při ověření WEB tak jak je. Jeho obsah je závislý na typu prohlížeče který provádí připojení.
S Netscape 3.0 spuštěném na Windows NT, můžete obdržet následující záhlaví:
GET HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/3.01 (WinNT; I)
Host: 192.9.200.11
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*

S Microsoft Internet Explorer běžícím na Windows NT, můžete obdržet podobné záhlaví:
GET / HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, */*
Accept-Language: en
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0d; Windows NT)
Connection: Keep-Alive
If-Modified-Since: Sunday, 10-Dec-96 01:51:37 GMT

• **IP adresa Web klienta**

Parametr $3 bude obsahovat IP adresu stroje prohlížeče. Tato informace vám umožní rozlišit spojení Internet a Intranet.

• **IP adresa serveru**

Parametr $4 bude obsahovat IP adresu Webserveru 4D. 4D verze 6.5 vám umožňuje "multi-homing", který umožní využívat stroj na více než jedné IP adrese. Podrobnější informace najdete v kapitole Web služby, Web server nastavení.

• **Jméno uživatele a heslo**

Parametry $5 a $6 obsahují jméno uživatele a heslo zadané uživatelem do standardního okna hesla zobrazeného prohlížečem. Jestliže je zaškrtnuta možnost **Užít hesla**, toto okno se zobrazí pro každé spojení (podívejte se na Web služby, Bezpečnost připojení).

Poznámka: Pokud jméno uživatele existuje ve 4D, parametr $6 (heslo uživatele) není vrácen z důvodů bezpečnosti.

Volání metody databáze Při Web spojení

Metoda databáze Při Web spojení je prováděna pokaždé když se prohlížeč připojí k databázi. Ale také hraje roli jako vstupní bod do 4D z nového "nekontextního" módu Web serveru.

Přepnutí z nekontextního do kontextního módu je provedeno přes Metodu databáze Při Web spojení (více informací najdete v kapitole Web služby, Nekontextní mód).

Upozornění: V nekontextním módu provedení příkazů které zobrazují prvky rozhraní (ALERT, DIALOG,...) ukončí průběh.
Metoda databáze Při Web spojení je volána v následujících případech:
• při připojení Web prohlížeče k Web serveru - jako s 4D 6.0.x. Metoda databáze je volána s URL /<akce>....
• při přepnutí z nekontextního do kontextního módu. Metoda databáze je volána s URL /4DMETHOD/NázevMetody.
• když je 4D volána semidynamickou stranou přes URL /4DCGI/<akce>. Metoda databáze je volána přes URL (podívejte se na číst Web služby, Nekontextní mód).

• když je Web stránka volána z nekontextního módu a URL typu <cesta>/<soubor> nebyl nalezen. Metoda databáze je volána s URL.
• pokud je Web stránka volána v nekontextním módu s URL typu <cesta>/ a nebyla definována žádná výchozí domovská stránka. Metoda databáze je volána s URL.

Aby jste zjistili kdy je Metoda databáze Při Web spojení volána z kontextního a kdy z nekontextního módu, použijte funkci Web Context, která vrací True pokud je volána kontextního módu a jinak False.

Doporučujeme vám psát metodu databáze Při Web spojení následujícím způsobem:

```
   `Metoda databáze Při Web spojení
C_TEXT($1;$2;$3;$4;$5;$6)
If (Web Context) `Pokud kontextní mód
   WithContext ($1;$2;$3;$4;$5;$6)
      ` WithContext obsahuje všechno co bylo v
      `metodě databáze Při Web spojení v 4D 6.0.x
Else
   NoContext ($1;$2;$3;$4;$5;$6)
      ` NoContext metoda provádí nekontextní
      `provádění dotazů (obvykle krátké)
End if
```

Příklad: Předání místní domovské stránky klienta

V následujícím příkladu je parametr $1 použit k předání domovské stránky podle organizace.

Databáze má dvě tabulky; [Zákazníci] a [Tabulky]. Následující metoda databáze Při spuštění nastaví meziprocení array které je dále použité v metodě databáze Při Web spojení.

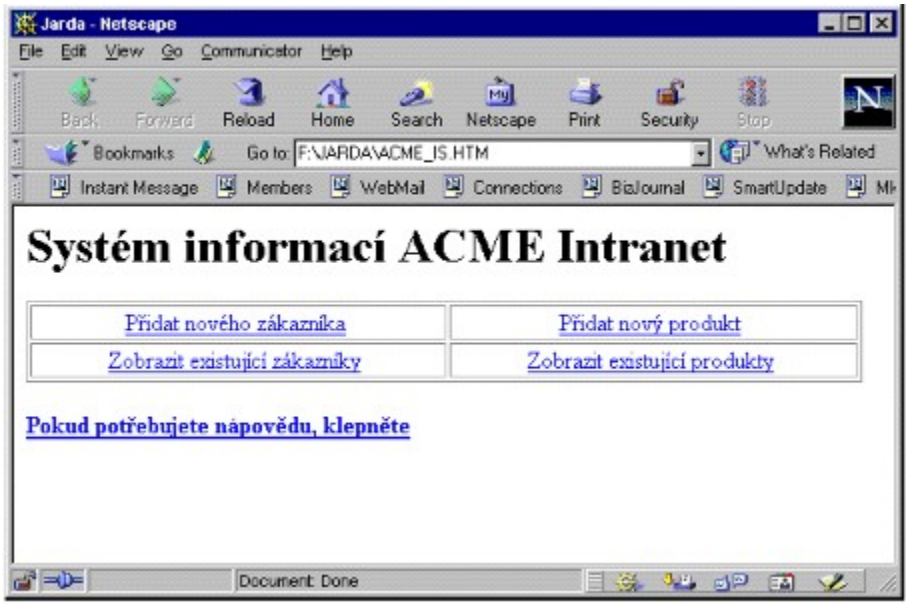
```
   ` Metoda databáze Při spuštění
   ` Seznam tabulek
ARRAY STRING(31;<>asTables;Count tables)
For ($vlTabulka;1;Size of array(<>asTables))
   <>asTables{$vlTabulka}:=Table name($vlTabulka)
End for
   ` Standardní Web akce při přihlášení
ARRAY STRING(31;<>asActions;2)
<>asActions{1}:="Add"
<>asActions{2}:="List"
```

Hlavní práce metody databáze Při Web spojení je rozluštit extra data předaná do ERL za část hosta do adresy a pracovat podle toho. Metoda je následující:

```
   ` Metoda databáze Při Web spojení
C_TEXT($1;$2;$3;$4;$5;$6)
C_TEXT($vtURL)
If (Web context) `Pokud jsme v kontextním módu
      ` Pouze v případě že $1 je rovno "/" nebo "/..."
   If ($1="/@")
         ` Kopírovat URL do místní proměnné mínus jedna "/"
      $vtURL:=Substring($1;2)
         ` Rozeberte URL a naplňte místní array částmi URL
         ` například jestli URL extra data je "aaa/bbb/ccc", výsledná array
         ` bude mít tři položky "aaa", "bbb" and "ccc" v tomto pořadí
      $vlElem:=0
```

```
        ARRAY TEXT($atTokens;$vlElem)
        While ($vtURL # "")
            $vlElem:=$vlElem+1
            INSERT ELEMENT($atTokens;$vlElem)
            $vlPos:=Position("/";$vtURL)
            If ($vlPos>0)
                $atTokens{$vlElem}:=Substring($vtURL;1;$vlPos-1)
                $vtURL:=Substring($vtURL;$vlPos+1)
            Else
                $atTokens{$vlElem}:=$vtURL
                $vtURL:=""
            End if
        End while
            ` Jestliže extra data byla předána za část HOST v URL
        If ($vlElem>0)
                ` S použitím meziprocesní array vytvořené v metodě databáze Při spuštění
                ` testuje jestli první je název tabulky
            $vlTabulkaNumber:=Find in array(<>asTables;$atTokens{1})
            If ($vlTabulkaNumber>0)
                    ` Pokud ano, vzít ukazatel na tuto tabulku
                $vpTable:=Table($vlTabulkaNumber)
                    ` nastavit vstupní a výstupní formulář
                INPUT FORM($vpTable→;"Input Web")
                OUTPUT FORM($vpTable→;"Output Web")
                    ` S použitím meziprocesní array vytvořené v MD Při spuštění
                    ` testuje jestli první je název tabulky
                $vlAction:=Find in array(<>asActions;$atTokens{2})
                Case of
                        ` Přidání záznamů
                    ÷ ($vlAction=1)
                        Repeat
                            ADD RECORD($vpTable→;*)
                        Until (OK=0)
                            ` Seznam záznamů
                    ÷ ($vlAction=2)
                        READ ONLY($vpTable→)
                        ALL RECORDS($vpTable→)
                        DISPLAY SELECTION($vpTable→;*)
                        READ WRITE($vpTable→)
                Else
                    ` Zde mohou být předány další standardní akce tabulek
                End case
            Else
                ` Zde mohou být předány další standardní akce
            End if
        End if
    End if
        ` Ať už se stalo cokoli, následuje normální Log On proces
    WWW NORMAL LOG ON
Else
    ... ` Zde může být předán kód pro nekontextní mód
End if
```

V této chvíli se mohou lidé z organizace připojit k databázi a vložit URL podle konvence nastavení v zobrazené metodě. Uživatelé mohou také vytvořit zkratky jestliže nechtěji pokaždé znovu zadávat URL. Poslední výsledek je opatřit každému členovi organizace HTML stránku kterou bude používat k přístupu do databáze. Tato HTML stránka je zde ukázána:



Jinými slovy HTML stránka ACME_IS.HTM je místní domovská stránka klienta z informačního organizace založeného na 4D . Pokud uživatel klepne na Přidat nový produkt, Web prohlížeč se připojí na URL http://123.4.567.89/Produkty/Přidat. Za předpokladu, že IP adresa serveru je 123.4.567.89, metoda databáze Při Web spojení obdrží do $1 zvláštní URL data "/Produkty/Přidat" a proto přidá záznamy do [Produkty].

Uživatel může vzít a přetáhnout odkazy ze stránky na plochu a vytvořit tak zkratky pro internet, jako Přidat nového zákazníka, ukázané zde. Jednoduchým poklepáním na tuto ikonu vás tato přesune do patřičné části 4D Web databáze.

Zdrojový kód HTML stránky je ukázán zde:

```
<html>

<head>
<meta http-equiv="Content-Type"
content="text/html; charset=windows-1250">
<meta name="GENERATOR" content="Microsoft FrontPage Express 2.0">
<title>Jarda</title>
</head>

<body>

<h1><b>Systém informací ACME Intranet</b></h1>

<table border="1" width="100%">
    <tr>
        <td><p align="center"><a
        href="http://123.4.567.89/Zákazníci/Přidat">Přidat
        nového zákazníka</a> </p>
        </td>
        <td><p align="center"><a
        href="http://123.4.567.89/Produkty/Přidat">Přidat nový
        produkt</a> </p>
        </td>
    </tr>
    <tr>
        <td><p align="center"><a
        href="http://123.4.567.89/Zákazníci/Seznam">Zobrazit
        existující zákazníky</a> </p>
        </td>
        <td><p align="center"><a
        href="http://123.4.567.89/Produkty/Seznam">Zobrazit
        existující produkty</a> </p>
        </td>
    </tr>
</table>

<p><a href="Help.htm"><b>Pokud potřebujete nápovědu, klepněte</b></a></p>
</body>
</html>
```

Dále si přečtěte

Metoda databáze Při ověření WEB, Metody databáze, Textové parametry předávané do 4D Metod volaných přes URL, Web služby, Nekontextní mód.

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Web služby, Web server nastavení

Příkazy a odkazy pro Web server

Verze 6.5

4D 6.5 vám umožňuje nastavit vaší Web část podle vašich potřeb. Jsou možné následující vlastnosti:
• Definování výchozí domovské stránky
• Použití JavaScript pro kontrolu vstupu dat
• Definování módu předání pro proměnné 4D ze statických stránek
• Upravení módu kontextu
• Nastavení konvence HTML znaků a definování Web filtrů
• Použití cache pro statické stránky
• Definování IP adresy pro kterou server bude vracet HTTP dotazy

Definování výchozí domovské stránky

Můžete definovat výchozí domovskou stránku pro všechny prohlížeče připojené k databázi. Tato stránka může být statická nebo semidinamická.

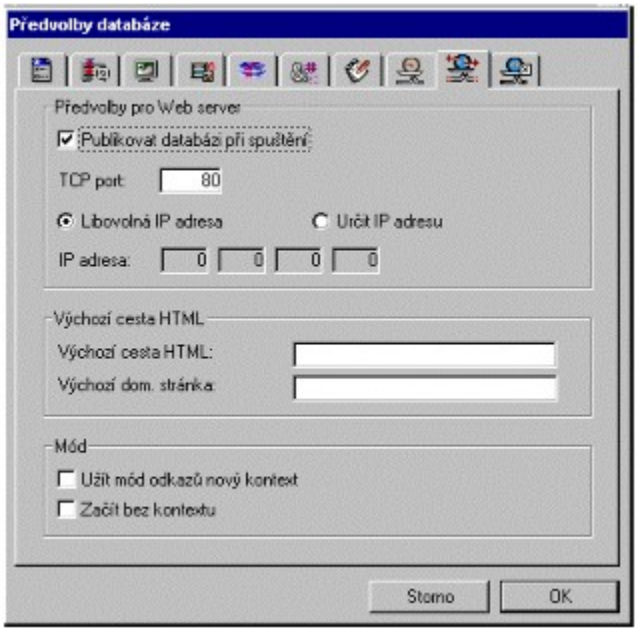
Pokud definujete výchozí domovskou stránku, tato stránka bude poslána na každý prohlížeč který se připojí k databázi bez závislosti na módu (kontextní nebo nekontextní), který byl definován pro část Web. Na rozdíl od předchozí verze 4D není posláno záhlaví nabídek v kontextním módu.

Jako výchozí není definována žádná domovská stránka. Pokud nedefinujete domovskou stránku, bude chování Webu rozdílné podle počátečního módu:

• Pokud Web server začne v kontextním módu (jako výchozí), je posláno platné záhlaví nabídek (jako výchozí Záhlaví #1) - jako v předchozí verzi 4D.
• Pokud Web server začne v nekontextním módu, je provedena pouze metoda databáze Při Web spojení. Je pouze na vás co provedete dále.

K definování výchozí domovské stránky:

1. V Předvolbách databáze přejděte na stránku "Web Server I". Objeví se následující okno:



2. V oblasti "Výchozí dom. stránka" zadejte relativní cestu ke složce kterou chcete definovat.
Cesta je relativní protože začíná ze složky obsahující vaší databázi.

Pro zachování kompatibility vaší databáze mezi platformami, Web server 4D používá následující koncepce zápisu pro zapsání cesty. Pravidla zápisu jsou následující:
• složky jsou odděleny lomítkem ("/")
• cesta přístupu končí lomítkem ("/")
• k přechodu v hierarchii složek o jednu úroveň výše, zadejte ".." (dvě tečky) před název složky
• cesta přístupu nemusí začínat lomítkem ("/") (pokud nechcete složku databáze jako složku HTML, čtěte dále)

Pokud například chcete mít HTML cestu k podsložce "Web" umístěné ve složce "4Ddatabáze", předáte 4DDatabáze/Web/

3. Potvrďte dialogové okno.

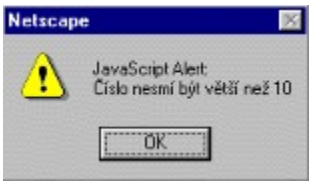
Poznámka: Výchozí domovskou stránku můžete také definovat pomocí příkazu SET HOME PAGE.

Použití JavaScript pro řízení vstupu dat

Ve 4D 6.5 může být část vstupu dat řízena v prohlížeči s použitím automatického JavaScriptu.

Na prohlížeči je řízení vstupu dat a typy dat (polí nebo proměných), které mohou být použity následující:
• minimální hodnota (číselné hodnoty)
• maximální hodnota (číselné hodnoty)
• nutný vstup (číslené a alfanumerické hodnoty)

4D automaticky generuje JavaSript, který je malý ve velikosti, a který zobrazí výstražné okno a zabrání uživateli přijmout vstup dat. Pokud je v poli nesprávná hodnota, zobrazí se výstražné okno při klepnutí na tlačítko (OK, Storno, atd.):



Jakmile je výstražné okno potvrzeno a závada opravena a jestliže uživatel klepne podruhé na tlačítko, akce tlačítka provedena.
Kompletní kontrola vstupu dat je provedena na Web serveru, v módu Uživatele a Vlastních nabídek.

K použití JavyScript ke kontrole vstupu dat:
1. V předvolbách databáze přejděte na stránku "Web Server II".
2. Zaškrtněte možnost "Užít JavaSript pro řízení vstupů".
Jako výchozí tato možnost není označena.
3. Klepněte na tlačítko **OK**.

Nový způsob zadání proměnných 4D ve statických stránkách

Možnost **Užít 4DVAR comment místo závorek**, která je v Předvolbách databáze na straně "Web server II", vám umožní definovat zápis pro použití při zadávání proměnných 4D na statických stránkách.

• Pokud je tato možnost zaškrtnuta, zápis který potřebujete je standardní HTML zápis <!--4DVAR MOPROM→ (znak mezery musí být mezi 4DVAR a názvem proměnné).
• Pokud tato možnost není zaškrtnuta (výchozí hodnota), zápis který potřebujete použít je zápis s hranatými závorkami ([MOPROM]) - což je speciální řešení.

Nový mód k odkazování kontextů

Jako výchozí Web server 4D posílá čísla platného kontextu na prohlížeč po každé akci uživatele. Pokud například stránka obsahuje dva odstavce a obrázek, 4D pošle číslo kontextu třikrát.

Tuto možnost je možné změnit a tím zvýšit rychlost kterou jsou stránky posílány. S novým módem odkazování kontextu je číslo kontextu umístěno do základní URL dokumentu. V tomto případě není opakováno pro každý objekt.

K použití nového módu odkazování kontextu:
1. V Předvolbách databáze na straně "Web Server I" označte možnost "Užít mód odkazů nový kontext"
Jako výchozí není tato možnost označena.
2. Klepněte na tlačítko **OK** a restartujte databázi.

Přímé posílání rozšířených ASCIIznaků

Jako výchozí, Web server 4D převede rozšířené ASCII znaky (nad 128) před jejich posláním v dynamické a statické stránce Web stránky podle HTML standardu. Pak jsou interpretovány prohlížečem.

Web server můžete nastavit tak, že bude znaky posílat "tak jak jsou" bez jejich konvertování do HTML entity. Tato vlastnost velmi zvýší rychlost na určitých operačních systémech (specilně Japonský sestém).

K umožnění Web serveru posílat ASCII znaky přímo:
1. V okně Předvolby databáze přejděte na stránku "Web server II".
2. Zaškrtněte možnost "Odesílat rozšířené znaky přímo".
3. Potvrďte okno.

Upravení nastavení konvence znaků ve 4D

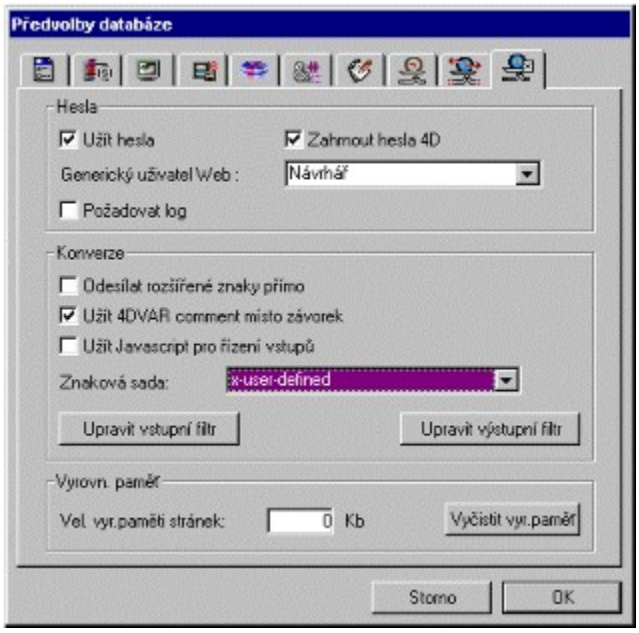
Tabulku ASCII znaků (Web filter), použitou pro vstup a výstup dat z 4D můžete přímo změnit. Tato vlastnost byla dříve v Customizer Plus.

Úprava tabulky znaků je možná pokud je nastavena sada znaků "x-user-defined" v Předvolbách databáze.

K upravení tabulky ASCII znaků:

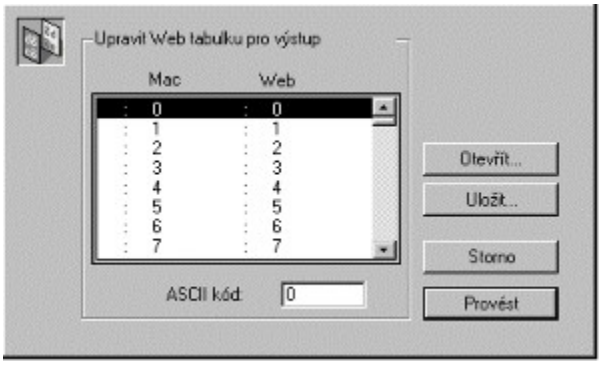
1. V okně Předvolby databáze přejděte na stránku "Web server II".
2. V rozevírací nabídce "Znaková sada" vyberte sadu "x-user-defined".

Nyní jsou dostupná tlačítka **Upravit vstupní filtr** a **Upravit výstupní filtr**.



3. Klepněte na tlačítko filtru který chcete upravit. Vstupní filtr ovlivňuje znaky poslané z prohlížeče na Web server 4D a výstupní filtr upravuje znaky poslané z Web serveru 4D na prohlížeč.

Objeví se následující okno:



Poznámka: Toto okno vypadá stejně jako okno filtru pro Import/export 4D, ale obě jsou nezávislé; Web filter nemá ŽÁDNÝ vztah s filtry pro import/export.

4. V posuvné oblasti najděte a klepněte na Mac znak, který chcete filtrovat.

NEBO

Načtěte již vytvořený Web filtr klepnutím na tlačítko "Otevřít".

5. V dostupné oblasti "ASCII kód" zadejte nový ASCII kód znaku.

Poznámka: K zajištění kompatibility databáze, jsou výchozí hodnoty filtru stejné jako ty, které používá 4D 6.0.6 (z

"MapC" - jestliže definováno v Customizer Plus).

6. Opakujte tuto akci pro všechny znaky které chcete filtrovat.
7. Klepněte na tlačítko "Uložit" k uložení filtru.
8. Klepněte na tlačítko **Užít mapu**.
9. Klepněte na tlačítko **OK** v Předvolbách databáze.

Upravený vstupní a/nebo výstupní Web filter je nyní aktivní.

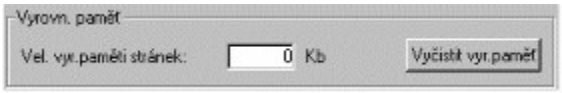
Paměť pro statické stránky

Web server 4D obsahuje paměť která vám umožní načíst statické stránky, GIF obrázky, JPEG obrázky (<100Kb) a styly (.css soubory) do paměti když jsou vyžadovány. S použitím této paměti můžete výrazně zvýšit výkon Web serveru při posílání statických stránek.

Tato paměť je sdílena všemi Web procesy. Velikost této paměti můžete nastavit v Předvolbách databáze. Jako výchozí není paměť pro statické stránky dostupná (její velikost je 0).

K zpřístupnění paměti pro statické stránky:

1. V Předvolbách databáze na straně "Web Server II" nastavte hodnotu (vyjádřenou v Kb) paměti pro statické stránky.



Hodnota kterou předáte je závislá na počtu a velikosti vašich statických Web stránek stejně jako na zdrojích se kterými má stroj hostitele právo disponovat.

Poznámka: Při používání vaší Web databáze, můžete testovat výkon paměti s použitím příkazu WEB CACHE STATISTICS. Pokud dostanete například výsledek, že využití paměti je 100%, měli by jste zvětšit velikost přiřazené paměti.
URL /4DSTATS a /4DHTMLSTATS vám umožní získat informace o stavu paměti (cache). Přečtěte si část Web služby, Informace o Web straně.

2. Klepněte na **OK**.
Jakmile je paměť zpřístupněna, Web server 4D se podívá po vyžadované stránce prohlížeče první v paměti. Pokud nalezne stránku, ihned jí pošle. Pokud ne, 4D načte stránku z disku a umístí ji do paměti.
Jakmile je paměť zaplněna a je vyžadováno další "místo", 4D odstraní nejstarší požadované stránky.

Vymazání paměti

V některých chvílích můžete potřebovat paměť vyčistit od stránek a obrázků které obsahuje (např. pokud máte upravenou statickou stránku a chcete jí publikovat na Web). K tomu musíte klepnout na tlačítko **Vyčistit vyr. paměť** v Předvolbách databáze na straně "Web server II. Paměť je ihned vyčištěna.

Definování IP adresy pro HTTP dotazy

V Předvolbách databáze na straně "Web Server I" je možné definovat HTTP adresu ze které Web server musí přijímat HTTP dotazy.

Jako výchozí přijímá server všechny adresy (možnost **Libovolná IP adresa**). Pokud vyberete možnost **Určit IP adresu**, oblast IP adresa je zpřístupněna a vy můžete definovat adresu jako třeba "194.166.100.101". V tomto případě bude server odpovídat pouze na dotazy na tuto adresu.

Tato možnost je pro Web server 4D umístěný na počítači s více TCP/IP adresami. Je to častý případ poskytovatelů

Internet služeb.

Poznámka: Předání systému více domovských stránek vyžaduje specifické nastavení stroje Web serveru. Více informací najdete v části Podpora druhé IP adresy v manuálu "Network Components for 4D Server".

Dále si přečtěte

SET HOME PAGE, Web služby, Bezpečnost připojení, Web služby, Podpora HTML, Web služby, Nekontextní mód, Web služby, Procesy Web spojení.

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Web služby, Nekontextní mód

Příkazy a odkazy pro Web server

Verze 6.5

Web server 4D může být použit v nekontextním módu (od verze 6.5). V tomto módu se 4D stává "normálním" HTTP serverem.

Kontextní a nekontextní mód

V předchozí verzi 4D byly možnosti Webu 4D založeny na **kontextu**: Web prohlížeč který se připojí k databázi vytvoří kontext do kterého se umístí jeho vlastní výběr záznamů, proměnné, atd. Tímto způsobem je každý prohlížeč klientem 4D.

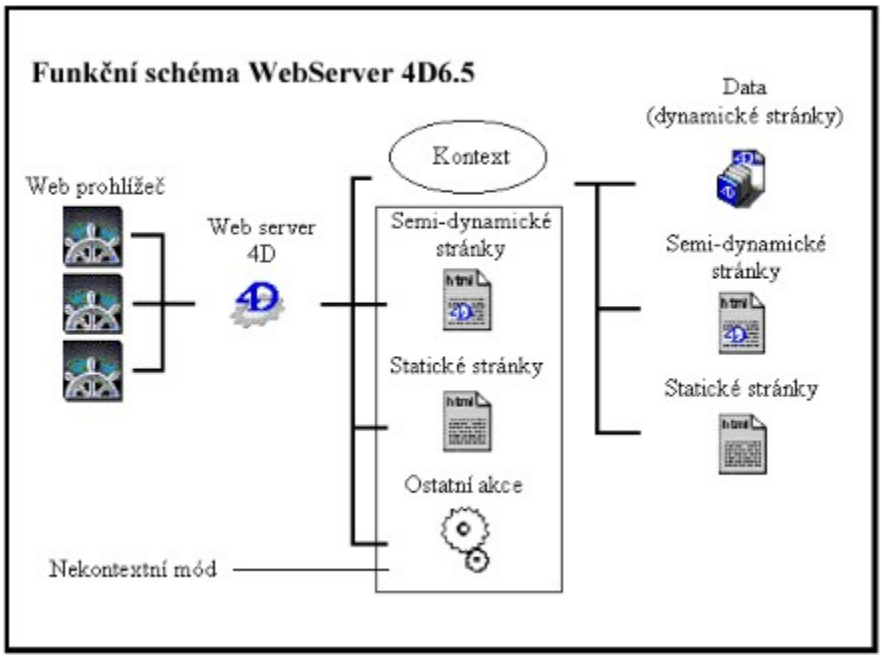
Tento způsob zajišťuje plnou kontrolu akcí prohlížeče a garantuje integritu dat. Vypíná se aby byla 4D použita jako "klasický" HTTP server, aniž by to navíc zatěžovalo Web Server, obzvlášť když 4D posílá statické HTML stránky. V tomto případě zůstával v předchozí verzi kontext stále aktivní dokud neuplul definovaný čas. Jestliže, například, prohlížeč opustí stránku v tomto čase, kontext je "zbytečný". Navíc standardní funkce prohlížeče (**Reload, Předchozí stránka**, atd.) nejsou v kontextu použitelné (přečtěte si kapitolu Web služby, Procesy Web spojení)

K použití Web serveru 4D jako klasického HTTP serveru, který posílá stránky, 4D obsahuje **nekontextní mód**. V tomto módu se Web server 4D stane klasickým HTTP serverem; statické stránky jsou posílány bez požadavků na kontext. Tyto stránky mohou být posílány z vyrovnávací paměti. V tomto případě můžete zjistit statistiku o využití této paměti pomocí příkazu WEB CACHE STATISTICS.

Kontext je samozřejmě vždy používán ke kontrole navigace prohlížeče po záznamech databáze.

Web server 4D můžete použít v jakémkoli módu chcete; kontextním, nekontextním nebo mezi nimi přepínat "za běhu" podle vašich potřeb.

Jak funguje Web server 4D 6.5 je znázorněno v následujícím obrázku:



Přepnutí z jednoho módu do druhého

Jako výchozí pracuje Web server 4D 6.5 jako v předchozí verzi a stejně tak je ovládá řízení kontextu; kontext je vytvořen pro každé Web spojení. Musíte výslovně zadat, že chcete přepnout do nekontextního módu.

Přepnutí z kontextního módu do nekontextního módu
Kdykoli během používání databáze můžete přepnout z kontextního na nekontextní mód a naopak.
Poznámka: Jako výchozí se zapne Web server 4D v kontextním módu. Nicméně můžete definovat, aby začal v nekontextním módu. Více informací najdete v další části "Definování nekontextnmího módu při spuštění".
Pro přepnutí Web serveru 4D do nekontextního módu jsou tři způsoby:
• Provedení nového příkazu SEND HTML BLOB s předáním True do volitelného parametru (bezKontextu). Například: SEND HTML BLOB (mujBLOB;".htm";True). 4D pošle BLOB v nekontextním módu. Proces který ovládá kontext je zastaven.
Poznámka: Nepředání volitelného parametru, je stejné jako předání False. Pokud je nekontextní mód již aktivní, je parametr ignorován. Platný mód můžete zjistit pomocí funkce Web Context.
• Pomocí nového příkazu SEND HTTP REDIRECT.
Tento příkaz vám umožní přesměrovat dotaz na URL předáné jako parametr. Volané z kontextního módu, zavře Web proces.
• Zavolat URL začínající /4DACTION (následující odstavec 4DACTION tag).
Nekontextní mód může být použit pokud není volána žádná URL vyžadující kontext.

Přepnutí z nekontextního do kontextního módu
Klasický kontextní mód 4D je použit při žádosti na databázi, což znamená že Web server musí vytvořit a poslat dynamickou stránku. Volání databáze z nekontextního módu je provedeno přes patřičnou URL /4DMETHOD/MojeMetoda. Stačí tuto URL umístit do statické stránky (např. jako tlačítko). Více informací najdete v části Web služby, Zahrnutí HTML a JavaScriptu.

Když je tato URL dostupná, Web server 4D vytvoří nový kontext a provede následující akce:
• Je spuštěna Metoda databáze Při ověření WEB (pokud existuje)
• Je spuštěna Metoda databáze Při Web spojení (pokud existuje)
metoda je provedena v novém kontextu.

Web server funguje tak jako v předchozí verzi 4D. Kontextní mód může pracovat dokud jej nějakým příkazem nevypnete.

Počet používaných kontextů

Podle prováděné akce, některé Web procesy použijí Web kontext a některé ne.

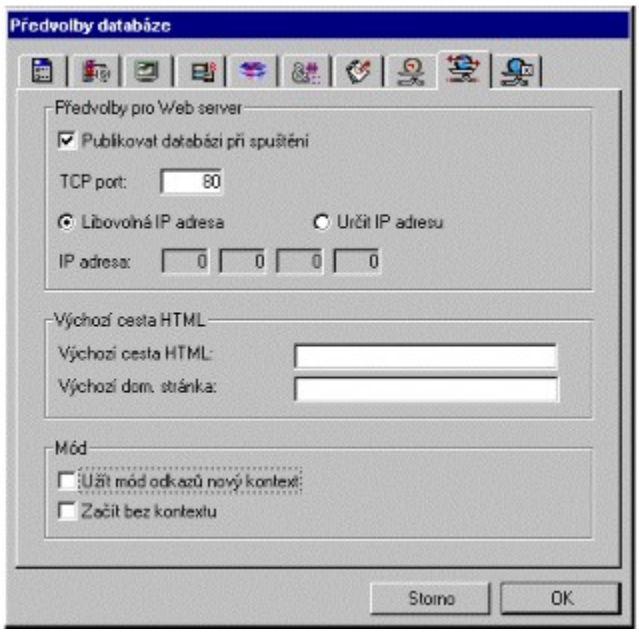
Počet vytvořených kontextů můžete zjistit pomocí příkazu PROCESS PROPERTIES pro některý Web proces. Tento příkaz určí do parametru původ, jestliže kontext byl použit (-11, Web proces s kontextem) nebo ne (-3, Web proces bez kontextu).

Definování nekontextního módu při spuštění

Web server můžete zapnout tak, aby byl automaticky v nekontextním módu. To znamená, že když se uživatel připojí k databázi, není vytvořen kontext.

K definování nekontextníhop módu při spuštění:
1. V předvolbách databáze přejděte na stránku "Web server II".
2. Označte možnost "Začít bez kontextu".

Jako výchozí není tato možnost označena a Web server začne v nekontextním módu.



3. Potvrďte toto okno.

Web server 4D pošle tuto stránku v nekontextním módu.

• Pokud máte definovánu vlastní domovskou stránku v Předvolbách databáze v oblasti "Výchozí dom. stránka", je tato stránka poslána.
Poznámka: Jestliže URL končí na /, název domovské stránky je předán a 4D se jí pokusí poslat. Jestliže URL je "/Složka/" a výchozí domovská stránka je MojStr.HTM, 4D bude hledat "/Složka/MojStr.HTM". Pokud není nalezena, bude spuštěna metoda Při Web spojení.

• Jestliže nemáte definovánu výchozí stránku, nebo tato stránka nebyla nalezena, je spuštěna metoda databáze Při Web spojení (v nekontextním módu). Spojení můžete řídit i vlastním způsobem. Například můžete použít příkaz SET HOME PAGE pro každého uživatele, který se připojí k databázi.

Semi-dynamické stránky

4D vám umožňuje používat Web stránky které byly vytvořeny semi-dynamickým způsobem. Semi-dynamické stránky jsou stránky, jejichž obsah je celkově nebo částečně vytvářen akcemi 4D.

Tyto stránky mohou být použity jak v kontextním tak v nekontextním módu. Nicméně jsou více používané v kontextním módu. V tomto módu 4D neovládá proměnné, výběry, atd. vztažené k navigaci prohlížeče. Jsou ovládány volným způsobem bez informování 4D. Účel semi-dynamických stránek je dát vám možnost využít možnosti 4D na vašich Web stránkách při používání nekontextního módu.

Semi-dynamické stránky můžete vytvořit těmito způsoby:
• Posláním URL /4DCGI/<akce> na Web server 4D
• Posláním HTML stránky s příponou ".shtm" nebo ".shtml".
• S použitím /4DACTION/ tagu

URL /4DCGI/<akce>

Jakmile Web server 4D obdrží URL /4DCGI/<akce>, zavolá metodu databáze Při Web spojení a vloží URL "tak

jak je" do parametru $1.

URL 4DCGI/ neodpovídá žádnému souboru. Jeho úloha je zavolat samotnou 4D. Parametr <akce> může obsahovat nějaký typ informace.

Tento URL vám umožní provést akci jakéhokoli typu. Musíte pouze testovat hodnotu $1 v metodě databáze Při Web spojení nebo v nějaké její podmetodě a provést požadovanou akci. Můžete například vytvořit vlastní HTML stránky pro přidávání, hledání a třídění záznamů a nebo vytvářet GIF obrázky za běhu. Příklady na využití tohoto URL jsou v příkladech příkazů PICT TO GIF a SEND HTTP REDIRECT.

Při provádění akce, musí být poslána odpověď s použitím jednoho z příkazů posílajících data (SEND HTML FILE, SEND HTML BLOB, atd.).

Upozornění: Provádějte nejkratší možné akce aby jste nezdržovali prohlížeč.

Soubor ".shtm"

Jakmile Web server pošle soubor jehož přípona je ".shtm" (nebo ".ahtml"), analyzuje odkazy na proměnné 4D a komentáře 4DACTION které mohou být ve stránce, uložené na disku před jejím posláním, obsaženy.

Tato možnost vám umožní používat 4D pro zpracování HTML stránky, kde je nutno zajistit obnovu dynamických odkazů ve statické stránce. Nicméně se ujistěte, že provádíte nejkratší možnou akci aby jste nezdržovali prohlížeč.

Poznámka: V nekontextním módu není možné vytvářet odkazy na obrázkové proměnné 4D.

Prosím, přečtěte si příklad u příkazu WEB CACHE STATISTICS.

4DACTION tag

4DACTION tag vám umožňuje provádět metody 4D při posílání statických HTML stránek. Můžete jej použít těmito způsoby:
• Jako HTML komentář (<!--4DACTION ...→),
• Jako URL (<A HREF="/4DACTION/...">),
• K poslání formuláře.

4DACTION jako HTML komentář

Přítomnost tagu 4DACTION/MojeMetoda/MůjParam ve statické stránce jako HTML komentář urychlí provedení metody MojeMetoda s parametrem MůParam jako řetězec v $1. Při načítání domovské stránky, 4D zavolá metodu databáze Při ověření WEB. Pokud vrátí True, 4D provede metodu.

Metoda vrací do $0 text. Jestliže řetězec začínání na ASCII znakem číslo 1, je následující řetězec považováno za HTML (stejný princip je použit na proměnné. Přečtěte si část Web služby, Zahrnutí HTML a JavaScriptu)

Upozornění: Aby tato možnost fungovala, musí být v Předvolbách databáze označena možnost "Užít 4DVAR comment místo závorek" (přečtěte si část Web služby, Web server nastavení).

Analyzování obsahu stránky je provedeno při příkazu SEND HTML FILE (.htm, .html, shtm, shtml) nebo SEND HTML BLOB (blob typu text/html). Nezapomeňte, že v nekontextním módu je analýza provedena, pokud URL ukáže na soubor ".shtm" nebo ".shtml" (například htp://www.server.com/dir/stránka.shtm).

Poznámka: V kontextním módu je metoda provedena v kontextu.

Řekněme že například předáte komentář "Dnes je <!--4DACTION/MOJEMET/MUJPAR→" do statické stránky. 4D spustí metodu Při ověření WEB (pokud existuje), provede metodu MOJEMET s MUJPAR předáným jako parametr $1.

Metoda vrátí do $0 text (například "31.12.99"). Výraz proto bude "Dnes je 31.12.99". Metoda MOJEMET je zde:

```
C_TEXT($0) ` Tento parametr musí být vždy deklarován
```

```
C_TEXT($1) ` Tento parametr musí být vždy deklarován
$0:=String(Current date)
```

Poznámka: Tato metoda nesmí volat prvky rozhraní (ALERT, DIALOG, atd.).

Jak 4D provádí metody v pořadí jak se objevují, je možné provést metodu, která nastaví hodnoty na které je odkazováno dále v dokumentu a to bez rozdílu používaného módu.

Poznámka: Do statické stránky můžete vložit libovolný počet komentářů 4DACTION.

4DACTION jako URL

Jakmile 4D obdrží dotaz 4DACTION/MOJEMET/MUJPAR, je provedena Metoda databáze Při ověření WEB (pokud existuje). Jestliže vrátí True, provede 4D metodu MOJEMET s MUJPAR předáným jako parametr $1. Zápis URL musí být v následující formě: <A HREF="/4DACTION/MOJEMET/MUJPAR">Udělat něco</A>

Poznámka: Tato možnost je stejná jako /4DMETHOD až na to, že je použita v nekontextním módu.

V kontextním módu, ve chvíli, kdy 4D obdrží dotaz /4DACTION, je odstraněn kontext a metoda je provedena mimo kontext. Je to přesný opak provedení 4DMETHOD v nekontextním módu.

Poznámka: Metoda prováděná /4DACTION nesmí volat prvky rozhraní (ALERT, DIALOG, atd.).

Příklad

Předáte následující instrukci do stránky HTML:

```
<IMG SRC=/SRC="/4DACTION/PICTFROMLIB/1000">
```

Metoda PICTFROMLIB následuje:

```
C_TEXT($1) `Tento parametr musí být vždy deklarován
C_PICTURE($PictVar)
C_BLOB($BlobVar)
C_LONGINT($Number)
   `Obdržíme číslo obrázku do $1
$Number:=Num(Substring($1;2;99))
GET PICTURE FROM LIBRARY($Number;$PictVar)
PICT TO GIF($PictVar;$BlobVar)
SEND HTML BLOB ($PictVar;"Pict/gif")
```

4DACTION k potvrzení (post) formuláře

Nekontextní mód umožňuje dodatečnou možnost pokud chcete použít "potvrzený" formulář, který je statickou HTML stránkou, která posílá data na Web server. Formulář musí být typu POST a akce formuláře musí povinně začínat /4DACTION/NázevMetody.

V tomto případě, pokud Web server obdží potvrzený formulář, provede metodu projektu **COMPILER_WEB** (jestli existuje, čtěte dále), pak metodu databáze Při ověření WEB (pokud existuje). Jestliže tato vrátí True, metoda NázevMetody je provedena. 4D analyzuje HTML pole obsažené ve stránce, vezme jejich hodnoty a automaticky je přiřadí do proměnných 4D. Pole ve formuláři a proměnné 4D musí mít stejné názvy.

Poznámka: Metoda prováděná 4DACTION nesmí volat prvky rozhraní (ALERT, DIALOG, atd.).

Poznámka: 4D přijme potvrzené formuláře typu GET. V tomto případě však musí návrhář ručně zpracovat HTML obsah generované URL. Navíc metoda **COMPILER_WEB** není volána (čtěte dále).

Zápis v HTML k použití potvrzeného formuláře je následujícího typu:

• definování akce ve formuláři HTML:
<FORM ACTION="/4DACTION/NázevMetody" METHOD=POST>
• definování pole ve formuláři HTML:
<INPUT TYPE=NÁZEV pole=HODNOTA pole="Výchozí hodnota">

Hodnotu každého pole ve formuláři HTML přiřadí 4D při přijetí do proměnné se stejným názvem.

Pro vlastnosti ve formuláři (například zaškrtávací políčka), 4D nastaví přislušnou proměnnou na 1, pokud je označeno a na 0 pokud není.

Pro číselná data, 4D převede hodnotu pole z Alfa→Real.

Poznámka: Pokud pole ve formuláři je nazváno OK (například tlačítko potvrzení), je systémová proměnná OK nastavena na 1, pokud hodnota pole není prázdná, jinak je 0.

Metoda COMPILER_WEB: Jakmile Web server 4D obdrží potvrzený formulář, provede metodu **COMPILER_WEB** (pokud existuje). Tato metoda musí obsahovat všechna deklareci a/nebo inicializaci proměnných, jejichž názvy jsou stejné jako názvy polí ve formuláři. Bude také použita 4D Compilerem při kompilování databáze.

Metoda **COMPILER_WEB** je společná pro všechny formuláře.

Příklad

V 4D Web databázi zapnuté a používané v nekontextním módu, předpokládáme, že prohlížeč bude vyhledávat záznamy pomocí statických HTML stránek. Tato stránka je nazvána "search.htm". Databáze obsahuje ještě další statické stránky, které vám například umožní zobrazit výsledek hledání. Daná stránka je typu POST a provádí akci /4DACTION/SEARCH.

Zde je zobrazená HTML stránka:

Zde je část kódu formuláře, který odpovídá této stránce:

```
<form action="/4DACTION/PROCESSFORM" method="POST"
name="search.htm">
    <p><font size="5"><strong>Vítejte na našem Web Serveru</strong></font></p>
    <p> </p>
    <p>Zadejte hodnotu, kterou chcete hledat:<br>
    <input type="text" size="20" name="vNAME"><br></p>
    <p><input type="checkbox" name="vEXACT" value="1">Celé slovo<br></p>
    <p><input type="reset" name="Clear" value="Odstranit"> <input
    type="submit" name="OK" value="Hledat"> </p>
```

<!-- Obvykle vložíte název tlačítka do VALUE, z důvodů výkladu, musíte do VALUE vložit i hodnotu→
<!-- OK je zvláštní případ→

Během vstupu dat, zadejte "ABCD" do oblasti vstupu dat zkontrolujte a potvrďte na tlačítku **Hledat**.

4D zavolá metodu COMPILER_WEB, která je následující:

```
C_TEXT(vNAME)
vNAME:=""
C_LONGINT(vEXACT)
vEXACT:=0
OK:=0 `patřičný případ
```

V tomto příkladu obsahuje vNAME řetězec "ABCD", vEXACT je rovno 1 a proměnná OK je nastavena na 1 (protože název tlačítka je OK).

4D provede metodu databáze Při ověření WEB (pokud existuje) a pak je provedena metoda PROCESSFORM, která je následující:

```
If (OK=1)
   If (vEXACT=0) `Pokud tato možnost nebyla označena
      vNAME:=VNAME+"@"
   End if
   QUERY([ŘízeníKlíče];[ ŘízeníKlíče]Název=vNAME)
   vLIST:=Char(1) `Vrátit seznam do HTML
   FIRST RECORD([ ŘízeníKlíče])
   While (Not(End selection([ ŘízeníKlíče])))
      vLIST:=vLIST+[ ŘízeníKlíče]Název+" "+[ ŘízeníKlíče]Tel+"<BR>"
      NEXT RECORD([ ŘízeníKlíče])
   End while
   SEND HTML FILE("results.htm") `Poslat seznam do formuláře vysledek.htm
   `která obsahuje odkaz na proměnnou vLIST (tojest <!--4DVAR vLIST→)
   ...
End if
```

Dále si přečtěte

SEND HTML BLOB, SEND HTTP REDIRECT, Textové parametry předávané do 4D Metod volaných přes URL, Web Context, Web služby, Zahrnutí HTML a JavaScriptu.

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Web služby, Podpora HTML

Příkazy a odkazy pro Web server

Verze 6.0

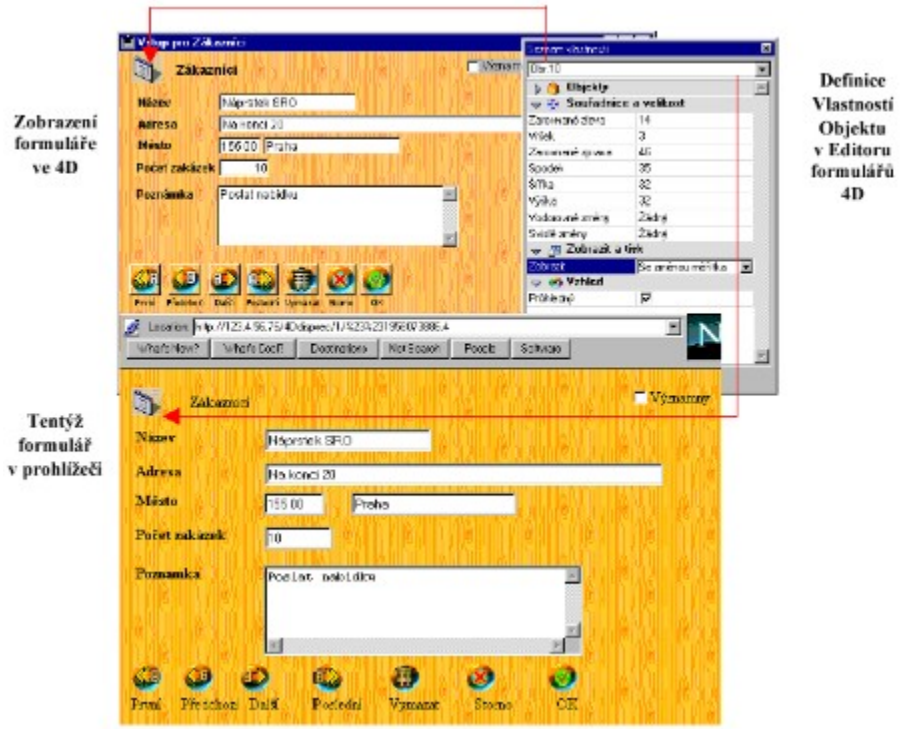
Před kombinováním HTML a JavaScriptu ve vašich aplikacích, je dobré vyzkoušet co pro vás 4th Dimension udělá.

Záhlaví nabídek

• Každé záhlaví nabídek je přeloženo do jedné HTML stránky. Každá nabídka se objeví jako text a položky jako odkazy na metody 4D.
• Obrázek přiřazený k záhlaví nabídek je umístěn pod nabídku v prohlížeči.
• Klepnutím na položku nabídky stránka Web prohlížeče provede přiřazenou metodu v procesu Web spojení.

Formuláře

• Objekty jsou přeloženy zhora dolů a zleva do prava a jsou umítěny stejně jako ve formuláři ve 4D. Nicméně si všiměte, že HTML je textově orientovaná aplikace: vodorovná pozice objektů bude rozdílná a můhou se objevit obtečené textem.
• Vícestránkové formuláře jsou podporovány každá stránka se zahrnutím stránky 0.
• Automatické akce, pokud jsou, jsou prováděny.
• Události formuláře (Při zavedení, Při vyvedení, Při klepnutí a Při časování) jsou podporovány. Ostatní nejsou.
• Záhlaví, Obsah, Zlom a Zápatí jsou vzaty do úvahy během provedení převodu do HTML při DISPLAY SELECTION a MODIFY SELECTION. Záhlaví formuláře se objeví jednou na začátku HTML stránky, oblast obsahu je opakována kolikrát je potřeba a proměnné (jako tlačítka) umístěná v Zápatí se zobrazí na konci stránky hned pod automatickými odkazy na navigaci stránek.
• Tipy přiřazené k tlačítkům zobrazeným jako obrázky se objeví i na prohlížeči - pokud jejich zobrazení umožňuje.
• Replikovaný obrázek umístěný na souřadnicích (0,0,x,x) v Editoru formulářů je poslán na prohlížeč jako obrázek pozadí, pokud není na souřadnicích (0,0,x,x), je poslán jako normální obrázek:



Zapamatujte si, že by jste se měli vyhnout tmavým obrázkům.

Objekty polí

Jakmile je formulář 4D přeložen do HTML stránky, objekty polí jsou přeloženy následovně:

Typ pole 4D	*HTML objekt*	*HTML značka*
Alfa	*Textová pole (*)*	*<INPUT Type="text" ...>*
Text	*Textová pole (*)*	*<TEXTAREA ...> (**)*
		<INPUT Type="text" ...> (***)
Real	*Textová pole (*)*	*<INPUT Type="text" ...>*
Integer	*Textová pole (*)*	*<INPUT Type="text" ...>*
Long Integer	*Textoví pole (*)*	*<INPUT Type="text" ...>*
Datum	*Textová pole (*)*	*<INPUT Type="text" ...>*
Čas	*Textová pole (*)*	*<INPUT Type="text" ...>*
Logické	*Přepínač a Zašktávací (*)*	*<INPUT Type="radio" ...>*
		<INPUT Type="checkbox" ...>
Obrázkové	*Image (vždy nedostupné)*	*<IMG SRC="..." ...>*
Pdtabulka	*Nepodporuje HTML*	*Žádný*
BLOB	*Nepodporuje HTML (****)*	*Žádný*

(*) nebo pouze text pokud je pole nedostupné
(**) jestliže je text složen z více řádků
(***) Pokud je text složen z jednoho řádku nebo je prázdný
(****) Příkaz SEND HTML BLOB vám umožňuje poslat libovolé BLOB pole nebo proměnnou na prohlížeč.

Poznámka: Dostupné proměnné se chovají jako pole.

Objekty formuláře

Jakmile je formulář 4D přeložen do HTML stránky, objekty formuláře jsou přeloženy následovně:

Objekt formuláře 4D	*Stejný HTML objekt*	*HTML značka*
Čára	*Vodorovná čára (1)*	*<HR>*
Obdélník	*Nepodporuje HTML*	*Žádný*
Ovál	*Nepodporuje HTML*	*Žádný*
Zaoblený obdélník	*Nepodporuje HTML*	*Žádný*
Statický obrázek	*Image nebo Image Map (2)*	*<IMG SRC="...">*
		<INPUT Type="image" ...>
Skupina	*Text*	*Text se značkami písma*
Statický Text	*Text*	*Text se značkami písma*
Tlačítko	*Tlačítko potvrzení*	*<INPUT Type="submit" ...>*
Výchozí tlačítko	*Tlačítko potvrzení*	*<INPUT Type="submit" ...>*
Přepínací tlačítko	*Přepínací tlačítko (3)*	*<INPUT Type="radio" ...>*
Zaškrtávací pol.	*Zaškrtávací pol.*	*<INPUT Type="checkbox" ...>*
Rozevírací nabídky/seznam	*Rozevírací seznam*	*<SELECT ...>...</SELECT>*
Text se seznamem	*Rozevírací seznam*	*<SELECT ...>...</SELECT>*
Posuvná oblast	*Posuvný seznam (4)*	*<SELECT ...>...</SELECT>*
Ovládací karta	*URL seznam (5)*	*<A HREF="/4DTAB/4DVar.Onglet...">*
Neviditelné tlačítko	*Přečtěte si poznámku 2*	
Zvýrazněné tlačítko	*Přečtěte si poznámku 2*	
3D tlačítko	*Přečtěte si poznámku 2*	
Tlačítka na síti	*Přečtěte si poznámku 2*	
Diagram	*Image (nedostupné)*	*<IMG SRC="..." ...>*
Plug-in	*HTML text, Image*	*Text se značkami písma*
	nebo Image Map (6)	*nebo <IMG SRC="..." ...>*
		nebo <INPUT Type="image"...>

Následující objekty nejsou podporovány HTML a proto jsou během překladu ignorovány:

Hierarchická rozevírací nabídka, Hierarchický seznam, Podformulář, Obrázekový volič, Teploměr, Pravítko, Výseč, Obrázková rozevírací nabídka, Obrázková tlačítka, 3D zaškrtávací políčko, 3D volič.

Poznámky

1. Nevodorovné čáry nejsou podporovány HTML a proto jsou vynechány.
2. Objekty ve 4D podobné neviditelným tlačítkům tj. objekty: Neviditelné tlačítko, Zvýrazněné tlačítko, 3D tlačítko a Tlačítka na síti. Jestliže zde statický obrázek není překryt neviditelným tlačítkem, je obrázek přeložen jako statický obrázek. Jestliže je obrázek překryt alespoň jedním "neviditelným tlačítkem", je přeložen jako Image map strany serveru (síť obrázků s odkazy). Na straně 4D při zpětném vyhodnocení potvrzení z prohlížeče, 4D přepočítá potvrzení na pozici klepnutí aby vytvořila událost Při klepnutí pro správné tlačítko. Ovládání těchto "neviditelných tlačítek" je proto velmi jednoduché. Tato tlačítka ovládáte přes metodu formuláře nebo jejich vlastní metody objektu, jako ve standardní 4D . Tento způsob Vám také umožňuje snadné vytváření odkazů na sítě obrázků Image map. Pokud není "neviditelné tlačítko" překryto obrázkem, je již během překladu ignorováno.
3. Skupiny přepínačů jsou ovládány přes obdobný překlad do HTML.
4. Svázané skupiny posuvných oblastí nejsou v HTML podporovány. 4D je přeloží jako jednotlivé nezávislé oblasti.
5. Prvky ovládací karty (typu array nebo vytvořené ve vlastnostech objektu) jsou převedeny do odkazů URL. Pokud jsou prvky array prázdné, 4D zobrazí na prohlížeči 1, 2, 3..
6. Oblasti Plug-in (4D Write, Chart…) jsou presentovány na Webu, nejdříve jsou převedeny do HTML, Image nebo Image mapy (sítě obrázků). Poslední možnost vám umožňuje ovládat klepnutí myši uvnitř oblasti. Způsob jakým je oblast Plug-in z formuláře převedena na Web je závislé na jejím nastavení.

Zobrazit výběr / Upravit výběr

• Mechanismus UserSet není podporován (v prohlížěči nelze vybrat více záznamů)

• Automatický mechanismus stránkování výběru je proveden 4D automaticky. Podrobnější informace najdete u příkazu SET WEB DISPLAY LIMITS.

Příkazy 4D

Při vývoji databáze se můžete dotazovat, co ten či onen příkaz dělá. Bude mít příkaz vliv na stroj serveru nebo na stroj prohlížeče? Procesy Web spojení jsou prováděny na Web serveru, ale jejich rozhraní je na prohlížeči. Pro vývoj Web databáze mohou být příkazy klasifikovány následovně:

Příkazy které nejsou ovlivněny spuštěním z procesu Web spojení

Příkazy jako CREATE RECORD pracují ve spuštěném procesu: v tomto případě v procesu Web spojení zde vytvoří záznam. Obdobně fugují příkazy jako Screen width, které vrací šířku obrazovky na stroji Web serveru (stroj kde je proces prováděn).

Přikazy obsahující extra množnosti pro podporu Web

Název příkazu	***Komentář***
ADD RECORD	*Automatický překlad formuláře, podpora vícestránkových*
ALERT	*Automatický překlad dialogového okna*
CONFIRM	*Automatický překlad dialogového okna*
DIALOG	*Automatický překlad formuláře, podpora vícestránkových*
DISPLAY SELECTION	*Automatický překlad formuláře* *Vestavěný mechnismus stránkování Web* *UserSet není podporováno*
MODIFY RECORD	*Automatický překlad formuláře, podpora vícestránkových*
MODIFY SELECTION	*Automatický překlad formuláře* *Vestavěny mechnismus stránkování Web*

	UserSet není podporováno
QUERY	*Statndardní okno dotazu přeložené do HTML*
QUERY BY EXAMPLE	*Automatický překlad formuláře, podpora vícestránkových*
Request	*Automatický překlad dialogového okna*
REDRAW	*Obnoví formulář zobrazený na prohlížeči*

Upozornění: V nekontextním módu nemohou příkazy volat prvky rozhraní (ALERT, DIALOG, ...), zobrazení by se provedlo na Web Serveru, bez možnosti ovlivnění a přerušil by se tak běh metody (Jestli chcete vědět více informací, přečtěte si Web služby, Nekontextní mód).

Příkazy k použití, pokud víte co chcete dělat

Následující příkazy se provádějí místně na stroji Web serveru.

Můžete z prohlížeče třeba i tisknout, ale tisk bude proveden na stroji Web serveru.

Pokud provedete prvky rozhraní, objeví se na stroji Web serveru, tzn. Open document("") či Open document("Tento Dokument"). Těmto voláním by jste se měli vyhnout, protože prohlížeč bude čekat na potvrzení tohoto okna na Web serveru. Na druhou stranu je volání těchto příkazů v pořádku pokud neprovede otevření oken.

Název příkazu	***Komentář***
Append document	*V pořádku, pokud nevyvolá okno*
BEEP	*Pípne na stroji Web serveru.*
Create document	*V pořádku, pokud nevyvolá okno*
DISPLAY RECORD	*Neprovede nic*
EXPORT DIF	*V pořádku pokud nevyvolá okno*
EXPORT SYLK	*V pořádku pokud nevyvolá okno*
EXPORT TEXT	*V pořádku pokud nevyvolá okno*
IMPORT DIF	*V pořádku pokud nevyvolá okno*
IMPORT SYLK	*V pořádku pokud nevyvolá okno*
IMPORT TEXT	*V pořádku pokud nevyvolá okno*
LOAD SET	*V pořádku pokud nevyvolá okno*
LOAD VARIABLES	*V pořádku pokud nevyvolá okno*
MESSAGE	*Zpráva se objeví na stroji Web serveru*
Open document	*V pořádku pokud nevyvolá okno*
Open external window	*Okno se otevře na stroji Web serveru*
Open resource file	*V pořádku pokud nevyvolá okno*
Open window	*Okno se otevře na stroji Web serveru*
PLAY	*Zvuk se zahraje na počítači Web serveru*
PRINT FORM	*V pořádku pokud nevyvolá tiskové okno*
PRINT LABEL	*V pořádku pokud nevyvolá tiskové okno*
PRINT RECORD	*V pořádku pokud nevyvolá tiskové okno*
PRINT SELECTION	*V pořádku pokud nevyvolá tiskové okno*
QUIT 4D	*Podporováno, ale vypíná Web server*
SAVE SET	*V pořádku pokud nevyvolá okno*
SAVE VARIABLES	*V pořádku pokud nevyvolá okno*
SELECT LOG FILE	*V pořádku pokud nevyvolá okno*
SET CHANNEL	*V pořádku pokud nevyvolá okno (dokumenty)*
TRACE	*Okno Ladění se objeví na stroji Web serveru*

Příkazy nepodporované procesem Web spojení

Název příkazu	***Komentář***
ADD DATA SEGMENT	*NEPROVÁDĚJTE z procesu Web spojení.*

	Tento příkaz nebyl navržen pro použití na Webu.
ADD SUBRECORD	*NEPROVÁDĚJTE z procesu Web spojení.*
	Tento příkaz nebyl navržen pro použití na Webu.
CHANGE ACCESS	*NEPROVÁDĚJTE z procesu Web spojení.*
	Tento příkaz nebyl navržen pro použití na Webu.
EDIT ACCESS	*NEPROVÁDĚJTE z procesu Web spojení.*
	Okno hesel se objeví na počítači 4D
	Prohlížeč bude čekat na potvrzení tohoto okna
GRAPH TABLE	*NEPROVÁDĚJTE z procesu Web spojení.*
	Tento příkaz nebyl navržen pro použití na Webu.
MODIFY SUBRECORD	*NEPROVÁDĚJTE z procesu Web spojení.*
	Tento příkaz nebyl navržen pro použití na Webu.
ORDER BY	*Pouze programovací podpora*
	Standardní okno třídění není na Webu
PRINT SETTINGS	*NEPROVÁDĚJTE z procesu Web spojení.*
	Tiskové okno se objeví na stroji 4D
	Prohlížeč bude čekat na potvrzení tohoto okna
REPORT	*NEPROVÁDĚJTE z procesu Web spojení.*
	Okno Rychlé zprávy se objeví na stroji 4D
	Prohlížeč bude čekat na potvrzení tohoto okna

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Web služby, Zahrnutí HTML a JavaScriptu

Příkazy a odkazy pro Web server

Verze 6.0

Důležitá poznámka

Před tím, než začnete pracovat s touto částí si přečtěte následující:

- Web služby, Popis
- Web služby, Nastavení
- Web služby, Poprvé (část I)
- Web služby, Poprvé (část II)
- Web služby, Procesy Web spojení
- Web služby, Podpora HTML

Tři způsoby, jak zahrnout HTML kód do vašich aplikací:

1. S použitím příkazu SEND HTML FILE nebo SEND HTML BLOB, můžete poslat statické stránky uložené na disku.
2. Statické textové objekty ve formuláři 4D jako"{NějaStránka.htm}", vloží HTML dokument "NějaStránka.htm" do formuláře do místa statického objektu.

Poznámka: Tato možnost není dostupná v nekontextním módu. Jestli chcete vědět více informací, přečtěte si část Web služby, Nekontextní mód.

3. Jakákoli textová proměnná 4D ve formuláři může zahrnout HTML kód do formuláře 4D, pokud začíná na ASCII znakem č.1 (tzn., vtHTML:=Char(1)+"...HTMLkód...").

V posledních dvou případech je výsledný formulář na straně prohlížeče kombinací objektů 4D a HTML. Nezapomeňte, že u statických textových objektů vkládáte dokument jako takový. Při použití proměnné, vkládáte kousky kódu.

Poznámka: V některých případech při HTML převodu 4D formulářů, obsahujících odkazy "{NějaStránka.htm}" a vytvořených verzí 6.0.x, můžete ve 4D 6.5 dostat neočekávané výsledky. V tomto případě je vhodné použít kompatibilitu převodu se 6.0.x, nastavenou příkazem SET DATABASE PARAMETER.

S použitím příkazu SEND HTML FILE nebo statických textových objektů, můžete použít existující HTML stránky nebo dokumenty, které jste vytvořili programově a uložili na disk. HTML kód můžete vytvářet v paměti s použitím textových proměnných ve formuláři.

Svázání HTML objektů s metodou 4D

Bez ohledu na to jak do vaší aplikace předáte HTML, můžete HTML objekty odkazovat pomocí 4D Method napsáním odkazů (link) pro tyto objekty. URL objektu musí být /4DMETHOD/Název_Metody, kde Název_Metody je název metody projektu 4D, která se má spustit při klepnutí na HTML objekt. Příklady metod projektu přiřazených k objektům HTML jsou v části Web služby, Poprvé (část II).

Důležité

- Pokud pošlete HTML soubor, můžete svázat metodu 4D s jakýmkoli typem HTML objektu, který umožňuje odkazy. Nezapomeňte, aby jste docílili POST, musíte mít na stránce HTML tlačítko Submit k provedení POST akce HTML stránky (stránka je potvrzena na Web server 4D).

- Na druhou stranu, pokud zahrnete HTML kód do formuláře 4D jako statický text nebo textovou proměnnou, nemůžete použít tlačítka Submit. Můžete mít odkazy (link) na metody 4D, ale nemůžete provést POST akci, protože 4D vytvořila stránku za vás.

Svázání HTML objektů s proměnnými 4D - část 1

HTML objekty můžete svázat s proměnnými 4D.

Poznámka: Pracujete s proměnnými procesu.

Za prvé, HTML objekt může mít hodnotu dosazenou z proměné 4D.

Za druhé, po zpětném poslání Web stránky do 4D může být hodnota objektu HTML přiřazena k proměnné 4D. K tomu v HTML zdroji stránky vytvoříte HTML objekt, jehož název je stejný jako název proměnné 4D, se kterou jej chcete svázat. Tato možnost je popsána v další části tohoto dokumentu "Svázání HTML objektů s proměnnými 4D - část 2".

Poznámka: V nekontextním módu se není možno odkazovat na obrázkové proměnné 4D. Jestli chcete vědět více informací, přečtěte si část Web služby, Nekontextní mód.

Hodnota HTML objektu může být inicializována z proměnné 4D, programem můžete přiřadit hodnotu proměnné jako výchozí hodnotu HTML objektu předáním [PromNázev] do **hodnoty** pole HTML objektu. PromNázev je název proměnné procesu jak je definován v procesu Web spojení. Toto je název který odevzdáte do hranatých závorek [].

Poznámka: Od 4D 6.5 můžete použít standarní HTML zápis pro předání proměnných 4D do statických stránek: <!--4DVAR MaProm→. K použití tohoto zápisu musíte označit patřičné možnosti v Předvolbách databáze (část Web služby, Web server nastavení).

Zápis [PromNázev] vám umožňuje vložit data 4D kamkoliv do stránky. Můžete třeba napsat:

<P>Vítejte v [vtNázStrán]!</P>

Hodnota 4D proměnné vtNázStrán bude vložena do HTML stránky.

Tip: Zápis [PromNázev] můžete používat opakovaně. Jestliže textová hodnota v proměnné PromNázev obsahuje další správný odkaz [...], 4D jej nahradí správnou hodnotou, pokud byla proměnná nalezena.

Zde je příklad:

```
   ` Následující část kódu 4D přiřadí "4D4D" do proměnné procesu vs4D
vs4D:="4D4D"
   ` Pak je poslána HTML stránka "AnyPage.HTM"
SEND HTML FILE("AnyPage.HTM")
```

Zdroj stránky AnyPage.htm je zde:

```
<html>

<head>
<title>anyPage</title>
<SRCIPT LANGUAGE="JavaScript">
<!--
function Is4DWebServer(){
return (document.frm.vs4D.value=="4D4D")
}
function HandleButton(){
if (Is4DWebServer()){
alert("Jste připojen k Web Serveru 4D!")
} else {
alert("Nejste připojen k Web serveru 4D!")
}
}
//-->
</head>

<body>

<form action="/4DMETHOD/WWW_STD_FORM_POST" method="POST"
name="frm">
    <input type="hidden" name="vs4D" value="[vs4D]"><p><a
    href="JavaScript:HandleButton()"><img src="AnyGIF.GIF"></a></p>
    <p><a href="JavaScript:HandleButton()"><input type="submit"
    name="b0K" hodnota="OK"></a></p>
</form>
</body>
</html>
```

V kontextním módu, před posláním HTML stránky, 4D vždy zkontroluje zdrojový kód HTML, aby zjistila, jestli neobsahuje nějaké odkazy na proměnné 4D k přemapování URL odkazů (probereme dále).
Ve zdrojovém kódu HTML si všimněte **skrytého** vstupního objektu vs4D. Hodnota tohoto objektu je nastavena na textovou hodnotu "[vs4D]". Protože metoda projektu posílající tuto HTML stránku před jejím odesláním nastavila hodnotu proměnné vs4D, bude v HTML zobrazeno "4D4D", tj.hodnota proměnné vs4D.

Ve stránce zahrnutá JavaScript funkce is4DWebServer testuje hodnotu HTML objektu vs4D. Zde je trik: jestliže je HTML stránka poslána 4D, hodnota objektu se změní na "4D4D". Nicméně, pokud je stránka poslána jinou aplikací, objektu zůstane hodnota definovaná ve stránce. Bingo! s pomocí JavaScriptu můžete testovat, jestli byla stránka poslána na prohlížeč pomocí 4D.

První příklad nám ukazuje, jak vytvořit "inteligentní" stránku, která obsahuje dodatečné možnosti pokud je posílána pomocí 4D, ale uchovává si stále kompatibilitu s jinými Web servery.

Důležité: Svazovat s objekty HTML můžete pouze procesní proměnné. Dále vám počáteční verze 6.5 neumožňuje svázat array 4D s HTML SELECT objektem. Na druhou stranu vám umožňuje přiřadit každý prvek objektu SELECT jednotlivým proměnným 4D (první prvek do V1, druhý do V2, atd.). Array s proměnnou délkou je nutno řešit semi-dynamickou stránkou a metodou ve 4D, tímto způsobem.

Svazování proměnných s objekty HTML pracuje ze 4D směrem k prohlížeči s libovolnou metodou vnořování HTML (SEND HTML FILE, SEND HTML BLOB, statický text nebo textová proměnná ve formuláři 4D).

Zahrnutí JavaScript

4D podporuje zdrojový kód JavaScript zahrnutý do HTML dokumentu a také JavaScript.js dokumenty zahrnuté v HTML dokumentech (například <SCRIPT SRC="...").

S pomocí příkazů SEND HTML FILE a SEND HTML BLOB pošlete připravené HTML stránky nebo stránky vytvořené programově a uložené na disk. V obou případech máte plnou kontrolu nad stránkou. JavaScript můžete vložit v části HEAD dokumentu stejně jako v části možnosti formuláře (Form markup). V předchozím příkladu se script odkazoval na formulář "frm" protože jste měli možnost pojmenovat formulář. Můžete také spustit, přijmout nebo odmítnout potvrzení formuláře na úrovni formuláře Form markup.

Pokud do formulářů 4D zahrnete HTML, nemáte kontrolu nad částí HEAD nebo deklarací FORM. Rozsah scriptu je proto rozdílný. Například nemůžete zpřistupit HTML formulář pomocí jeho názvu. Nicméně porovnejte JavaScript funkci is4DWebServer z předchozího příkladu s následující:

```
function IS4DWebServer() {
return (document. forms[0].vs4D.value== "4D4D")
}
```

Obě funkce dělají to samé ale druhý příklad používá vlastnost objektu *forms* HTML dokumentu k zpřístupnění objektu přes prvek forms(0). Prqacuje to i když neznáte název přeložené HTML stránky (formuláře), který může ve 4D mít nebo mu tato může přidělit,.

Poznámka: 4D podporuje Java applet transport.

Svázání HTML objektů s proměnnými 4D - část 2

Pokud posíláte stránku s pomocí příkazu SEND HTML FILE, můžete také svázat proměnné 4D s objekty HTML směrem "Web prohlížeč k 4D". Svázání funguje oběma směry: když je HTML stránka potvrzena, 4D zpět zkopíruje hodnoty HTML objektů do proměnných 4D.

Upozornění: Získání hodnot zpět do procesních proměnných 4D je možné pouze u stránek poslaných příkazy SEND HTML FILE a SEND HTML BLOB. S HTML zahrnutými do formuláře 4D je zpětné získávání hodnot omezeno pouze na skutečné 4D objekty.

Předpokládejme následující zdrojový HTML kód:

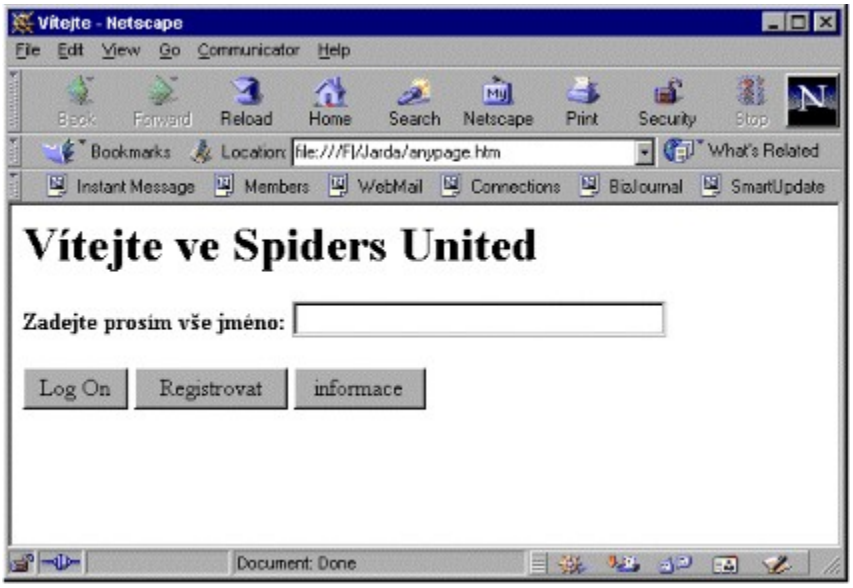
```
<script language="JavaScript"><!--
<!--

function GetBrowser_Information(formObj){
     formObj.vtNav_appName.value=navigator.appName
     formObj. vtNav_appVersion.value= navigator. appVersion
     formObj. vtNav_appCodeName.value= navigator. appCodeName
     formObj. vtNav_userAgent.value= navigator. UserAgent
}
function LogOn(formObj){
     if(formObj.vtUserName.value!=""){
     return true
}else{
     alert("Vložte své jméno a pak zkuste znovu.")
     return false
     }
}
//-- >
// -→</script>

<form action="/4DMethod/WWW_STD_FORM_POST" method="POST"
name="frmWelcome" onsubmit="return GetBrowser_Information(frmwelcome)">
    <input type="hidden" name="vtNav_appName" value>
     <input type="hidden" name="vtNav_appVersion" value>
     <input type="hidden" name="vtNav_appCodeName" value>
     <input type="hidden" name="vtNav_userAgent" value>
<p><font size="6"><strong>Vítejte ve SpidersUnited</strong></font></p>
     <p> </p>
     <p><font size="3"><strong>Zadejte prosím Vaše Jméno : </strong></font><input
     type="text" size="20" name="vtUserName" value="[vtUserName]"></p>
     <p> </p>
```

```
    <p><font size="2">
    <input type="submit" name="vsbLogOn" value="Log On" onclick="return logon(frmWelcome)">
    <input type="submit" name="vsbRegister" value="Registrovat">
    <input type="submit" name="vsbInformation" value="Informace">
</font></p>
</form>
```

Jakmile 4D pošle stránku na prohlížeč, bude vypadat následovně:



Hlavní možnosti této stránky jsou:

• Obsahuje tři tlačítka potvrzení: vsbLogOn, vsbRegister a vsbInformation.

• Pokud klepnete na Log On, potvrzení formuláře je nejdříve provedeno funkcí JavaScriptu LogOn. Pokud nebylo předáno žádné jméno, formulář není odeslán do 4D a zobrazí se JavaScript upozornění.

• Formulář má POST metodu 4D stejně jako script potvrzení který kopíruje vlastnosti Navigátoru do čtyř skrytých objektů, jejichž názvy začínají na vtNav_App.

• Počáteční hodnota objektu vtUserName je [vtUserName].

Předpokládejme, že metoda projektu WWW Welcome, která posílá tuto stránku, používá příklad SEND HTML FILE. Tato metoda je volána metodou databáze Při Web spojení.

```
    ` Metoda projektu WWW Welcome
    ` WWW Welcome → Logické
    ` WWW Welcome → Ano =Může začít práce
C_BOOLEAN($0)
$0:=False
    ` Skrytý INPUT HTML objekt vrací informace prohlížeče
C_TEXT(vtNav_appName;vtNav_appVersion;vtNav_appCodeName; vtNav_userAgent)
vtNav_appName:=""
vtNav_appVersion:=""
vtNav_appCodeName:=""
vtNav_userAgent:=""
    ` Text INPUT HTML kam je zadáno jméno uživatele
C_TEXT(vtUserName)
vtUserName:=""
```

```
    ` HTML hodnoty   tlačítek potvrzení
C_STRING(31;vsbLogOn;vsbRegister;vsbInformation)
Repeat
      ` Nezapomeňte resetovat hodnoty tlačítek!
   vsbLogOn:=""
   vsbRegister:=""
   vsbInformation:=""
      ` Poslat Web stránku
   SEND HTML FILE("Vítejte.HTM")
      ` Testovat hodnotu tlačítek aby se zjistilo na které bylo klepnuto
   Case of
         ` Bylo klepnuto na Log On
      ÷ (vsbLogOn # "")
         QUERY([WWW Users];[WWW Users]User Name=vtUserName)
         $0:=(Records in selection([WWW Users])>0)
         If ($0)
            WWW POST EVENT ("Log On";WWW Log information )
               ` WWW POST EVENT
               ` uloží informace do tabulky databáze
         Else
            CONFIRM("Neznámé jméno uživatele. Chcete se registrovat?")
            $0:=(OK=1)
            If ($0)
               $0:=WWW Register
                  ` Metoda WWW Register umožní Web uživateli novou registraci
            End if
         End if
            ` Klepnuto na tlačítko Registrovat
      ÷ (vsbRegister # "")
         $0:=WWW Register
         ` Klepnuto na Informace
      ÷ (vsbInformation # "")
         DIALOG([User Interface];"WWW Information")
   End case
Until (Not(<>vbWebServicesOn) | $0)
```

Možnosti této metody jsou:

• Proměnné *4D vtNav_appName, vtNav_appVersion, vtNav_appCodeName* a *vtNav_userAgent* (svázané s HTML objekty stejného názvu) používají JavaScript GetBrowser_Information k získání hodnot přiřazených jím k HTML objektům. Jednoduché a přímé, metoda nastaví proměnné jako řetězec a po potvrzení Web stránky získá jejich hodnoty.

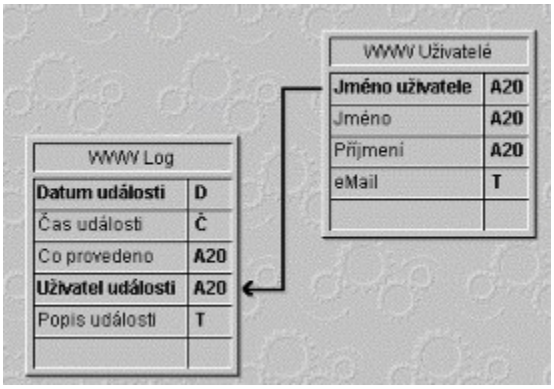
• Proměnné 4D vsbLogOn, vsbRegister a vsbInformation jsou přiřazeny ke třem tlačítkům potvrzení. Všimněte si, že tyto proměnné jsou nastaveny pokaždé když je stránka poslána na Web prohlížeč. Jakmile je provedeno potvrzení jedním z těchto tlačítek, prohlížeč vrátí hodnotu tlačítka do 4D. Protože proměnné jsou nastaveny pokaždé, ta která nebude prázdná, vám řekne které tlačítko bylo použito. Ostatní dvě proměnné jsou prázdný řetězec, ne proto že prohlížeč vrátí prázdný řetězec, ale protože prohlížeč o nich nic "neříká". Proto je 4D nechá nezměněné. To je důvod, proč je potřeba je nastavit při každém poslání na Web.

Toto je způsob jak odlišit, které tlačítko potvrzení bylo stisknuto, pokud jich je na Web stránce více. Všimněte si, že 4D tlačítka ve formuláři 4D jsou číselné proměnné, ale v HTML jsou všechny objekty textové. Proto při vrácení do 4D je potřeba testovat textovou hodnotu přiřazené svázané proměnné 4D.

Pokud svážete proměnnou 4D s objektem SELECT, svazujete opět textovou proměnnou. K zjištění který prvek rozevíracího seznamu byl vybrán, testujete ve 4D číselnou hodnotu array. V HTML je ale do proměnné 4D svázané s HTML objektem vrácena hodnota vybrané položky.

Nezáleží na tom, který objekt svážete s proměnnou 4D, ale vždy je vracena textová hodnota. Proto musíte přiřazovat do řetězcových nebo textových proměnných 4D.

Zajímavé na tomto příkladu je, že po získání informací o prohlížeči, můžete tyto hodnoty uložit do tabulky 4D a opět tak kombinovat možnosti Web a 4D. Toto dělá metoda projektu WWW POST EVENT. Neposílá událost: ukládá informace o práci Webu do tabulek ukázaných zde:



Po uložení informací do tabulky, můžete použít jiné metody projektu k poslání informací zpět k Web uživateli. K tomu použijte QUERY pro vyhledání správných informací a DISPLAY SELECTION pro zobrazení záznamů. Následující obrázek ukazuje načtené informace dostupné pro registrovaného uživatele na straně Web:

S možností vázání proměnných a objektů ukázaných v tomto příkladu, zkombinovaných se všemi informacemi které můžete poslat a získat od uživatele pomocí HTML dialogu a formuláře 4D, můžete přidat některé zajímavé možnosti do vaší Web aplikace.

Svázání HTML objektů s proměnnými 4D - část 3

Jak bylo ukázáno v části Web služby, Podpora HTML, pokud je použit formulář 4D jako Web stránka, 4D provede pro typy “neviditelná tlačítka” vytvoření buď řady obrázkových tlačítek nebo jednoho obrázaku (imagemap) s odkazy (dle typu převáděného objektu) , pokud jsou tyto překryty obrázkem.

Pokud pošlete HTML dokument s pomocí příkazu SEND HTML FILE nebo SEND HTML BLOB, můžete k získání informací svázat proměnnou 4D s Image Map HTML objekty (INPUT TYPE= "IMAGE"). Můžete například

vytvořit ImageMap HTML objekt nazvaný bImageMap. Pokaždé když klepnete na tento obrázek, je na Web server posláno potvrzení (submit) s pozicí klepnutí. K získání souřadnice klepnutí (vyjádřené od levého vrchního okraje image) vám stačí svázat proměnnou procesu 4D bImageMape a dále proměnné bImageMap_X a bImageMap_Y, které vrací (jako text) vodorovné a svislé souřadnice klepnutí.

V HTML stránce napíšete něco jako:

<P><INPUT TYPE="image SRC=MujImage.GIF NAME="bMapaSveta" BORDER=0></P>

Do metody 4D která posílá HTML stránku vložte následující:

```
bImageMap:=""
bImageMap_X:=""
bImageMap_Y:=""
SEND HTML FILE("TatoStranka.HTM")
```

Pak v POST akce metody 4D nebo v platné metodě, za provedením příkazu SEND HTML FILE("..") při navrácení POST akce, můžete získat souřadnice následujícm způsobem:

```
$vlX:=Num(bImageMap_X) ` Get horizontal coordinates in numeric form
$vlY:=Num(bImageMap_Y) ` Get vertical coordinates in numeric form
If (($vlX#0)&($vlY#0))
   ` Provést něco podle souřadnic
End if
```

Odkazy na soubory a URL (kontextní mód)

V kontextním módu, k udržování ID kontextu a podkontextu, 4D automaticky přetvoří odkazy souborů a URL. Například, 4D přemapuje všechny IMG a HREF odkazy na místní soubory.

Pokud předáte váš vlastní HTML kód do formuláře 4D s použitím textové proměnné, musíte postupovat podle syntaxe zápisu přemapování 4D.

Místí GIF soubory jsou přemapovány jako "/4DBin/_/GIF_soubor_cesta", kde GIF_soubor_cesta je plná cesta ke GIF souboru podle cesty na jednotku kde je soubor umístěn.

Příklad

Následující metoda 4D vrací přemapované odkazy pro cestu obdrženou jako parametr:

```
   ` Metoda projektu WWW Local GIF URL
   ` WWW Local GIF URL Projekt ( Text )
   ` WWW Local GIF URL ( Nativní cesta ) → URL k místnímu GIF souboru
C_TEXT($0;$1)
$0:="/4Bin/_/"+HTML Cesta($1)
```

Poznámka: Podrobnosti o mětodě HTML Cesta najdete u příkazu Mac to ISO.

Pak při předání HTML kódu do formuláře 4D pomocí textové proměnné můžete napsat:

```
vtHTML:=Char(1)+"<P><IMG SRC="+Char(34)+WWW Local GIF URL(F:.HTM"
+Char(34)+"ALIGN=MIDDLE></P>+Char(13)
```

Tento kód vloží GIF dokument do formuláře 4D v místě proměnné vtHTML.

Důležité: K předání vlastního HTML kódu do formuláře 4D musíte napsat pouze tento typ 4D kódu. Pokud pouze pošlete HTML stránku pomocí SEND HTML FILE nebo použijete příkaz jako ADD RECORD, nezapomeňte že 4D dočasně překládá a přemapuje HTML.

Přemapování nemění odkazy které mají následující protokol:

- http:
- ftp:
- mailto:
- news:
- gopher:
- javascript:
- nntp:
- wais:
- prospero:

Dále si přečtěte

SEND HTML BLOB, SEND HTML FILE, Web služby, Nekontextní mód.

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Textové parametry předávané do 4D Metod volaných přes URL

Příkazy a odkazy pro Web server

Verze 6.0.2

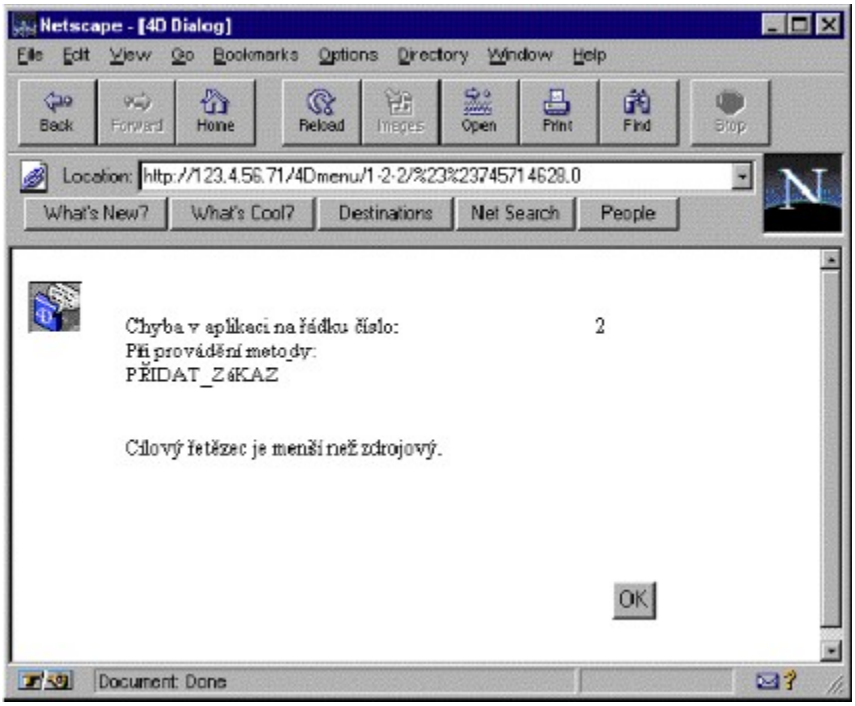
4th Dimension posílá textové parametry do jakékoli metody 4D volané přes specielní URL (4DMETHOD/, 4DACTION/...) v kontextním i nekontextním módu. Co se týká těchto textových parametrů:

• I když nebudete tyto parametry používat, musíte je deklarovat příkazem C_TEXT, jinak ve zkompilované databázi bude generována runtime chyba při přístupu k Web Serveru.

• Parametr $1 obsahuje extra data umístěná na koneci URL a může být rovněž použit jako způsob předání hodnot z prostředí HTML do prostředí 4D.

Runtime chyby ve zkompilované verzi

Uvažujme následující příklad. Provedete metodu svázanou s HTML objektem s odkazem (link) a na Web prohlížeč obdržíte následující obrazovku.



Tato chyba se týká chybějícího deklarace parametru $1 v metodě 4D která je volána při klepnutí na HTML objekt odkazující se k této metodě. Jako je kontext pro provedení je platná HTML stránka, chyba se vztahuje na "řádek 0" v metodě která poslala tuto stránku na Web prohlížeč.

Následující jsou příklady z části Web služby, Poprvé (část I), problému se vyvarujete explicitním deklarováním parametru $1 v metodách M_ADD_RECORDS a M_LIST_RECORDS:

```
   ` Metoda projektu M_ADD_RECORDS
C_TEXT($1) ` Tento parametr musí být deklarován explicitně
Repeat
   ADD RECORD([Customers])
Until(OK=0)
```

```
    ` Metoda projektu M_LIST_RECORDS
C_TEXT($1) ` Tento parametr musí být deklarován explicitně
ALL RECORDS([Customers])
MODIFY SELECTION([Customers])
```

Po provedení těchto změn se již runtime chyba nebude objevovat.

Parametry k deklarování ve volaných metodách 4D

Musíte deklarovat různé parametry podle podstaty a původu volané metody 4D.

• Metody databáze Při ověření WEB a Při Web spojení.

Musíte deklarovat šest parametrů pro spojení:

```
    ` Metoda databáze Při Web spojení
C_TEXT ($1;$2;$3;$4;$5;$6)
```

• Metoda volaná URL 4DMETHOD/

Musíte deklarovat jeden parametr:

```
    ` Metoda volaná URL 4DMETHOD/
C_TEXT ($1)
```

• Metoda volaná tagem 4DACTION/ jako URL:

Musíte deklarovat jeden parametr:

```
    ` Metoda volaná tagem 4DACTION/
C_TEXT ($1)
```

• Metoda volaná tagem 4DACTION/ jako HTML komentář v dokumentu

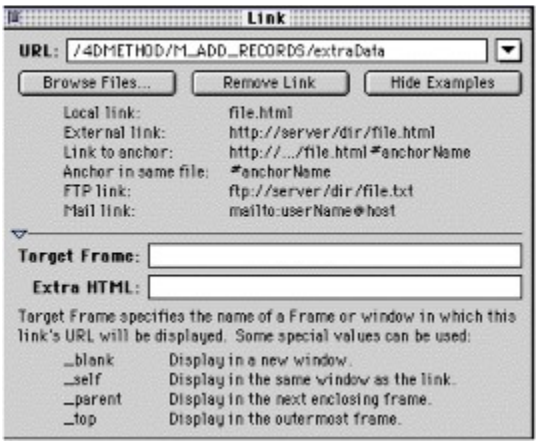
Metoda může vracet do $0 hodnotu. Musíte deklarovat parametr $1 a $0.

```
    ` Metoda volaná tagem 4DACTION/ jako HTML komentář
C_TEXT ($1;$0)
```

Práce s daty přidanými v URL

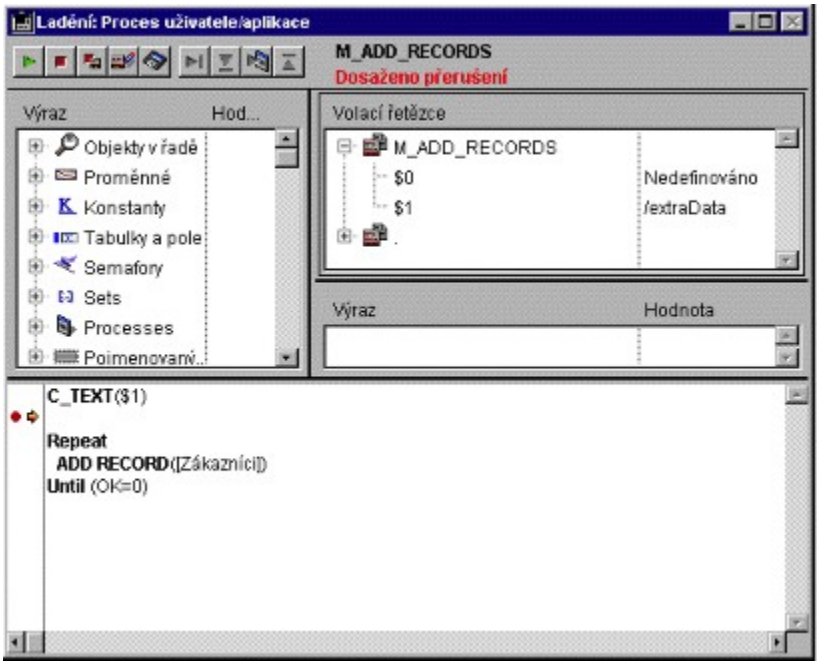
Textový parametr $1 předáný do metody 4D vrací přidaná data přiřazená k URL.

Opět následující příkad z Web služby, Poprvé (část II), změna je provedena do URL který odkazuje na metodu M_ADD_RECORDS:



Poznámka: Obrázek ukazuje změnu jak byla provedena s použitím ClarisHomePage na MacOs.

Přidaná data do URL jsou zde řetězec "/extraData". Po provedení této změny, můžete použít okno Ladění k rychlému testování, jestli hodnota $1 je skutečně řezězec "/extraData".



S použitím konvencí a algoritmů popsaných v části Metoda databáze Při Web spojení , máte prostředek k výměně patřičných dat mezi prostředími HTML a 4D pokud je metoda 4D volána HTML odkazem.

Jak dynamicky nastavit přidávání dat do URL

Pokud vaše vlastní HTML soubory vytváříte "za běhu" aplikace programem (např. příkazy Create document a SEND PACKET), jednoduše píšete URL přesně podle vašich potřeb.

Pokud pracujete s existujícím HTML souborem, můžete použít JavaScript k dynamickému nastavení odkazů k vašim objektům.

Dále si přečtěte

Web služby, Poprvé (část II).

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Web služby, Informace o Web site

Příkazy a odkazy pro Web server

Verze 6.5

4D vám umožňuje získat informace o funkčnosti vašeho Web site 4D.

• Web site můžete kontrolovat a řídit pomocí zvláštních URL (/4DSTATS, /4DHTMLSTATS a /4DCACHECLEAR).
• Můžete vytvořit Log soubor všech dotazů.

Řízení URL Web serverem

Web server 4D přijímá tři zvláštní URL: /4DSTATS, /4DHTMLSTATS a /4DCACHECLEAR.

Tyto URL jsou dostupné pouze pro Návrháře a Administrátora databáze. Pokud nebyl aktivován systém hesel 4D, jsou tyto URL dostupné všem uživatelům.

/4DSTATS

/4DSTATS vrací následující informace v textové formě:

• počet "přístupů" (hit - nízkoúrovňové spojení)
• počet vytvořených kontextů
• počet kontextů které nemohly být vytvořeny
• počet chyb hesla
• počet stránek uložených ve vyrovnávací paměti
• procenta použití vyrovnávací paměti
• seznam stránek a GIF image uložených ve vyrovnávací paměti statických stránek (*)

(*) Jestli chcete vědět více informací o vyrovnávací paměti, přečtěte si Web služby, Web server nastavení.

Tyto informace vám mohou pomoci zjistit funkčnost vašeho Web serveru a připadně upravit odpovídající parametry.

Poznámka: Příkaz WEB CACHE STATISTICS vám také umožní zjistit používání paměti pro statické stránky.

/4DHTMLSTATS

URL /4DHTMLSTATS má stejný účinek jako /4DSTATS a též vrací vše v textové formě. Je mezi nimi pouze ten rozdíl, že v posledním parametru (obsahující vyrovnávací paměť) obsahuje seznam pouze HTML stránek - ne GIF image.

/4DCACHECLEAR

URL /4DCACHECLEAR okamžitě vyčistí vyrovnávací paměť od statických stránek a image. Tím vám umožní okamžitě publikovat statickou stránku, která byla upravena a umístěn na disk.

Log soubor připojení

4D vám umožňuje získat log soubor dotazů na Web. Log je vytvořen do souboru s názvem "weblog.txt", který je umístěn na stejnou úroveň jako soubor struktury. Tento soubor je ve formátu CLF (Common Log Format) nebo NCSA formátu, rozpoznávaném většinou nástrojů pro analýzy Webu.

Každý řádek souboru reprezentuje dotaz, jako:

host rfc931 uživatel [DD/MMM/YYYY:HH:MM:SS] "dotaz" stav délka

Každé pole je odděleno mezerou a každý řádek končí CR/LF sekvencí (znak 13, znak 10).

• host: IP adresa klienta (např. 192.100.100.10)
• rfc931: informace nevytvořené 4D, jejím způsobem - (znaménko mínus)
• uživatel: ověřené jméno uživatele nebo jinak - (znaménko mínus). Jestliže název uživatele obsahuje mezery, budou nahrazeny _ (podtržítkem).
• DD: den, MMM: tří písmenný název měsíce (Led, Úno,...), YYYY: rok, HH: hodina, MM: minuty, SS: sekundy.
• Datum a čas jsou podle serveru.
• dotaz: dotaz poslaný klientem (např. GET /index.htm HTTP/1.0)
• stav: odpověď poslaná serverem
• délka: velikost vrácených dat (mimo HTTP záhlaví) nebo 0.

Poznámka: Z důvodů výkonnosti jsou operace před zapsáním na disk ukládány do paměťových bufferů po 1Kb. Operace jsou zapsány na disk jestliže není poslán žádný dotaz po 5 sekundách.

Možné hodnoty stavu jsou:
200: OK
204: Bez komentáře
302: poslání na jinou adresu
400: Nesprávný dotaz
401: vyžadováno ověření
500: Vnitřní chyba

Příklady řádků vytvořených v log dotazu

192.100.100.10 - - [25/Jan/1998:12:54:06] "GET /index.htm" 200 6524

Web klient jehož adresa byla 192.100.100.10 byl přijat. Požádal o stránku index.htm, která mu byla odeslána (obsahuje 6,524 bytech).

192.100.101.25 - - [25/Jan/1998:12:54:09] "GET /123456.htm" 404 125

Web klient jehož adresa byla 192.100.101.25 byl přijat. Požádal o stránku 123456.htm, ktrerá nebyla nalezena (4D poslala zprávu o 125 bytech).

192.100.101.31 - - [25/Jan/1998:12:54:10] "GET /secret.htm" 401 0

Web klient jehož adresa byla 192.100.101.31 byl přijat. Požádal o stránku secret.htm, server požadoval ověření.

192.100.101.31 - ZZZZ [25/Jan/1998:12:54:11] "GET /secret.htm" 401 0

Web klient jehož adresa byla 192.100.101.31 byl přijat jako ZZZZ. Požádal o stránku secret.htm, jméno uživatele bylo neznámé.

192.100.101.31 - ACI [25/Jan/1998:12:54:12] "GET /secret.htm" 200 2543

Web klient jehož adresa byla 192.100.101.31 byl přijat jako ACI. Požádal o stránku secret.htm, byla poslána (Obsahuje 2,543 bytů).

Upozornění: Log soubor může být naimportován do tabulky nebo přímo do 4D. Nicméně před importem musíte dočasně zastavit Web server.

Jako výchozí, není log soubor dotazů vytvářen. K žádosti na vytvoření log souboru pro všechny Web dotazy:

1. V předvolbách databáze na straně "Web server II" označte možnost "Požadovat log".

2. Klepněte na tlačítko **OK**.

Dále si přečtěte

WEB CACHE STATISTICS, Web služby, Web server nastavení.

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

START WEB SERVER

Příkazy a odkazy pro Web server

Verze 6.0

START WEB SERVER

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz START WEB SERVER zaktivuje vaší databázi na Intranetu nebo na Internetu s použitím vestavěného Web serveru 4D.

Jakmile je Web server zapnut, OK je nastavena na 1, jinak je nastavena na 0. Například, jestliže TCP/IP síťové komponenty nejsou aktivní, OK je nastavena na 0.

Dále si přečtěte

STOP WEB SERVER

Systémové proměnné a Sady

Pokud jsou Web služby správně zapnuty, je proměnná OK nastavena na 1, jinak je nastavena na 0.

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

STOP WEB SERVER

Příkazy a odkazy pro Web server

Verze 6.0

STOP WEB SERVER

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz STOP WEB SERVER zastaví publikování vaší databáze jako Web server. Jestliže byla databáze publikována jako Web site, všechny Web spojení jsou zastaveny a všechny Web procesy zastaveny.

Pokud databáze nebyla publikována jako Web site, příkaz neprovede nic.

Dále si přečtěte

START WEB SERVER.

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET WEB TIMEOUT (NASTAVIT WEB LIMIT)

Verze 6.0

SET WEB TIMEOUT (vyprší)

Parametr	Typ		Popis
vyprší	Číslo	→	čas vypršení Web spojení při nečinnosti, v sekundách

Popis

Příkaz SET WEB TIMEOUT nastaví čas, za který proces Web spojení skončí. Výchozí je nastaveno 5 minut.

Můžete omezit nebo zvýšit tento čas zadáním nového času do parametru *vyprší*.

Příkaz má okamžitý efekt a jeho rozsah jsou všechny procesy Web spojení.

• Pokud je SET WEB TIMEOUT proveden z procesu Web spojení, je čas použit pouze na tento proces.
• Pokud je SET WEB TIMEOUT není proveden z procesu Web spojení, je čas použit na všechna spojení.

Dále si přečtěte

Web služby, Procesy Web spojení.

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET HTML ROOT (NASTAVIT CESTU K HTML)

Verze 6.0

SET HTML ROOT (cestaHTML)

Parametr	Typ		Popis
cestaHTML	Řetězec	→	HTML cesta k výchozí složce s HTML soubory

Popis

Příkaz SET HTML ROOT nastaví výchozí složku kde bude 4D hledat HTML soubory, které předáte jako parametry do příkazu SEND HTML FILE.

Jako výchozí hledá 4D HTML soubory ve složce obsahující strukturu databáze.

Cesta kterou definujete musí být cesta HTML, kde složky jsou odděleny lomítkem ("/") a to bez rozdílu platformy. Jestli chcete vědět více informací o cestách HTML, přečtěte si část HMTL manuálu Přiručka jazyka,kterou seženete v knihkupectví.

Pokud zadáte nesprávnou cestu, je generována chyba OS File manažeru. Tuto chybu můžete zachytit pomocí metody instalované příkazem ON ERR CALL. Pokud zobrazíte upozornění nebo zprávu z metody chyby, zobrazí se tato zpráva na straně prohlížeče.

Poznámky:

• Příkaz SET HTML ROOT mění výchozí cestu HTML definovanou v předvolbách databáze. Jestli chcete vědět více informací, přečtěte si část Web služby, Bezpečnost připojení.
Příkaz SET HTML ROOT nemá žádný efekt pokud je Web server v nekontextním módu (podrobnější informace najdete v části Web služby, Nekontextní mód).

Příkad

Podívejte se na příklad u příkazu SEND HTML FILE.

Dále si přečtěte

ON ERR CALL.

Ovládání chyb

Pokud definujete nesprávnou cestu, je generována chyba OS File manažeru. Tuto chybu můžete zachytit pomocí metody instalované příkazem ON ERR CALL.

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET WEB DISPLAY LIMITS (NASTAVIT POČET K ZOBRAZENÍ WEB)

Příkazy a odkazy pro Web server

Verze 6.0

SET WEB DISPLAY LIMITS (početZáznamů{; početStran{; picRef}})

Parametr	Typ		Popis
početZáznamů	číslo	→	Maximální počet záznamů, zobrazený na každé stránce HTML seznamu záznamů
početStran	číslo	→	Maximální počet odkazů na stránky na spodu každé HTML stránky seznamu
picRef	číslo	→	číslo odkazu do knihovny, pro tlačítko odkazu na plný záznam

Popis

Příkaz SET WEB DISPLAY LIMITS upraví způsob jakým 4th Dimension zobrazí výběr záznamů pomocí příkazů DISPLAY SELECTION a MODIFY SELECTION na Web prohlížeči.

Pokud zobrazíte výběr záznamů s použitím 4th Dimension nebo 4D Client, program nenačte všechny záznamy výběru; načte pouze záznamy platně zobrazené v okně. Proto když vytvoříte výběr tisíce záznamů, je jejich zobrazení velmi rychlé. Při posunování seznamu, nebo při upravení velikosti okna, 4D opět načte patřičné záznamy.

Na Webu 4D rozdělí výběr záznamů na stránky. Bez systému stránkování, by výběr tisíce záznamů znamenal přenos tisíce záznamů po internetu nebo intranetu aby se zobrazili na jedinou stránku. Mohlo by to trvat dlouho a váš Web prohlížeč by pravděpodobě neměl dostatek paměti.

Jako výchozí zobrazuje 4D prvních 20 záznamů výběru a na konec HTML stránky umístí 20 odkazů na prvních 20 stránek výběru záznamů. To znamená,že jako výchozí můžete procházet 400 záznamů klepnutím na jednotlivé odkazy na konci stránky. Všiměte si, že tento stránkovací systém je proveden bez vašeho kódu; všecho obstarají příkazy DISPLAY SELECTION nebo MODIFY SELECTION.

SET WEB DISPLAY LIMITS vám umožní změnit tato nastavení. V parametru *početZáznamů* můžete zadat maximální počet záznamů, které chcete zobrazit na jednu stránku výběru. V parametru *početStran*, zadáte maximální počet stránek, které chcete zobrazit na konci stránky výběru.

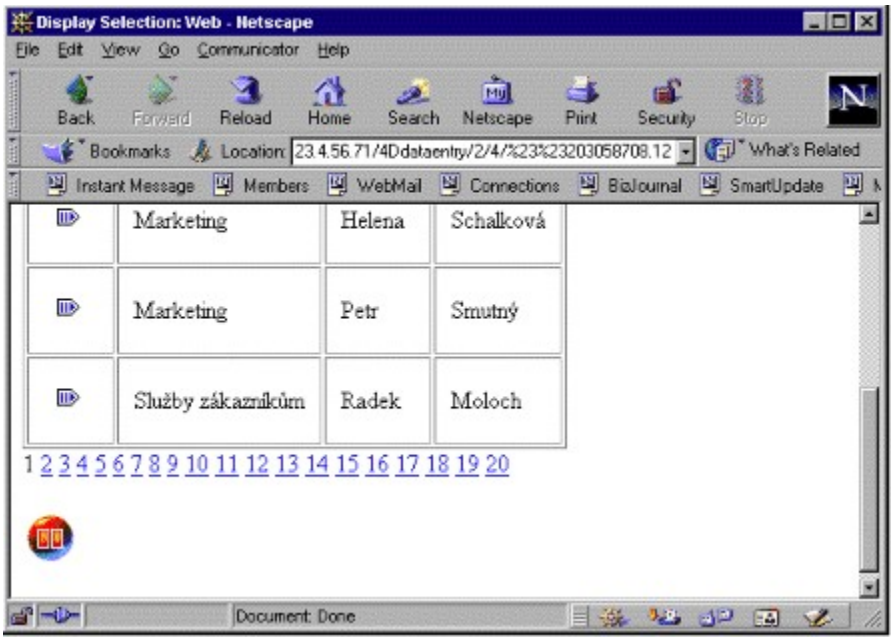
Poku máte například výběr 10000 záznamů a chcete jimi procházet v jednom výběru, můžete vložit do *početZáznamů* 100 a do *početStran* 100. Nicméně nezapomeňte, že data jdou přes síť nebo Internet; u Internetu musíte při upravování nastavení výběru vzít v úvahu rychlostní faktor.

SET WEB DISPLAY LIMITS může také změnit výchozí ikonu tlačítka plné stránky. Do parametru *picRef* zadejte číslo odkazu na obrázek uložený v knihovně obrázků.

SET WEB DISPLAY LIMITS má efekt pouze na následující provedení DISPLAY SELECTION nebo MODIFY SELECTION a jeho rozsah je místní v platném procesu.

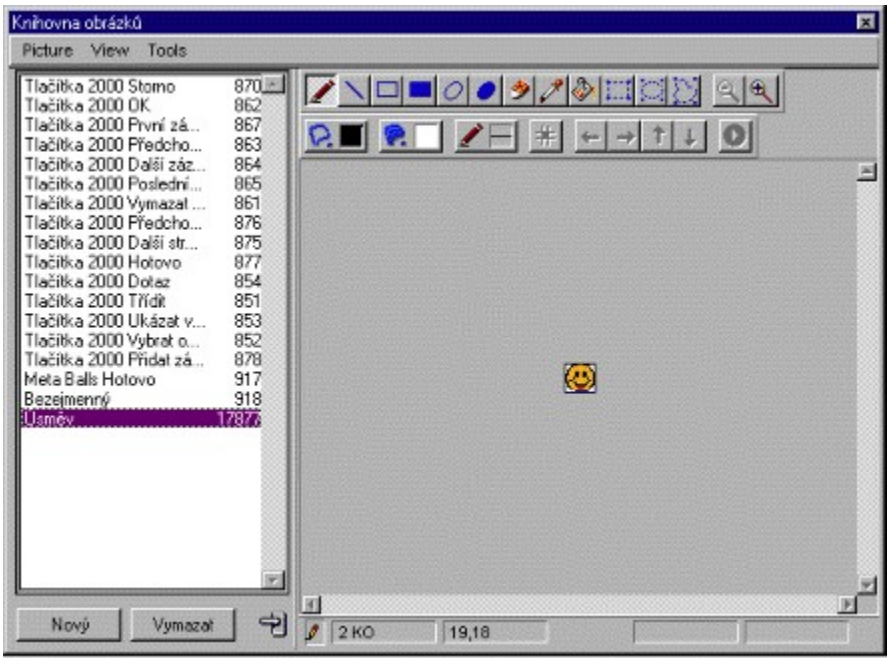
Příklad

V následujícím příkladu je DISPLAY SELECTION nebo MODIFY SELECTION použito na tabulku [PSČ]. Jako výchozí zobrazí 4D záznamy jak je ukázáno zde:



Všiměte si, že je ukázáno 400 záznamů.

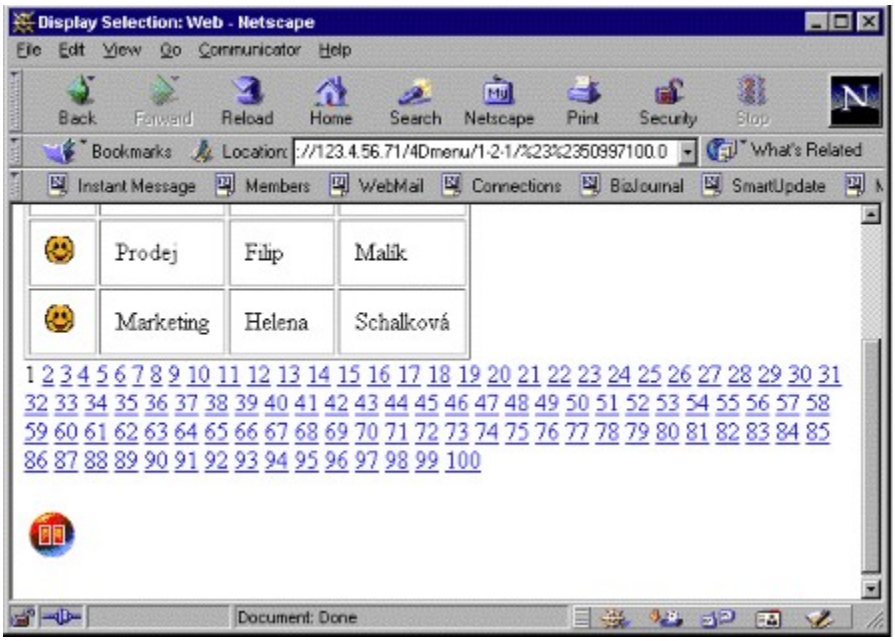
Jestliže je následující obrázek předán do knihovny obrázků:



A jestli metoda projektu, která zobrazí výběr, provede příkaz SET WEB DISPLAY LIMITS před příkazem DISPLAY SELECTION nebo MODIFY SELECTION:

```
SET WEB DISPLAY LIMITS (50;100;17877)
```

Pak bude výběr na prohlížeči vypadat následovně:



Nyní můžete procházet 5000 záznamů výběru.

Dále si přečtěte

DISPLAY SELECTION, MODIFY SELECTION.

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET HOME PAGE (NASTAVIT DOMOVSKOU STRÁNKU)

Verze 6.5

SET HOME PAGE (DomovStránka)

Parametr	Typ		Popis
DomovStránka	řetězec	→	Název stránky, nebo HTML cesta přístupu ke stránce, nebo "" k neodeslání uživatelské domovské stránky

Popis

Příkaz SET HOME PAGE vám umožní definovat vlastní domovskou stránku pro platný Web proces. Jako výchozí je v kontextním módu použito záhlaví nabídek 1 jako domovská stránka.

Definovaná stránka je přiřazená k Web procesu a proto můžete definovat několik odlišných domovských stránek. Například pro každého připojeného uživatele jinou. Tato stránka může být jak statická, tak dynamická.

Název nebo HTML cestu k domovské HTML stránce předáte do parametru *DomovStránka*.

K zastavení posílání této stránky pro platný Web proces, proveďte SET HOME PAGE s parametrem ("") jako prázdný řetězec.

Poznámka: 4D verze 6.5 vám umožňuje definovat výchozí domovskou stránku v Předvolbách databáze. V tomto případě je tato stránka použita na všechny Web procesy jak v kontextním tak v nekontextním módu.

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SEND HTML FILE (POSLAT HTML SOUBOR)

Příkazy a odkazy pro Web server

Verze 6.0

SEND HTML FILE (htmlSoubor)

Parametr	Typ		Popis
htmlSoubor	Řetězec	→	HTML cesta k HTML souboru nebo prázdný řetězec k zastavení SEND HTML FILE

Popis

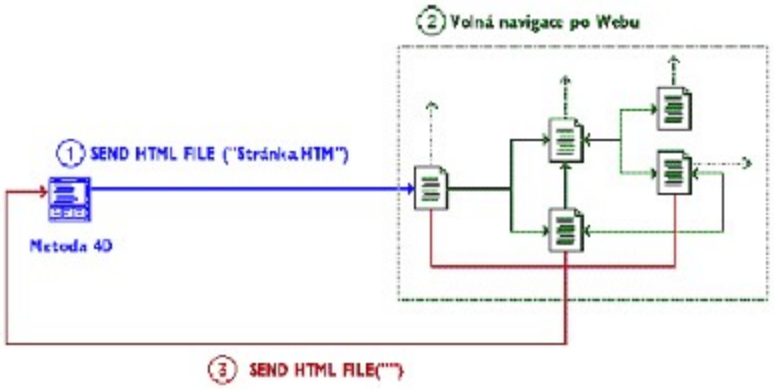
Příkaz SEND HTML FILE pošle na prohlížeč HTML stránku uloženou v HTML souboru, jejíž cestu zadáte do parametru *htmlSoubor*.

DŮLEŽITÉ: 4th Dimension očekává HTML dokumenty kódované ve Win1250.

Jako výchozí, bude 4th Dimension hledat HTML dokumenty ve stejné složce jako je struktura databáze. Můžete nastavit jinou složku pomocí příkazu SET HTML ROOT.

Pokud zadáte neplatnou HTML cestu, 4D pošle na Web prohlížeč zprávu "Požadovaná HTML stránka nemůže být nalezena".

Alternativní zápis SEND HTML FILE (""), je že předáte prázdný řetězec jako parametr. Tento zápis vám umožní skončit volání na SEND HTML FILE, který inicializuje HTML mód. Je to zobrazeno v následujícím obrázku:



1. Metoda 4D (projektu, objektu nebo databáze) provede SEND HTML FILE a tím pošle HTML dokument na prohlížeč.

2. Základní Web stránka může obsahovat odkazy na jiné stránky, nebo se může odkazovat na metodu 4D, která provede SEND HTML FILE k poslání jiné Web stránky. Tyto stránky mohou mít odkazy na další stránky a metody atd. Při pohybování po stránkách můžete použít ovládání prohlížeče, jako tlačítko Zpět.

3. Kterákoli ze stránek může obsahovat odkaz na metodu, která provede příkaz SEND HTML FILE(""), který pošle ukončí odesílání stránek a vy se vrátíte do metody 4D v místě za SEND HTML FILE , kterým odesílání statických stránek započalo.

Jakmile je SEND HTML FILE proveden, je obnovena systémová proměnná OK; pokud stránka existuje a nevypršel časový limit, OK je nastavena na 1. Jinak je nastavena na 0.

Poznámka: Pokud provedete příkaz SEND HTML FILE z procesu, který není Web procesem, příkaz neprovede nic a nevrátí žádnou chybu: je ignorován.

Příklady

1. Ve složce obsahující strukturu databáze je dokument s názvem "HomePage.HTM". Toto je stránka, kterou chcete poslat uživateli, když se připojí k databázi, místo záhlaví nabídek 1. K poslání této stránky, můžete do metody databáze Při Web spojení vložit následující kód:

```
` Metoda databáze Při Web spojení
SEND HTML FILE ("HomePage.HTM")
```

2. Vaše složka databáze je organizována následovně:

../Dokumenty/Práce/Databáze/MojeDB.4DB
../Dokumenty/Práce/Databáze/MojeDB.RSR
../Dokumenty/Práce/Databáze/MojeDB.4DD
../Dokumenty/Práce/Databáze/Web/HTM/HomePage.HTM

Stránku "HomePage.HTM" můžete poslat následovným způsobem:

```
SEND HTML FILE ("Web/HTM/HomePage.HTM")
```

nebo takto:

```
SET HTML ROOT ("Web/HTM/")
SEND HTML FILE ("HomePage.HTM")
```

3. Během práce na Web, přidáváte záznamy s použitím formuláře 4D. V tomto formuláři je tlačítko bHelp,jehož metoda objektu je následující:

```
` Metoda objektu bHelp
SEND HTML FILE ("Help.htm")
```

Od této stránky se můžete volně pohybovat po dalších Web stránkách vašeho systému nápovědy. Na každé stránce máte tlačítko potvrzení, nazvané Hotovo, které vám umožní dostat se zpět do zadávání dat. K tomu musí každá stránka obsahovat definici tlačítka:

```
<!-- bDone Tlačítko potvrzení →
<P><INPUT TYPE="submit" NAME="bDone" VALUE="Hotovo"></P>
```

stejně jako definici post akce:

```
<!-- Provést 4D htm_Help_Done při stisknutí tlačítka submit →
<FORM action="/4DMETHOD/htm_Help_Done" method="POST">
```

Na straně 4D přeruší metoda projektu htm_Help_Done posílání stránek z tlačítka bHelp.

```
` Metoda projektu htm_Help_Done
SEND HTML FILE ("")
```

Příkaz SEND HTML FILE v metodě objektu tlačítka bHelp je na posledním řádku metody. Jakmile je metoda dokončena, vrátíte se do zadávání dat.

Systémové proměnné a Sady

Pokud soubor k poslání existuje a časový limit nevypršel, OK je nastavena na 1 jinak je nastavena na 0.

Dále si přečtěte

SEND HTML BLOB, Web služby, Zahrnutí HTML a JavaScriptu, Web služby, Poprvé (část I), Web služby, Poprvé (část II).

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SEND HTML BLOB (POSLAT HTML BLOB)

Příkazy a odkazy pro Web server

Verze 6.5

SEND HTML BLOB (blob; typ{; bezKontextu})

Parametr	Typ		Popis
blob	BLOB	→	BLOB k odeslání prohlížeči
typ	řetězec	→	Datový typ MIME obsahu BLOB
bezKontextu	Logické	→	True = přepne do nekontextového módu 4D False = přepne do kontextového módu

Popis

Příkaz SEND HTML BLOB vám umožní poslat *blob* na prohlížeč.

Typ dat obsažený v BLOBu je definován v parametru *typ*. Tento parametr může být následujících typů:

• *typ* = **Prázdný řetězec** (""): V tomto případě nepotřebujete přidávat další informace do BLOBu. Prohlížeč se pokusí "pochopit" informace v BLOBu.

• *typ* = **Přípona souboru** (příklad: ".htm", ".gif", ".jpeg", atd.): V tomo případě definujete data v BLOBu předávanou příponou. Prohlížeč se pokusí "pochopit" data podle předáné přípony. Tato přípona ale musí být typu, který prohlížeč běžně přijímá.

• *typ* = **MIME/Typ** (příklad: "text/html", "image/tiff", atd.): V tomto případě přímo definujete typ dat obsažených v BLOBu. Toto řešení vám dává více svobody. Vedle standardních typů MIME můžete po Intranetu poslat vlastní typ dokumentu. K tomu musíte pouze nastavit prohlížeč tak, aby byl schopen rozeznat poslaný typ a aby mohl otevřít patřičnou aplikaci. Hodnota kterou předáte do *typ*, v tomto případě, "aplikace/x-[NázevTypu]". Na prohlížeči klienta, odkážete tento typ a přiřadíte jej k akci "Spustit aplikaci". Příkaz SEND HTML BLOB vám v tomto případě umožní poslat jakýkoli typ dokumentu. Intranet klient automaticky otevře přiřazenou aplikaci.

Poznámka: Pokud je BLOB typu "text/html" (.htm, .html, .shtm, .shtml), je přeložen a pochopen jako HTML soubor.

Zde je seznam nejpoužívanějších typů MIME:

Přípona	***Určení/Typ***
.htm	*text/html*
.html	*text/html*
.shtml	*text/html*
.shtm	*text/html*
.css	*text/css*
.pdf	*application/pdf*
.rtf	*application/rtf*
.ps	*application/postscript*
.eps	*application/postscript*
.hqx	*application/mac-binhex40*
.js	*application/javascript*
.txt	*text/plain*
.text	*text/plain*
.gif	*image/gif*
.jpg	*image/jpeg*

.jpeg	*image/jpeg*
.jpe	*image/jpeg*
.jfif	*image/jpeg*
.pic	*image/pict*
.pict	*image/pict*
.tif	*image/tiff*
.tiff	*image/tiff*
.mpeg	*video/mpeg*
.mpg	*video/mpeg*
.mov	*video/quicktime*
.moov	*video/quicktime*
.aif	*audio/aiff*
.aiff	*audio/aiff*
.wav	*audio/wav*
.ram	*audio/x-pn-realaudio*
.sit	*application/x-stuffit*
.bin	*application/x-stuffit*
.z	*application/x-zip*
.zip	*application/x-zip*
.gz	*application/x-gzip*
.tar	*application/x-tar*

Parametr *bezKontextu* vám umožní řici 4D jestli chcete přepnout z kontextního módu do nekontextního a naopak. K použití nekontextního módu, vložete do parametru True. K použití kontextního módu, vložte do parametru False. Pokud je parametr vynechán, je jako výchozí použit kontextní mód. Odkazy na proměnné 4D a tagy 4DACTION jsou analyzovány za jakéhokoli módu.

Poznámka: Více informací o nekontextním módu najdete v kapitole Web služby, Nekontextní mód.

Příklad

Podívejte se na příklad u příkazu PICT TO GIF.

Dále si přečtěte

SEND HTML FILE, Web služby, Nekontextní mód.

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Web context (Web kontext)

Příkazy a odkazy pro Web server

Verze 6.5

Web context → Logické

Parametr	Typ		Popis
Tento příkaz nevyžaduje žádné parametry.			
Výsledek funkce	Logické	←	True = Kontextní mód False = Nekontextní mód

Popis

Příkaz Web kontext musí být proveden z Web procesu. Vrací logickou hodnotu, která definuje jestli je Web spojení prováděno v kontextním módu (True) nebo nekontextním módu (False).

Poznámka: Provedeno z jiného než Web procesu, bude příkaz vždy vracet False.

Použití této funkce je probíráno v metodě databáze Při Web spojení.

Poznámka: Více informací o nekontextním módu najdete v části Web služby, Nekontextní mód.

Příklad

Zde je příklad metody databáze Při Web spojení:

```
If (Web Context) `Pokud kontextní mód
   WithContext ($1;$2;$3;$4;$5;$6)
Else
   NoContext ($1;$2;$3;$4;$5;$6)
End if
```

Dále si přečtěte

Metoda databáze Při Web spojení, PROCESS PROPERTIES, Web služby, Nekontextní mód, Web služby, Procesy Web spojení.

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET HTTP HEADER (NASTAVIT ZÁHLAVÍ HTTP)

SET HTTP HEADER (poleHTTP)

Parametr	Typ		Popis
poleHTTP	řetězec	→	PoleHTTP v záhlaví HTTP k naplnění pomocí 4D

Popis

Příkaz SET HTTP HEADER vám umožní nastavit pole v HTTP záhlaví k odeslání z 4D na prohlížeč. Tento příkaz má efekt pouze ve Web procesu v nekontextním módu.

Tento příkaz vám umožňuje ovládat "cookies".

Do parametru *poleHTTP* předáte pole záhlaví HTTP typu Text, které chcete nastavit.

Poznámka: Pole musí být vždy oddělena cr/lf sekvencí (carriage return/line feed). Podrobnější informace najdete v R.F.C (Reguest For Comments), které mohou být nalezeny na následující Internetové adrese: www.w3c.org.

Pokud nedefinujete stav, bude automaticky HTTP/1.0 200 OK.

Pole Charset, Expires, Content-Length, Content-Type, Date a Last-Modified jsou automaticky nastaveny 4D.

Příklad

Zde je příklad vlastního "cookie":

```
SET HTTP HEADER ("SET-COOKIE: USER="+String(Abs(Random))+"; PATH/")
```

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SEND HTTP REDIRECT (POSLAT HTTP PŘESMĚROVÁNÍ)

Příkazy a odkazy pro Web server

Verze 6.5

SEND HTTP REDIRECT (url{; *})

Parametr	Typ		Popis
url	řetězec	→	Nové URL
*	*	→	je-li uvedena = URL není překládána, vynechána = URL je překládána v speciálních znacích

Popis

Příkaz SEND HTTP REDIRECT vám umožní převést jednu URL na jinou.

Parametr *url* obsahuje nový URL který vám umožní přesměrovat dotaz. Pokud tento parametr je URL k souboru, musí obsahovat odkaz na tento soubor jako v nekontextním módu. Například: SEND HTTP REDIRECT ("/MojeStr.htm").

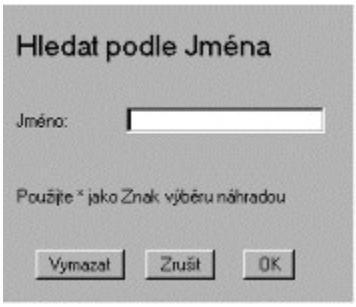
Pokud je tento příkaz volán v kontextním módu, Web proces spojení je odstraněn hned po provedení příkazu. Tento příkaz převládá nad příkazy posílajícími data (SEND HTML FILE, SEND HTML BLOB, atd.), které mohou být ve stejné metodě.

Tento příkaz vám také umožní přesměrovat dotaz na jiný Web server.

4D automaticky odkóduje specielní znaky URL. Pokud předáte znak *, 4D nebude překládat specielní znaky.

Příklad

Tento příkaz můžete použít k vlastnímu hledání ve 4D pomocí statických stránek. Představte si, že máte ve vaší statické stránce umístěny následující prvky:



Poznámka: POST akce "/4dcgi/rech" je přiřazena k textové oblasti a k tlačítkům **OK** a **Zrušit**.
V Metodě databáze Při Web spojení do části ovládající nekontextní mód předáte:

```
Case of
   ÷ ($1="/4dcgi/rech") ` Když 4D obdrží URL
         ` Jestliže bylo použito tlačítko OK a pole name obsahuje hodnotu
      If ((bOK="OK") & (name # ""))
            ` Změnit URL k provedení kódu dotazu,
            ` umístěném dále ve stejné metodě
         SEND HTTP REDIRECT("/4dcgi/rech?"+name)
      Else
            ` Jinak vrátit na původní stránku
```

```
            SEND HTTP REDIRECT("/page1.htm")
        End if
        ...
    ÷ ($1="/4dcgi/rech?@") `Jestliže URL bylo přesměrováno
        ... ` Zde umístit kód dotazu
End case
```

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

WEB CACHE STATISTICS (STATISTIKA WEB CACHE)

Verze 6.5

WEB CACHE STATISTICS (stránky; hitů; užití)

Parametr	Typ		Popis
stránky	Text Array	→	Názvy nejčastěji používaných stránek
hitů	Longint Array	→	Počet dostupů ke každé stránce
užití	číslo	→	Procenta využití paměti cache

Popis

Příkaz WEB CACHE STATISTICS vám umožní získat informace o nejpoužívanějších stránkách načtených ve vyrovnávací paměti Web serveru. Tato statistika se týká pouze statických stránek, GIF a JPEG obrázků pod 100Kb a stylů (.css).

Poznámka: Předpokládejme že chcete vytvořit semi-dynamicku stránku která bude zobrazovat statistiku o vyrovnávací paměti Web serveru. K tomu umístěte do statické HTML stránky s názvem "stats.htm" následující tag:

<!--4DACTION/STATS→. Pak předáte dvě proměnné 4D - *vPages* a *vUsage.* V metodě projektu STATS, napíšete následující kód:

```
C_TEXT ($1)
ARRAY TEXT (pages;0)
ARRAY LONGINT (hits;0)
C_LONGINT (vUsage)
WEB CACHE STATISTICS(pages;hits;vUsage)
vPages:=Char(1)
For ($i;1;Size of array(pages))
      ` Pro každou stránku umístěnou v paměti
   vPages:=vPages+pages{$i}+"   "+String(hits{$i})+"<br>"
      ` Vloží název a HTML kód stránky
End for
SEND HTML FILE("stats.shtm")
   ` Obsah stránek s příponou ".shtm" je vždy rozebrán.
```

Dále si přečtěte

Web služby, Web server nastavení.

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

OPEN WEB URL (OTEVŘÍT WEB URL)

Příkazy a odkazy pro Web server

Verze 6.5

OPEN WEB URL (url{; *})

Parametr	Typ		Popis
url	řetězec	→	Počáteční URL a spuštění prohlížeče na tomtéž počítači
*	*	→	je-li uvedena = URL není překládána, vynechána = URL je překládána v speciálních znacích

Popis

Příkaz OPEN WEB URL spustí váš Web prohlížeč a otevře jej na adrese URL předáné do parametru *url*.

Pokud byl Web prohlížeč otevřen před provedením tohoto příkazu:

• Na Windows je otevřen další výskyt prohlížeče a zobrazí stránku specifikovanou URL.
• Na Macintoshi nahradí specifikovaná stránka platnou stránku.

Pokud na jednotkách připojených k počítači není žádný prohlížeč, příkaz neprovede nic.

4D automaticky odkóduje specifické znaky URL. Pokud předáte znak *, 4D nebude překládat specialní znaky URL. Tato možnost vám umožní zpřístupnit a poslat URL typu "http://www.srver.net/stránka.htm?q=něco".

Poznámka: Příkaz nedělá nic pokud je volán z Web procesu.

Příklady

1. Když je proveden následující řádek kódu:

```
OPEN WEB URL ("file:///D:/web soubor.htm")
```

Je spuštěn Web prohlížeč a URL je přeložen do "file:///D%3A/web%20soubor.htm".

2. Pokud je proveden následující řádek kódu:

```
OPEN WEB URL (file:///D:/web soubor.htm";*)
```

Web prohlížeč je spuštěn a URL bude "file:///D:/web soubor.htm"

3. Následující řádek kódu spustí prohlížeč a předá do něj domovskou stránku ACI:

```
OPEN WEB URL ("http://www.aci-4d.com/")
```

Příkazy a odkazy pro Web server

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Řízení oken

Příkazy a odkazy pro Okna

Verze 6.0

Okna jsou použita k zobrazení iformací pro uživatele. Mají tři hlavní použití: k zadávání dat, k zobrazení dat a k informování uživatele zprávami a dialogy.

Vždy je otevřeno alespoň jedno okno. Pokud je potřeba, jsou předány posuvníky, aby uživateli umožnily posunovat formulář, který je větší než otevřené okno. V prostředí uživatele zobrazí okno seznam záznamů (výstupní formulář) nebo obrazovku pro vstup dat (vstupní formulář). V prostředí vlastních nabídek toto okno zobrazí úvodní obrazovku (vlastní obrázek).

Pokud v prostředí vlastních nabídek provedete příkaz nabídky, může být úvodní obrazovka nahrazena daty příkazu, který zobrazí formulář. Jakmile příkaz skončí provádění, je opět zobrazena úvodní obrazovka.

S příkazem Open window můžete otevřít několik typů oken. Pokud již nepotřebujete vlastní okno, můžete jej uzavřít příkazem CLOSE WINDOW, nebo klepnutím na zavírací políčko, pokud existuje.

Některé příkazy otevírají své vlastní okno. Příkazy jako GRAPH TABLE, REPORT a PRINT LABEL otevřou okno, které bude oknem na popředí.

Pokud začnete nový proces a neotevřete okno na začátku metody procesu, 4D automaticky jedno otevře ve chvíli, kdy je zobrazen formulář.

Dále si přečtěte

Open window, Typy oken.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Open window (Otevřít okno)

Příkazy a odkazy pro Okna

Verze 6.0 (Upraveno)

Open window (levá; vrch; pravá; spodní{; typ{; titul{; řídícíokénko}}}) { → WinRef}

Parametr	Typ		Popis
levá	číslo	→	levá souřadnice otevíraného okna v okně aplikace
vrch	číslo	→	vrchní souřadnice otevíraného okna v okně aplikace
pravá	číslo	→	pravá souřadnice otevíraného okna v okně aplikace, nebo -1 pro použití velikosti z vlastností formuláře
spodní	číslo	→	spodní souřadnice otevíraného okna v okně aplikace, nebo -1 pro použití velikosti z vlastností formuláře
typ	číslo	→	typ otevíraného okna
Titul	řetězec	→	Titulek v okně nebo "" pro užití titulku formuláře z vlastností
řídícíokénko	řetězec	→	Název metody, která bude volána při klepnutí na uzavírací okénko, nebo při poklepání na řídící nabídku okna
Výsledek funkce	WinRef	←	Číslo odkazu na okno

Popis

Open window otevře nové okno s rozměry zadanými do prvních čtyř parametrů:

• *levá* je vzdálenost v bodech od levého okraje okna aplikace k levému kraji okna.
• *vrch* je vzdálenost v bodech od vrchního okraje okna aplikace po vrchní okraj okna.
• *pravá* je vzdálenost v bodech od levého okraje okna k pravému okraji okna.
• *spodní* je vzdálenost v bodech od vrchního okraje okna k spodnímu okraji okna.

Pokud předáte 1 do parametrů *pravá* a *spodní*, řeknete 4D aby sama vytvořila velikost okna podle následujících podmínek:

• Máte navržený formulář a nastaveny jeho možnosti změny velikosti v okně Vlastnosti formuláře v prostředí návrháře.

• Před provedením Open window, označíte formulář pomocí INPUT FORM, ke kterému přiřadíte parametr *.

Důležité: Automatické nastavení velikosti okna bude možné pouze pokud provedete patřičné volání INPUT FORM pro zobrazení formuláře a předáte * jako parametr příkazu INPUT FORM.

• Parametr *typ* je volitelný. Můžete jím vybrat typ okna k zobrazení, který odpovídá hodnotám ukázaným v části Typy oken. Pokud je typ okna záporný, vytvořené okno je plovoucí okno. Pokud není definováno, je použit typ 1.

• Parametr *Titul* je volitelný název okna.

Pokud předáte prázdný řetězec ("") do titulu, řeknete 4D, aby použila jako název okna název formuláře nastavený ve Vlastnostech formuláře v prostředí návrháře.

Důležité: Výchozí název formuláře bude nastaven pro okno pouze když provedete patřičné volání INPUT FORM pro zobrazení formuláře a předáte * jako parametr příkazu INPUT FORM.

• Parametr *řídícíokénko* je volitelná metoda políčka zavření pro toto okno. Pokud je tento parametr definován, řídící okénko (Windows) nebo Zavírací políčko (Macintosh) je přidáno do okna. Pokud uživatel poklepe na Řídící okénko (Windows) nebo klepne na Zavírací políčko (Macintosh), je provedena metoda předáná parametrem *řídícíokénko*.

Poznámka verze 6: Zavírání okna můžete řídit i z metody formuláře pokud nastane událost Při zavření okna. Více informací najdete u příkazu Form event.

Pokud je pro proces otevřeno více než jedno okno, poslední otevřené je aktivní okno procesu (na popředí). Informace mohou být upravovány pouze v okně na popředí. Ostatní okna mohou být prohlížena. Když uživatel bude psát, aktivní okno se přesune na popředí, pokud tam již není.

Formuláře se zobrazují uvnitř otevřených oken. Text z příkazu MESSAGE se také objevuje v okně.

Příklady

1. Následující metoda projektu otevře okno ve středu hlavního okna (Windows) nebo hlavní obrazovky (Macintosh). Všiměte si, že můžete předat dva, tři nebo čtyři parametry alternativně:

```
   ` Metoda projektu OPEN CENTERED WINDOW
   ` $1 - Šířka okna
   ` $2 - Výška okna
   ` $3 - Typ okna (volitelné)
   ` $4 - Titul okna (volitelné)
$SW:= Screen width \cf1 2
$SH:=(Screen height\cf1 2)
$WW:=$1\cf1 2
$WH:=$2\cf1 2
Case of
   ÷ (Count parameters=2)
      Open window($SW-$WW;$SH-$WH;$SW+$WW;$SH+$WH)
   ÷ (Count parameters=3)
      Open window($SW-$WW;$SH-$WH;$SW+$WW;$SH+$WH;$3)
   ÷ (Count parameters=4)
      Open window($SW-$WW;$SH-$WH;$SW+$WW;$SH+$WH;$3;$4)
End case
```

Po napsání této metody ji můžete použít následovně:

```
OPEN CENTERED WINDOW (400;250;Movable dialog box;"Obnovit archiv")
DIALOG([Utility Table];"UPDATE OPTIONS")
CLOSE WINDOW
If (OK=1)
   ` ...
End if
```

2. Následující příklad otevře plovoucí okno které má metodu Řídícího políčka (Windows) nebo Zavíracího políčka (Macintosh). Okno je otevřeno v horním pravém rohu okna aplikace:

```
Open window(Screen width -149;33; Screen width -4;178;- Palette window;""; "CloseColorPalette")
DIALOG([Dialogs];"Color Palette")
```

Metoda *CloseColorPalette* provede příkaz CANCEL:

```
CANCEL
```

3. Následující příklad otevře okno, jehož velikost a titul budou z vlastností formuláře zobrazeného v okně:

```
INPUT FORM([Zákazníci];"Přidat Záznam";*)
Open window(10;80;-1;-1;Plain window;"")
Repeat
```

```
    ADD RECORD([Zákazníci])
Until (OK=0)
```

Připomínka: Automatické nastavení velikosti okna bude možné pouze pokud provedete patřičné volání INPUT FORM pro zobrazení formuláře a předáte * jako parametr příkazu INPUT FORM.

Dále si přečtěte

CLOSE WINDOW, Open external window, Open form window.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Typy oken

Verze 6.0

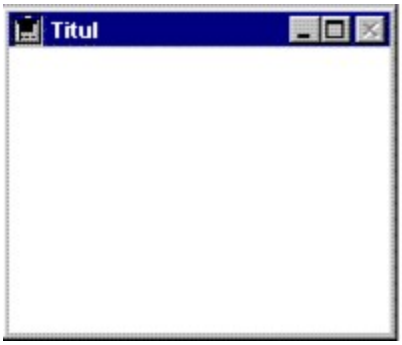
K definování typu okna v příkazu Open window můžete použít jednu z následujících předdefinovaných konstant:

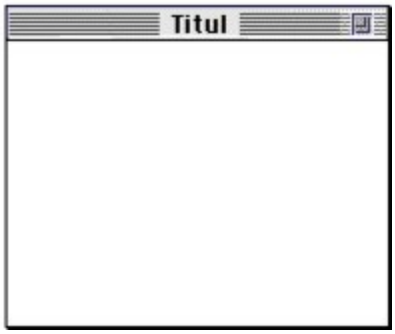
Konstanta	*Typ*	*Hodnota*	*Může být plovoucí okno*
Plain window	*Long Integer*	*8*	*Ne*
Plain no zoom box window	*Long Integer*	*0*	*Ne*
Plain fixed size window	*Long Integer*	*4*	*Ne*
Modal dialog box	*Long Integer*	*1*	*Ne*
Alternate dialog box	*Long Integer*	*3*	*Ano*
Movable dialog box	*Long Integer*	*5*	*Ano*
Plain dialog box	*Long Integer*	*2*	*Ano*
Palette window	*Long Integer*	*720*	*Ano*
Round corner window	*Long Integer*	*16*	*Ne*

Plovoucí okno: Pokud předáte jednu z těchto konstant do Open window, otevřete normální okno. K otevření plovoucího okna, vložte do Open window zápornou hodnotu typu okna .

Následující tabulka ukazuje každý typ okna na Windows (nalevo) a na Macintosh (napravo).

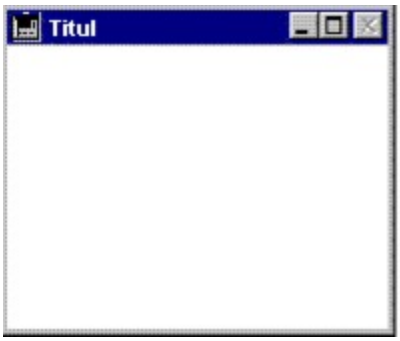
Normální okno (8)

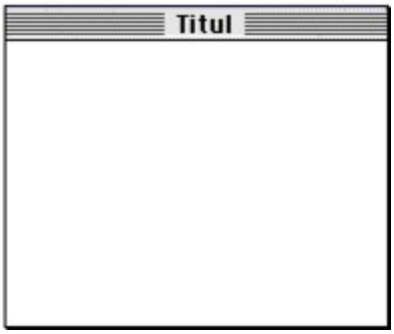




- Může mít titul: Ano
- Může mít zavírací políčko nebo ekvivalent: Ano
- Může být změněna velikost: Ano
- Může být minimalizováno/maximalizováno nebo přiblíženo: Ano
- Vhodné pro posuvníky: Ano
- Použití: vstup dat bez posuvníků, DISPLAY SELECTION, MODIFY SELECTION, atd.

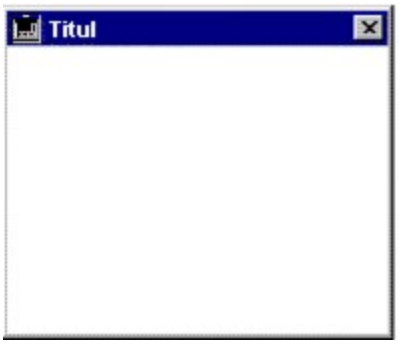
Normální okno bez automatické změny velikosti (0)

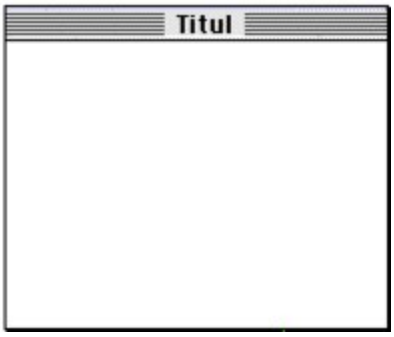




• Může mít titul: Ano
• Může mít zavírací políčko nebo ekvivalent: Ano
• Může být změněna velikost: Ano
• Může být minimalizováno/maximalizováno nebo přiblíženo: Ne na Macintoshi
• Vhodné pro posuvníky: Ano
• Použití: vstup dat s posuvníky, DISPLAY SELECTION, MODIFY SELECTION, atd.

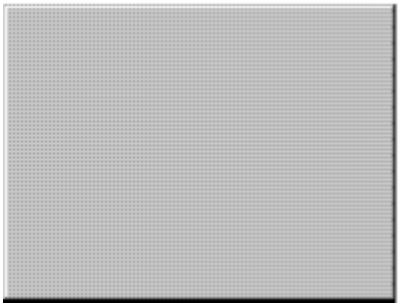
Normální okno s pevnou velikostí (4)





• Může mít titul: Ano
• Může mít zavírací políčko nebo ekvivalent: Ano
• Může být změněna velikost: Ne na Macintoshi
• Může být minimalizováno/maximalizováno nebo přiblíženo: Ano
• Vhodné pro posuvníky: Ano a Ne
• Použití: vstup dat s ADD RECORD (...;...) nebo ekvivalent

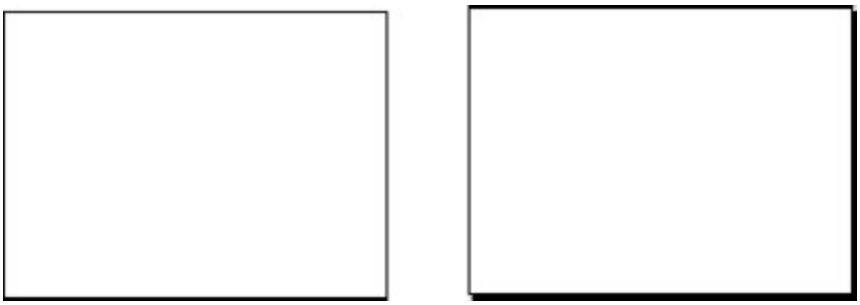
Modální dialogové okno (1)




• Může mít titul: Ne
• Může mít zavírací políčko nebo ekvivalent: Ne

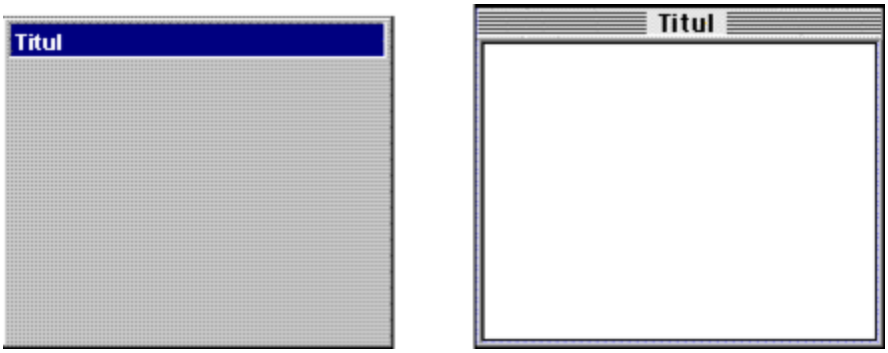
• Může být změněna velikost: Ne
• Může být minimalizováno/maximalizováno nebo přiblíženo: Ne
• Vhodné pro posuvníky: Ne
• Použití: DIALOG, ADD RECORD (...;...;*) nebo ekvivalent
• Okna tohoto typu jsou modální

Alternativní dialogové okno (3)



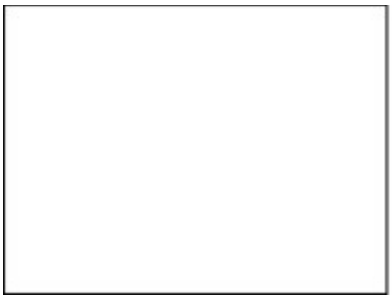
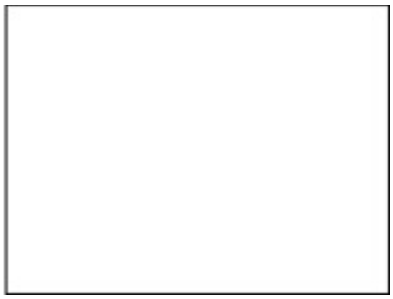
• Může mít titul: Ne
• Může mít zavírací políčko nebo ekvivalent: Ne
• Může být změněna velikost: Ne
• Může být minimalizováno/maximalizováno nebo přiblíženo: Ne
• Vhodné pro posuvníky: Ne
• Použití: DIALOG, ADD RECORD (...;...;*) nebo ekvivalent
• Okna tohoto typu jsou modální, pokud nejsou použity jako plovoucí okno.

Posuvné dialogové okno (5)



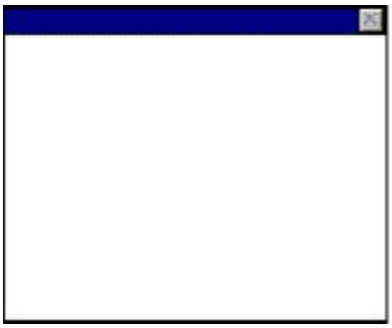
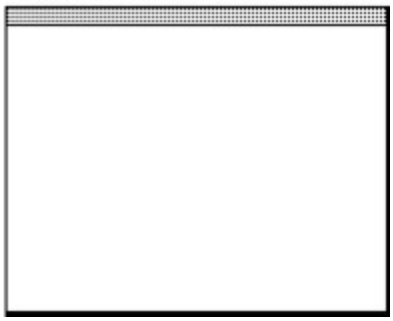
• Může mít titul: Ano
• Může mít zavírací políčko nebo ekvivalent: Ne
• Může být změněna velikost: Ne
• Může být minimalizováno/maximalizováno nebo přiblíženo: Ne
• Vhodné pro posuvníky: Ne
• Použití: DIALOG, ADD RECORD (...;...;*) nebo ekvivalent
• Okna tohoto typu jsou modální, ale lze s nimi pohybovat a mohou být plovoucí okna.

Normální dialogové okno (2)

• Může mít titul: Ne
• Může mít zavírací políčko nebo ekvivalent: Ne
• Může být změněna velikost: Ne
• Může být minimalizováno/maximalizováno nebo přiblíženo: Ne
• Vhodné pro posuvníky: Ne
• Použití: DIALOG, ADD RECORD (...;...;*) nebo ekvivalent, úvodní obrazovky
• Okna tohoto typu jsou modální, pokud nejsou použity jako plovoucí okno.

Okno palety (720+1+2+4+8)

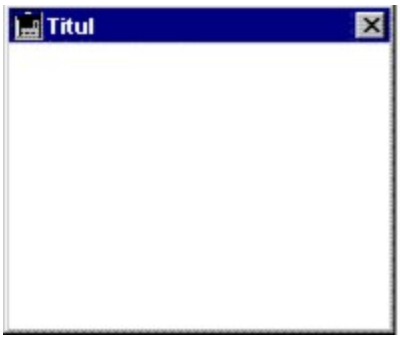



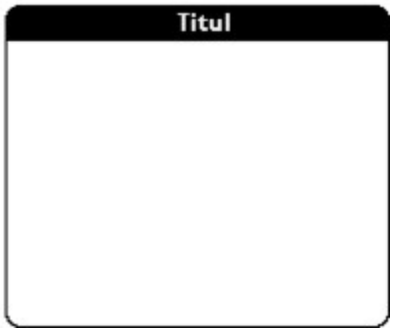
Pokud provedete Open window, můžete vložit jednu nebo více z následujících konstant do Okna palety. Docílít lze tím různého chování okna:

Konstanta	***Typ***	***Hodnota***
Has zoom box	*Long Integer*	*8*
Has grow box	*Long Integer*	*4*
Has window title	*Long Integer*	*2*
Has highlight	*Long Integer*	*1*

• Může mít titul: Ano, pokud je definovaná možnost titulu
• Může mít zavírací políčko nebo ekvivalent: Ano
• Může být změněna velikost: Ano, pokud je definována možnost políčka zvětšení
• Může být minimalizováno/maximalizováno nebo přiblíženo: Ano, pokud je definována možnost automatické změny (zoom)
• Vhodné pro posuvníky: Ano, pokud je definována možnost políčka zvětšení
• Použití: Plovoucí okna pro DIALOG nebo DISPLAY SELECTION (bez vstupu dat)

Okno se zakulacenými rohy (16)





• Může mít titul: Ano
• Může mít zavírací políčko nebo ekvivalent: Ano
• Může být změněna velikost: Ne na Macintoshi
• Může být minimalizováno/maximalizováno nebo přiblíženo: Ne
• Vhodné pro posuvníky: Ne na Macintoshi
• Použití: Vyjímečně

Dále si přečtěte

Open external window, Open window.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Open external window (Otevřít externí okno)

Příkazy a odkazy pro Okna

Verze 6.0 (upraveno)

Open external window (levá; vrch; pravá; spodní; typ; titlul; OblastPlugIn) → WinRef

Parametr	Typ		Popis
levá	číslo	→	levá souřadnice otevíraného okna v okně aplikace
vrch	číslo	→	vrchní souřadnice otevíraného okna v okně aplikace
pravá	číslo	→	pravá souřadnice otevíraného okna v okně aplikace
spodní	číslo	→	spodní souřadnice otevíraného okna v okně aplikace
typ	číslo	→	typ otevíraného okna
Titul	řetězec	→	Titulek v okně
OblastPlugIn	řetězec	→	Název typu externí oblasti, příkaz pro volání zásuvného modulu
Výsledek funkce	WinRef	←	Číslo odkazu na okno

Popis

Open external window otevře nové okno a zobrazí externí oblast podporovanou přikazy pro *oblastPlugIn* obsaženými v Plug-in 4D.

Open external window vrací Long Inteeger hodnotu, která může být použita jako číslo odkazu na okno (které může být použito dalšími příkazy Oken) nebo jako číslo odkazu k externí oblasti zobrazené k okně (které může být použito dalšími příkazy Plug-in 4D).

Prvních šest parametrů je stejných jako u příkazu Open window. Nicméně žádý z těchto parametrů není volitelný.

Open external window vytvoří modelové okno. Příkaz nečeká na vstup uživatele a tak můžete mít vytvořeno několik aktivních oken najednou. Můžete mezi nimi přepínat a měnit data v okně na popředí. Pokud má okno titul, zobrazí se Řídící okénko (Windows) nebo Zavírací okénko (Macintosh) aby bylo uživateli umožněno zavřít okno.

Příklad

Následující příklad otevře externí okno a zobrazí externí oblast 4D Write:

```
wrWind:=Open external window (50; 50; 350; 450; 8; "Letter Writing"; "_4D WRITE")
```

Následující příklad zavře externí okno otevřené předchozím příkladem:

```
CLOSE WINDOW (wrWind)
```

Dále si přečtěte

CLOSE WINDOW, Open window.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SHOW WINDOW (UKÁZAT OKNO)

Verze 6.0**.5**

SHOW WINDOW ({okno})

Parametr	Typ	Popis	
okno	WinRef	→	Číslo odkazu na okno nebo pokud vynecháno okno na popředí

Popis

Příkaz SHOW WINDOW vám umožní zobrazit okno, jehož číslo předáte do okno. Pokud je tento parametr vynechán, je zobrazeno okno na popředí v platném procesu.

Pro provedení příkazu SHOW WINDOW musí být okno před tím skryto příkazem HIDE WINDOW. Pokud je okno již zobrazeno, příkaz neprovede nic.

Příklad

Podívejte se na příklad u příkazu HIDE WINDOW.

Dále si přečtěte

HIDE WINDOW.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

HIDE WINDOW (SKRÝT OKNO)

Příkazy a odkazy pro Okna

Verze 6.0**.5**

HIDE WINDOW ({okno})

Parametr	**Typ**		**Popis**
okno	WinRef	→	Číslo odkazu na okno nebo pokud vynecháno okno na popředí

Popis

Příkaz HIDE WINDOW vám umožní skrýt okno jehož číslo zadáte do parametru *okno*. Pokud je tento parametr vynechán, bude skryto okno na popředí v platném procesu. Například vám tento příkaz umožní zobrazit pouze aktivní okno v procesech které se skládají z několika procesů.

Okno zmizí z obrazovky, ale stále zůstane otevřené. Stále na něj můžete uplatnit programovací změny možné v oknech 4D.

K zobrazení okna skrytého příkazem HIDE WINDOW:

• Použijte příkaz SHOW WINDOW s předaným číslem odkazu na *okno*.
• Použijte seznam procesů z prostředí návrháře, vyberte proces ve kterém je okno skryto a z nabídky **Procesy** vyberte položku **Ukázat**.

Ke skrytí všech oken procesu, použijte příkaz HIDE WINDOW.

Příklad

Tento příklad je metoda tlačítka umístěného ve vstupním formuláři. Toto tlačítko otevře dialog v novém okně které patří stejnému procesu. V tomto případě může uživatel chtít skrýt ostatní okna procesu (vstupní formulář a palety) při zobrazení tohoto okna. Jakmile je dialog potvrzen, objeví se ostatní okna.

```
   ` Metoda objektu pro tlačítko "Information"
HIDE WINDOW(Entry) ` Skrýt vstupní okno
HIDE WINDOW(Palette) ` Skrýt paletu
$Infos:=Open window(20;100;500;400;8) ` Vytvořit informační okno
   ... ` Zde umístěte informace věnované řízení dialogu
If(OK=1) ` Když uživatel potvrdí dialog
   CLOSE WINDOW($Infos) ` Zavřít okno
   SHOW WINDOW(Entry)
   SHOW WINDOW(Palette) ` Zobrazit ostatní okna
End if
```

Dále si přečtěte

SHOW WINDOW.

CLOSE WINDOW (ZAVŘÍT OKNO)

Příkazy a odkazy pro Okna

Verze 3

CLOSE WINDOW ({RefExtOkno})

Parametr	Typ		Popis
RefExtOkno	WinRef	→	číslo odkazu na okno, které má být uzavřeno, nebo okno na popředí, není-li uvedeno nic

Popis

Příkaz CLOSE WINDOW zavře okno v platném procesu otevřené příkazem Open window. Příkaz neprovede nic, pokud není otevřeno vlastní okno; nemůže zavřít standarní okna. CLOSE WINDOW také neprovede nic ve chvíli, kdy je formulář aktivní v okně. Můžete použít CLOSE WINDOW pokud skončíte práci v okně otevřeném příkazem Open window.

Je potřeba vložit číslo jako parametr, jinak příkaz zavře okno, které bylo poslední vytvořeno příkazem Open window.

Pokud předáte číslo odkazu na externí okno, CLOSE WINDOW zavře definované externí okno. Jestli chcete vědět více informací o externích oknech, přečtěte si popis u funkce Open external window.

Příklad

Následující příklad otevře okno a přidá nové záznamy pomocí příkazu ADD RECORD. Jakmile jsou záznamy přidány, okno je zavřeno příkazem CLOSE WINDOW:

```
Open window (5; 40; 250; 300; 0; "Nový zaměstnanec")
Repeat
   ADD RECORD ([Zaměstnanci]) ` Přidat nový záznam
Until (OK = 0) ` opakovat dokud nezrušeno
CLOSE WINDOW ` Zavřít okno
```

Dále si přečtěte

Open external window, Open window.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ERASE WINDOW (VYMAZAT OKNO)

Verze 6.0 (upraveno)

ERASE WINDOW ({okno})

Parametr	Typ		Popis
okno	WinRef	→	číslo odkazu na okno k vymazání obsahu, nebo okno na popředí platného procesu, není-li uvedeno nic

Popis

Příkaz ERASE WINDOW vymaže obsah okna, jehož číslo odkazu předáte jako parametr.

Pokud vynecháte parametr okno, ERASE WINDOW vymaže obsah okna na popředí.

Obvykle budete ERASE WINDOW používat v kombinaci s příkazy MESSAGE a GOTO XY. V tomto případě vymaže příkaz obsah okna a posune kurzor do levého horního rohu (na pozici GOTO XY(0;0)).

Nepleťte si příkaz ERASE WINDOW, který vymaže obsah okna s příkazem CLOSE WINDOW, který odstraní okno z obrazovky.

Dále si přečtěte

GOTO XY, MESSAGE.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

REDRAW WINDOW (PŘEKRESLIT OKNO)

Verze 6.0

REDRAW WINDOW ({okno})

Parametr	Typ		Popis	
okno	WinRef	→	číslo odkazu (WinRef) na okno, nebo okno na popředí platného procesu	je-li vynecháno

Popis

Příkaz REDRAW WINDOW provede grafické obnovení okna, jehož číslo odkazu předáte jako parametr.

Pokud vynecháte parametr *Okno*, REDRAW WINDOW se provede na okno na popředí v platném procesu.

Poznámka: 4th Dimension provede obnovu okna při každém přesunutí okna, změně velikosti nebo přenesení na popředí, stejně jako při změně formuláře a/nebo hodnoty zobrazené v okně. Tento příkaz budete používat jen vyjímečně.

Dále si přečtěte

ERASE WINDOW.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

DRAG WINDOW (PŘETÁHNOUT OKNO)

Příkazy a odkazy pro Okna

Verze 6.0

DRAG WINDOW

Parametr	Typ	Popis

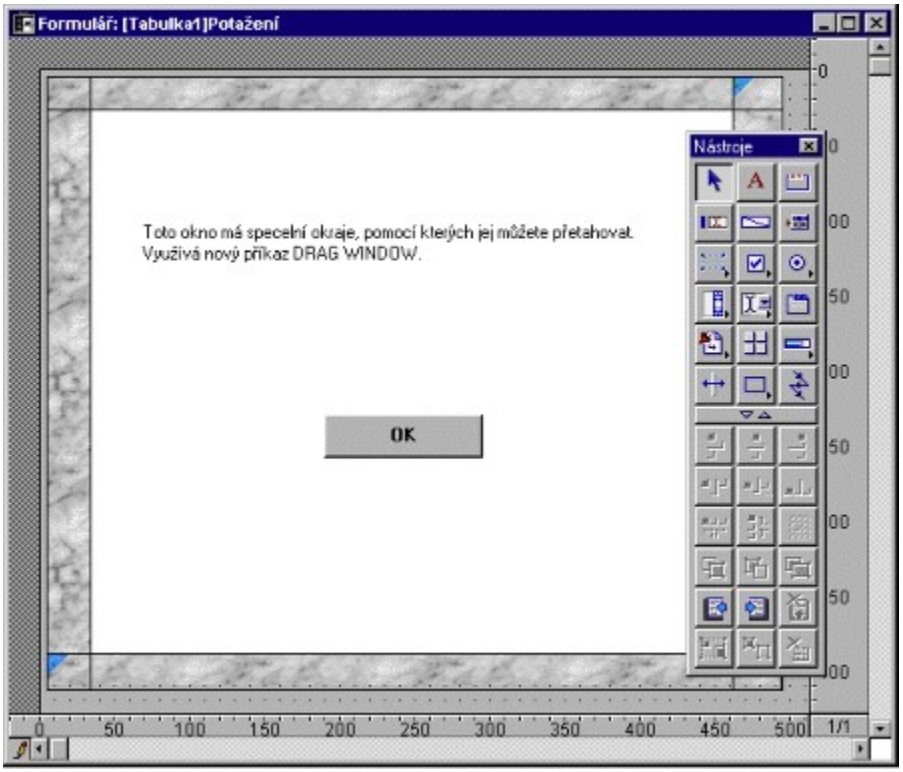
Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz DRAG WINDOW přetáhne platné okno podle pohybů myši. Obvykle budete tento příkaz volat z metody objektu, který bude ihned reagovat na klepnutí myši (neviditelné tlačítko).

Příklad

Následující formulář, ukázaný v prostředí návrháře, obsahuje rámeček vytvořený statickým obrázkem, přes který jsou čtyři neviditelná tlačítka pro každou stránku:



Každé tlačítko má následující metodu:

```
DRAG WINDOW ` Začít přetahovat okno po klepnutí
```

Po provedení následující metody v prostředí vlastní nabídky a uživatele:

```
Open window(50;50;50+400;50+300;2)
DIALOG([Table1];"Vlastní tažení")
CLOSE WINDOW
```

Obdržíte následující okno:

Nyní můžete okno přetáhnout klepnutím kamkoliv na okraj.

Dále si přečtěte

GET WINDOW RECT, SET WINDOW RECT.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Get window title (Získat název okna)

Příkazy a odkazy pro Okna

Verze 6.0

Get window title ({okno}) → Řetězec

Parametr	Typ		Popis
okno	číslo	→	Číslo odkazu (WinRef) na okno, nebo okno na popředí platného procesu, je-li vynecháno
Výsledek funkce	řetězec	←	Nadpis okna

Popis

Příkaz Get window title vrací titul okna, jehož číslo odkazu zadáte jako parametr *okno*. Pokud okno neexistuje, příkaz vrátí prázdný řetězec.

Pokud vynecháte parametr okno, Get window title vrátí titul okna na popředí.

Příklad

Podívejte se na příklad u příkazu SET WINDOW TITLE.

Dále si přečtěte

SET WINDOW TITLE.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET WINDOW TITLE (NASTAVIT NÁZEV OKNA)

Příkazy a odkazy pro Okna

Verze 6.0 (Upraveno)

SET WINDOW TITLE (titul{; okno})

Parametr	Typ		Popis
titul	řetězec	→	titul okna
okno	WinRef	→	číslo odkazu na okno, nebo okno na popředí v platném procesu, není-li uvedeno nic

Popis

Příkaz SET WINDOW TITLE nastaví titul okna, jehož číslo odkazu předáte do parametru *okno* na titul předány do parametru *titul* (max. délka 80 znaků). Pokud okno neexistuje, SET WINDOW TITLE neprovede nic. Pokud vynecháte parametr okno, je příkaz použit na okno na popředí platného procesu.

Poznámka: V prostředí uživatele mění 4th Dimension titul okna automaticky - na "Vstup pro Tabulka" při provádění vstupu dat. Pokud změníte titul okna, 4D jej pravděpodobně přepíše. Na druhou stranu v prostředí vlastních nabídek, 4th Dimension nezmění titul okna.

Příklad

Při vkládání dat ve formuláři, klepnete na tlačítko které provede dlouhou akci (např. prohlížení programově vztažených záznamů v podformuláři). Informace o průběhu budete zobrazovat v titulu okna:

```
   ` Metoda objektu bAnalysis
Case of
   ÷ (Form event=On Clicked)
      $vsCurTitle:=Get window title ` Uložit titul okna do proměnné
      FIRST RECORD([Invoice Line Items]) ` Začít dlouhou akci
      For($vlZaznam;1;Records in selection([Invoice Line Items]))
         UDELAT COSI
            ` ukázat teploměr
         SET WINDOW TITLE("Provádí se položka #"+String($vlZaznam))
      End for
         ` Vrátit původní titul okna
      SET WINDOW TITLE($vsCurTitle)
End case
```

Dále si přečtěte

Get window title.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

HIDE TOOL BAR (SKRÝT LIŠTU NÁSTROJŮ)

Verze 6.0

HIDE TOOL BAR

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz HIDE TOOL BAR udělá lištu nástrojů neviditelnou.

Pokud byla lišta již skryta, příkaz neprovede nic.

Dále si přečtěte

HIDE MENU BAR, SHOW MENU BAR, SHOW TOOL BAR.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SHOW TOOL BAR (UKÁZAT LIŠTU NÁSTROJŮ)

Verze 6.0

SHOW TOOL BAR

Parametr	Typ	Popis

Tento příkaz nevyžaduje žádné parametry.

Popis

Příkaz SHOW TOOL BAR ukáže lištu nástrojů.

Pokud je lišta již viditelná, příkaz neprovede nic.

Dále si přečtěte

HIDE MENU BAR, HIDE TOOL BAR, SHOW MENU BAR.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

WINDOW LIST (SEZNAM OKEN)

Příkazy a odkazy pro Okna

Verze 6.0

WINDOW LIST (okna{; *})

Parametr	Typ		Popis
okna	Array	→	Array čísel odkazů na otevřená okna
*	*	→	je-li uveden, bere v úvahu i plovoucí okna je-li vynechán, bez plovoucích oken

Popis

Příkaz WINDOW LIST naplní array *okna* čísly odkazů na okna všech oken otevřených ve všech procesech (základní nebo uživatelské).

Pokud nepředáte parametr *, budou plovoucí okna ignorována.

Příklad

Následující metoda projektu vezme všechna otevřená okna, mimo plovoucích oken a dialogových oken:

```
   ` Metoda projektu TILE WINDOWS
WINDOW LIST($alWnd)
$vlLeft:=10
$vlTop:=80 ` Nechat dost místa pro lištu nástrojů
For ($vlWnd;1;Size of array($alWnd))
   If (Window kind($alWnd{$vlWnd}) # Modal Dialog)
      GET WINDOW RECT($vlWL;$vlWT;$vlWR;$vlWB;$alWnd{$vlWnd})
      $vlWR:=$vlLeft+($vlWR-$vlWL)
      $vlWB:=$vlTop+($vlWB-$vlWT)
      $vlWL:=$vlLeft
      $vlWT:=$vlTop
      SET WINDOW RECT($vlWL;$vlWT;$vlWR;$vlWB;$alWnd{$vlWnd})
      $vlLeft:=$vlLeft+10
      $vlTop:=$vlTop+25
   End if
End for
```

Poznámka: Tato metoda může být zlepšena přidáním testu na velikost hlavního okna (na Windows) nebo velikosti a umístění obrazovek (na Macintosh).

Dále si přečtěte

Window kind, Window process.

Window kind (Typ okna)

Verze 6.0

Window kind ({okno})

Parametr	Typ		Popis
okno	číslo	→	číslo odkazu (WinRef) na okno, nebo okno na popředí platného procesu, je-li vynecháno

Popis

Příkaz Window kind vrací typ okna, jehož číslo odkazu zadáte do parametru *okno*. Pokud okno neexistuje, příkaz vrátí 0 (nula).

Jinak může příkaz vrátit následující kostanty:

Konstanta	***Typ***	***Hodnota***
Regular window	*Long Integer*	*8*
Modal dialog	*Long Integer*	*9*
External window	*Long Integer*	*5*
Floating window	*Long Integer*	*14*

Pokud vynecháte parametr *okno*, příkaz bude použit na okno na popředí.

Příklad

Podívejte se na příklad u příkazu WINDOW LIST.

Dále si přečtěte

GET WINDOW RECT, Get window title, Window process.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Window process (Proces okna)

Příkazy a odkazy pro Okna

Verze 6.0

Window process ({okno}) → Číslo

Parametr	Typ		Popis
okno	číslo	→	číslo odkazu (WinRef) na okno, nebo okno na popředí platného procesu, je-li vynecháno
Výsledek funkce	číslo	←	číslo odkazu na proces

Popis

Příkaz Window process vrací číslo procesu ve kterém je okno ,jehož číslo odkazu zadáte do parametru *okno*. Pokud okno neexistuje, příkaz vrátí 0 (nula).

Pokud vynecháte parametr okno, bude příkaz použit na okno na popředí.

Dále si přečtěte

Current process.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GET WINDOW RECT (ZÍSKAT SOUŘADNICE OKNA)

Verze 6.0

GET WINDOW RECT (levá; vrch; pravá; spodní{; okno})

Parametr	Typ		Popis
levá	číslo	←	levá souřadnice okna v okně aplikace
vrch	číslo	←	vrchní souřadnice okna v okně aplikace
pravá	číslo	←	pravá souřadnice okna v okně aplikace
spodní	číslo	←	spodní souřadnice okna v okně aplikace
okno	WinRef	→	číslo odkazu na okno, nebo okno na popředí platného procesu, je-li vynecháno

Popis

Příkaz GET WINDOW RECT vrací globální souřednice okna, jehož číslo odkazu zadáte do parametru *okno*. Pokud okno neexistuje, zůstanou ostatní parametry nezměněné.

Pokud vynecháte parametr *okno*, bude příkaz použit na okno na popředí.

Souřadnice jsou vyjádřeny vzhledem k levému hornímu rohu obsahu okna aplikace (Windows) nebo hlavní obrazovky (Macintosh). Souřadnice odpovídají oblasti obsahu okna (mimo lišty titulu a okrajů).

Příklad

Podívejte se na příklad u příkazu WINDOW LIST.

Dále si přečtěte

SET WINDOW RECT.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SET WINDOW RECT (NASTAVIT RÁMEC OKNA)

Verze 6.0

SET WINDOW RECT (levá; vrch; pravá; spodní{; okno})

Parametr	Typ		Popis
levá	číslo	→	nová levá souřadnice okna v okně aplikace
vrch	číslo	→	nová vrchní souřadnice okna v okně aplikace
pravá	číslo	→	nová pravá souřadnice okna v okně aplikace
spodní	číslo	→	nová spodní souřadnice okna v okně aplikace
okno	WinRef	→	číslo odkazu na okno, nebo okno na popředí platného procesu, je-li vynecháno

Popis

Příkaz SET WINDOW RECT mění globální souřednice okna, jehož číslo odkazu zadáte do parametru *okno*. Pokud okno neexistuje, příkaz neprovede nic.

Pokud vynecháte parametr *okno*, bude příkaz použit na okno na popředí.

Příkaz může změnit velikost nebo přesunout okno podle zadaných souřednic.

Souřadnice jsou vyjádřeny vzhledem k levému hornímu rohu obsahu okna aplikace (Windows) nebo hlavní obrazovky (Macintosh). Souřadnice odpovídají oblasti obsahu okna (mimo lišty titulu a okrajů).

Upozornění: Při použití tohoto příkazu buďte opatrní, protože můžete okno přesunout mimo okno aplikace (Windows) nebo mimo obrazovky (Macintosh). Aby jste tomu zabránili, použijte příkazy jako Screen width a Screen height ke kontrole nových souřednic okna.

Příklad

Podívejte se na příklad u příkazu WINDOW LIST.

Dále si přečtěte

DRAG WINDOW, GET WINDOW RECT.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Frontmost window (Okno na popředí)

Příkazy a odkazy pro Okna

Verze 6.0

Frontmost window {(*)} → WinRef

Parametr	Typ		Popis
*	*	→	je-li uvedena, bere do úvahy plovoucí okna je-li vynechána, ignoruje plovoucí okna
Výsledek funkce	WinRef	←	číslo odkazu na okno

Popis

Příkaz Frontmost window vrací číslo odkazu na okno pro okno na popředí.

Dále si přečtěte

Frontmost process, Next window.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Next window (Další okno)

Příkazy a odkazy pro Okna

Verze 6.0

Next window (okno) → Číslo

Parametr	Typ		Popis
okno	číslo	→	číslo odkazu (WinRef) na okno, nebo okno na popředí platného procesu, je-li vynecháno
Výsledek funkce	číslo	←	číslo odkazu na další okno za testovaným

Popis

Příkaz Next window vrací číslo odkazu na okno pro okno "za" oknem, jehož číslo odkazu předáte do parametru *okno* (založené na vrstvení oken).

Dále si přečtěte

Frontmost window.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Find window (Najít okno)

Příkazy a odkazy pro Okna

Verze 6.0

Find window (levá; vrch{; okničást}) -→> WinRef

Parametr	Typ		Popis
levá	číslo	→	levá souřadnice v okně aplikace
vrch	číslo	→	vrchní souřadnice v okně aplikace
okničást	číslo	←	číslo části okna, v této pozici (pouze Macintosh)
Výsledek funkce	WinRef	←	číslo odkazu na okno, nalezené v této pozici

Popis

Příkaz Find window vrací (pokud nějaké) číslo odkazu na okno které bylo první na souřadnicích zadaných do parametrů *levá* a *vrch*.

Souřadnice musí být vyjádřeny vzhledem k hornímu levému kraji obsahu okna aplikace (Windows) nebo hlavní obrazovky (Macintosh).

Pokud zadáte parametr *okničást*, ať už okno bylo nebo nebylo nalezeno, parametr vrátí jednu z následujících konstant:

Konstanta	***Typ***	***Hodnota***	***Platforma***
In menu bar	*Long Integer*	*1*	*Pouze Macintosh*
In system window	*Long Integer*	*2*	*Pouze Macintosh*
In contents	*Long Integer*	*3*	*Windows nebo Macintosh*
In drag	*Long Integer*	*4*	*Pouze Macintosh*
In grow	*Long Integer*	*5*	*Pouze Macintosh*
In go away	*Long Integer*	*6*	*Pouze Macintosh*
In zoom box	*Long Integer*	*7*	*Pouze Macintosh*

Dále si přečtěte

Frontmost window, Next window.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

MAXIMIZE WINDOW (MAXIMALIZOVAT OKNO)

Verze 6.0**.5**

MAXIMIZE WINDOW ({okno})

Parametr	**Typ**		**Popis**
okno	WinRef	→	číslo odkazu na okno, nebo je-li vynecháno (Windows) všechna okna na popředí platného procesu (Macintosh) okno na popředí platného procesu

Popis

Příkaz MAXIMIZE WINDOW provede zvětšení okna, jehož číslo odkazu předáte do parametru *okno*. Pokud je tento parametr vynechán, efekt je stejný, ale je použit na všechna okna na popředí platného procesu (Windows) nebo okno na popředí platného procesu (Macintosh).

Tento příkaz má stejný efekt jako klepnutí na políčko zvětšení okna aplikace 4D:

Na Windows

Velikost okna je zvětšena na velikost okna aplikace. Maximalizované okno je nastaveno jako okno na popředí. Pokud nepředáte parametr okno, příkaz je použit na všechna okna aplikace.

Políčko zvětšení (maximalizovat) na Windows

Na Mac OS

Velikost okna je zvětšena na velikost obrazovky. Pokud nepředáte parametr *okno*, příkaz bude použit na okno popředí platného procesu.

Políčko zvětšení (maximalizovat) na Macintoshi

Tento příkaz je použit pouze na okna která mají políčko zvětšení. Pokud jej typ okna neobsahuje,příkaz neprovede nic. Jestli chcete vědět více informací, přečtěte si část Typy oken.

MAXIMIZE WINDOW nastaví okno na jeho maximální velikost. Pokud je okno formulář, jehož velikost byla definována ve vlastnostech formuláře, okno je nastaveno na tuto velikost. Pokud je okno již maximalizováno, příkaz neprovede nic.

Příklad

Tento příklad nastaví velikost vašeho okna databáze při jejím otevření na velikost celé obrazovky. K dosažení tohoto je umístěn následující řádek do metody databáze Při spuštění:

```
` Metoda databáze Při spuštění
MAXIMIZE WINDOW
```

Dále si přečtěte

MINIMIZE WINDOW.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

MINIMIZE WINDOW (MINIMALIZOVAT OKNO)

Verze 6.0.5

MINIMIZE WINDOW ({okno})

Parametr	Typ		Popis
okno	číslo	→	číslo odkazu (WinRef) na okno, nebo je-li vynecháno (Windows) všechna okna na popředí platného procesu (Macintosh) okno na popředí platného procesu

Popis

Příkaz MINIMIZE WINDOW nastaví velikost okna, jehož číslo odkazu zadáte do parametru *okno*, na velikost, kterou mělo okno před maximalizováním. Pokud je parametr okno vynechán, příkaz je použit na každé okno (Windows) nebo na okno popředí platného procesu (Macintosh).

Příkaz má stejný efekt jako klepnutí na políčko redukce okna aplikace 4D:

Na Windows

Velikost okna je nastavena na původní velikost, to znamená na velikost před maximalizováním. Pokud je parametr okno vynechán, okna aplikace jsou nastavena na jejich původní velikost.

Políčko obnovit na Windows

Na Mac OS

Velikost okna je nastavena na původní velikost, to znamená na velikost před maximalizováním. Pokud je parametr okno vynechán, okno popředí platného procesu je nastaveno na původní velikost.

Políčko obnovit na Macintoshi

Pokud okno, na které je příkaz použit nebylo maximalizováno (ručně nebo příkazem MAXIMIZE WINDOW), nebo pokud typ okna neobsahuje políčko zvětšení, příkaz neprovede nic. Více informací o typech oken najdete v kapitole Typy oken.

Poznámka: Na Windows si nepleťte tuto funkci s minimalizováním okna, které se provede po klepnutí na zobrazené tlačítko:

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Open form window (Otevřít okno formuláře)

Příkazy a odkazy pro Okna

Verze 6.5

Open form window ({tabulka;} formulářNázev{; typ{; hPoz{; vPoz{; *}}}}) → WinRef

Parametr	Typ		Popis
tabulka	tabulka	→	tabulka, jíž náleží formulář, nebo výchozí tabulka, není-li uvedena
formulářNázev	řetězec	→	název formuláře pro otevření v novém okně
typ	číslo	→	typ otevíraného okna
hPoz	číslo	→	horizontální pozice okna
vPoz	číslo	→	vertikální pozice okna
*	*	→	Uložit platnou pozici a velikost okna
Výsledek funkce	WinRef	←	Číslo odkazu na okno

Popis

Příkaz Open foem window s použitím velikosti a vlastností změny velikosti formuláře *formulářNázev*.

Všimněte si, že *formulářNázev* není v okně zobrazen. Pokud chcete formulář v okně zobrazit, musíte provést jeden z příkazů, který jej načte (například ADD RECORD).

Jako výchozí (pokud parametr typ není předán), je otevřeno standardní okno s políčkem zavření. Narozdíl od příkazu Open window, není k tomuto políčku přiřazena žádná metoda. Při klepnutí na zavírací políčko je okno zrušeno a uzavřeno, pokud není pro formulář aktivována událost Při zavření okna. V tomto případě bude proveden kód přiřazený této události.

Pokud je možné měnit velikost u *formulářNázev*, bude okno obsahovat políčko zvětšení a bude možné měnit jeho velikost ručně.

Poznámka: K získání hlavních vlastností formuláře, použijte příkaz GET FORM PROPERTIES.

Volitelný parametr *typ* vám umožní definovat typ okna. Musíte pro něj použít jednu z následujících konstant:

Konstanta	***Typ***	***Hodnota***
Plain window	*Long Integer*	*8*
Modal dialog box	*Long Integer*	*1*
Movable dialog box	*Long Integer*	*5*
Palette window	*Long Integer*	*720*

Volitelný parametr *hPoz* vám umožní definovat vodorovnou pozici okna. Do tohoto parametru můžete vložit definovanou pozici, vyjádřenou v bodech (podívejte se na příkaz Open window), nebo jednu z předdefinovaných konstant umístěných v části "Window position":

Konstanta	***Typ***	***Hodnota***
Horizontally Centered	*Longint*	*65536*
On the Left	*Longint*	*131072*
On the Right	*Longint*	*196608*

Volitelný parametr *vPoz* vám umožní definovat svislou pozici okna. Do tohoto parametru můžete vložit definovanou pozici, vyjádřenu v bodech (podívejte se na příkaz Open window), nebo jednu z předdefinovaných konstant

umístěných v části "Window position":

Konstanta	*Typ*	*Hodnota*
Vertically Centered	*Longint*	*262144*
At the Top	*Longint*	*327680*
At the Bottom	*Longint*	*393216*

Tyto parametry berou v potaz lištu nástrojů a záhlaví nabídek stejně jako platnou velikost okna aplikace (Windows).

Pokud předáte volitelný parametr *, je při zavření zapamatována platná pozice a velikost okna. Když je okno znovu otevřeno, jsou použity tyto údaje. V tomto případě jsou parametry vPoz a hPoz použity pouze při prvním otevření okna.

Příklady

1. Následující řádek otevře standardní okno se zavíracím políčkem a automaticky jej nastaví na velikost formuláře. Pokud je u formuláře možno měnit velikost, jsou přidána i políčka změny velikosti:

```
$WinRef:=Open form window ([Tabulka1];"Vstup")
```

2. Následující příklad otevře plovoucí paletu a umístí je do levé horní části obrazovky. Tato paleta použije umístění, na které byla paleta přisunuta před jejím minulým zavřením.

```
$winRef := Open form window ([Tabulka1]; "Paleta"; Palette window; On the Left; At the Top;*)
```

Dále si přečtěte

GET FORM PROPERTIES, Open window.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

GET FORM PROPERTIES (VZÍT VLASTNOSTI FORMULÁŘE)

Verze 6.5

GET FORM PROPERTIES ({tabulka;} formulář; šířka; výška{; počStran{; PromŠíř{; PromVýš{; titul}}}})

Parametr	Typ		Popis
tabulka	tabulka	→	Tabulka vlastnící formulář, nebo výchozí tabulka, není-li uvedeno nic
formulář	řetězec	→	název formuláře k získání informací
šířka	Longint	←	šířka formuláře (v bodech)
výška	Longint	←	Výška formuláře (v bodech)
počStran	Longint	←	Počet stránek ve formuláři
PromŠíř	logické	←	True = proměnná šířka, False = konstantní šířka
PromVýš	logické	←	True = proměnná výška, False = konstantní výška
Titul	Text	←	Titul okna formuláře

Popis

Příkaz GET FORM PROPERTIES vrací vlastnosti formuláře *formulář*.

Parametry *šířka* a *výška* vrací šířku a výšku formuláře v bodech. Tyto hodnoty jsou použity z vlastností formuláře Výchozí velikost okna.

- Pokud je velikost formuláře **Automatická**, s použitím definovaných vodorovných a svislých okrajů jsou vypočítány šířka a výška tak, aby byli zobrazeny všechny objekty formuláře.
- Pokud je velikost formuláře **nastavena**, jsou použity ručně zadané hodnoty.
- Pokud je velikost formuláře **Založena na objektech**, je šířka a výška formuláře vypočítána podle pozice objektu.

Parametr *počStran* vrací počet stran ve formuláři, mimo stránky 0 (nula).

Parametry *PromŠíř* a *PromVýš* ukazují, jestli výška a šířka formuláře jsou měnitelné (vrací True) nebo nastavené (vrací False).

Parametr Titul vrací titul formuláře, tak jak je definován. Pokud nebyl definován žádný název, parametr vrátí prázdný řetězec.

Dále si přečtěte

Open form window.

Příkazy a odkazy pro Okna

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Chyby syntaxe

Příkazy a odkazy pro Kódy chyb

Verze 6.0

Následující tabulka ukazuje kódy chyb syntaxe, které mohou nastat při provádění kódu v prostředí uživatele nebo vlastních nabídek. Některé z těchto chyb se mohou objevit pouze v nezkompilované verzi, některé pouze ve zkompilované verzi a některé v obou. Tyto chyby můžete zachytit pomocí metody instalované příkazem ON ERR CALL:

Kód	Popis
1	Byla očekávána "(".
2	Bylo očekáváno pole.
3	Příkaz může být použit pouze na pole v podtabulce.
4	Všechny parametry v seznamu musí být stejného typu.
5	Žádná vybraná tabulka pro provedení příkazu.
6	Příkaz může být použit pouze na pole typu Podtabulka.
7	Byl očekáván číselný argument.
8	Byl očekáván alfanumerický argument.
9	Byl očekáván výsledek podmínkového testu.
10	Příkaz nemůže být použit na tento typ pole.
11	Příkaz nemůže být použit mezi dvěma podmínkovými testy.
12	Příkaz nemůže být použit mezi dvěma číselnými argumenty.
13	Příkaz nemůže být použit mezi dvěma alfanumerickými argumenty.
14	Příkaz nemůže být použit mezi dvěma datumovými argumenty.
15	Operace není slučitelná s dvěma argumenty.
16	Pole nemá žádný vztah.
17	Byla očekávána tabulka.
18	Typy polí jsou neslučitelné.
19	Pole není indexováno.
20	Bylo očekánáno "=".
21	Metoda neexistuje.
22	Pole pro třídění nebo diagram musí patřit ke stejné tabulce nebo podtabulce.
23	Bylo očekáváno "<" nebo ">".
24	Byl očekáván ";".
25	Příliš mnoho polí pro třídění.
26	Typ pole nemůže být Text, obrázek, Blob nebo podtabulka.
27	Před polem musí být název tabulky.
28	Pole musí být číselné.
29	Hodnota musí být 1 nebo 0.
30	Byla očekávána proměnná.
31	Neexistuje záhlaví nabídky s tímto číslem.
32	Bylo očekáváno datum.
33	Neznámý příkaz nebo funkce.
34	
35	Sady jsou z různých tabulek.
36	Neznámý název tabulky.
37	Bylo očekáváno ":=".
38	Toto je funkce a ne procedura.
39	Výběr neexistuje.
40	Toto je procedura, ne funkce.
41	Byla očekávána proměnná nebo pole patřící k podtabulce.

42	Záznam nemůže být vsunut do paměti.
43	Funkce nemůže být nalezena.
44	Metoda nemůže být nalezena.
45	Bylo očekáváno pole nebo proměnná.
46	Byl očekáván číselný nebo alfanumerický argument.
47	Typ pole musí být Alfanumerické.
48	Chyba syntaxe.
49	Tento operátor zde nemůže být použit.
50	Tyto operátory nemohou být použity společně.
51	Modul nebyl přídán.
52	Bylo očekáváno array.
53	Indexy mimo rozsah.
54	Typy argumentů jsou neslučitelné.
55	Byl očekáván logický argument.
56	Bylo očekáváno pole, proměnná nebo tabulka.
57	Byl očekáván operátor.
58	Byla očekávána ")".
59	Tento typ argumentu zde nebyl očekáván.
60	Parametr nebo místní proměnná nemohou být použity v tvrzení EXECUTE ve zkompilované databázi.
61	Typ array nemůže být změněn ve zkompilované databázi.
62	Příkaz nemůže být použit na podtabulka.
63	Pole není indexováno.
64	Bylo očekáváno obrázkové pole nebo proměnná.
65	
66	
67	Tento příkaz nemůže být použit na 4D Server.
68	Byl očekáván seznam.

Tipy

Některé z těchto kódů chyb označují obyčejnou chybu syntaxe překlepnutí. Například chyba #37 nastane pokud spustíte tvrzení v=0, ale chtěli jste spustit v:=0. Této chybě můžete zabránit fixováním vašeho kódu v Editoru metod.

Některé tyto chyby jsou přesné programátorské chyby. Chyba #5 například nastane pokud provedete příkaz ADD RECORD a nedefinujete výchozí tabulku a vynecháte parametr tabulka. V tomto případě není žádná tabulka na kterou by byl příkaz proveden. Této chybě zabráníte pokud si zkontrolujete jestli jste nastavili výchozí tabulku, nebo jestli jste nezapomněli definovat tabulku pro tento výskyt příkazu.

Některé z těchto kódů chyb označují chyby v návrhu. Například chybu #16 dostanete pokud použijete příkaz RELATE ONE na pole, které není vztažené k jinému poli. Tuto chybu opravíte tím, že se podíváte jestli je váš kód skutečně špatný a nebo jesti jste jenom zapoměli vytvořit vztah u tohoto pole.

Některé z chyb, pokud nastanou, neurčují přesně, kde došlo k chybě. Pokud například v podprogramu nastane na řádku vpPol:=Field($1;$2) chyba #53, chyba je špatná číslo tabulky a/nebo pole předáné jako parametr k podprogramu. Proto je chyba umístěna ve volací metodě a ne v metodě kde skutečně nastane. V tomto případě projděte váš kód v okně Ladění a zjistěte na kterém řádku je chyba a opravte jí v Editoru metod.

Dále si přečtěte

ON ERR CALL.

Příkazy a odkazy pro Kódy chyb

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Chyby databázového stroje

Příkazy a odkazy pro Kódy chyb

Verze 6.0

Tato tabulka zobrazuje chyby, které vytváří stroj databáze 4th Dimension. Tyto kódy ukazují chyby vzniklé na nízké úrovni stroje databáze, jako přerušení uživatelem, chyby oprávnění a poškozené objekty.

Kód	Popis
1006	Program přerušen uživatelem_uživatel stisknul Alt-klepnutí (Windows) nebo Option-klepnutí (MacOS)
-9937	Systém hesel je zamčen jiným uživatelem.
-9938	Platný záznam byl změněn z triggeru.
-9939	Externí rutina nebyla nalezena.
-9940	Inicializace 4D Extension byla neúspěšná.
-9941	Neznámý EX_GESTALT volič.
-9942	Schéma licencí 4D Client je nekompatibilní s verzí 4D Server.
-9943	Chyba verze 4D Connectivity Plug-ins.
-9944	Uživatel nepatří ke skupině přístupu 4D Open.
-9945	Chyba CD-ROM 4D Runtime, zapisovocí operace nejsou povoleny.
-9946	Nelze vymazat pojmenovaný výběr, protože neexistuje.
-9947	Bylo označeno políčko "Umožnit pouze připojení 4D Client".
-9948	Modální dialog je aktivován.
-9949	Chyba licence nebo oprávnění.
-9950	Neplatné číslo datové části.
-9951	Toto pole nemá vztah.
-9952	Neplatné záhlaví datové části.
-9953	Neexistuje Log soubor.
-9954	Není žádný platný záznam.
-9955	QuickTime není instalován.
-9956	Verze 4D Client a 4D Server jsou různé.
-9957	Výběrový seznam je zamčený.
-9958	Proces nemůže být zapnut.
-9959	Proces zálohování byl již zapnut jiným uživatelem nebo procesem.
-9960	4D Backup není na serveru instalován.
-9961	Proces zálohování není zapnutý.
-9962	Zálohování nemůže být zapnuto, protože se vypíná server.
-9963	Stanice požaduje neplatné číslo záznamu.
-9964	Stanice poslala špatně definovanou tabulku třídění.
-9965	Stanice poslala špatně definovanou tabulku hledání.
-9966	Stanice požaduje neplatný typ.
-9967	Záznam nemůže být upraven, protože nemůže být načten.
-9968	Stanice vyžaduje neplatné číslo záznamu ve výběru.
-9969	Stanice požaduje neplatný typ pole.
-9970	Pole není indexováno.
-9971	Číslo pole je mimo rozsah vyžadovaný stanicí.
-9972	Číslo souboru je mimo rozsah vyžadovný stanicí.
-9973	TRIC zdroje nejsou stejné.
-9974	Záznam byl již vymazán.
-9975	Strána indexů transakce nemůže být načtena
-9976	Probíhá zálohování, nemůže být provedena změna.

-9977 Výběr neexistuje.
-9978 Špatné heslo uživatele.
-9979 Neznámý uživatel.
-9980 Tabulka nemůže být vytvořena, protože struktura je zamčena.
-9981 Stanice poslala nesprávnou tabulku názvů polí/čísel polí.
-9982 Záznam nebyl načten, protože není ve výběru stanice.
-9983 Stejný externí balík je instalovaný dvakrát
-9984 Transakce byla zrušena kvůli chybě zdvojeného indexu.
-9985 Vrátitelná integrita.
-9986 Záznam zamčen během automatické akce mazání.
-9987 Některé záznamy jsou vztaženy k tomuto záznamu.
-9989 Neplatná struktura (databáze musí být opravena).
-9990 Chyba časování.
-9991 Chyba oprávnění.
-9992 Špatné heslo.
-9993 Záhlaví nabídek je poškozeno (databáze musí být opravena).
-9994 Sériová komunikace přerušena uživatelem. Uživatel stisknul Ctrl-Alt-Shift (Windows) nebo Command-Option-Shift (MacOS)
-9995 Bylo dosaženo limitu demoverze.
-9996 Vyrovnávací paměť je plná (příliš mnoho vracení nebo nahromaděných volání).
-9997 Bylo dosaženo maximálního počtu záznamů.
-9998 Zdvojený klíč.
-9999 Není místo k uložení záznamu (podívejte se na poznámku 4).
-10500 Neplatná adresa záznamu (databáze musí být opravena).
-10501 Neplatná stránka indexu (indexy je třeba opravit).
-10502 Neplatná struktura záznamu (datový soubor musí být opraven).
-10503 Záznam # je mimo rozsah (během GOTO RECORD, nebo musí být datový soubor opraven) (podívejte se na poznámku 3).
-10504 Stránka indexu # je mimo rozsah (indexy je třeba opravit).
-1 Plug-in vyžaduje neznámý vstupní bod.
4001 Plug-in vyžaduje neplatné číslo tabulky.
4002 Plug-in vyžaduje neplatné číslo záznamu.
4003 Plug-in vyžaduje neplatné číslo pole
4004 Přístup k platnému záznamu tabulky z Plug-in ve chvíli, kdy neexistuje platný záznam.

Poznámky

1. Některé z chyb odráží skutečné problémy, např. -10502, jiné chyby se objevují na normálních základech a mohou být zachyceny metodou ON ERR CALL. Například ovládání chyby -9998 Zdvojený klíč bývá důležité pokud vaše aplikace obsahuje možnost vytvoření duplikované hodnoty pro tabulku obsahující indexované pole pro nějž je nastavena vlastnost Jedinečné.

2. Některé z chyb se nikdy neobjeví na úrovni jazyka 4D. Mohou se objevit a být ovládány rutinami stroje databáze nebo při použití 4D Backup nebo 4D Open.

3. Chyba -10503 nemusí vždy znamenat, že databáze potřebuje opravit. Tato chyba může nastat při použití čísla záznamu (např. příkaz GOTO RECORD) na nově vytvořený záznam v transakci. Je to z důvodu, že nově vytvořené záznamy v transakci mají přiřazeno dočasné číslo do doby než je transakce potvrzena. Pokud tato chyba nastane v tomto kontextu, je vaše databáze v pořádku, ale váš kód ne.

4. Chyba -9999 nastane, pokud jsou všechny segmenty vašich dat plné, nebo umístěné na plné jednotce. Tato chyba může být vytvořena pokud je datový soubor zamčený nebo je umístěný na zamčeném disku. Toto vám umožní, například, určit zamčené datové soubory z metody databáze Při spuštění. Podrobnější informace najdete v části Testování jestli je datový soubor zamčený.

Dále si přečtěte

ON ERR CALL.

Příkazy a odkazy pro Kódy chyb

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Chyby síťových komponentů

Verze 6.0

Následující tabulka popisuje chyby které mohou nastat se síťovými komponenty.

Kód	Popis
-10001	Platné připojení k databázi bylo přerušeno.
-10002	Spojení pro tento proces bylo přerušeno.
-10003	Špatné parametry připojení.
-10020	Žádný server nebyl označen pomocí OP Vybrat 4D server.
-10021	Nebyl nalezen žádný server s použitím OP Najít 4D server.
-10030	Během psaní cyklu nastala desynchronizace.
-10031	Během čtení cyklu nastala desynchronizace.
-10033	Nesprávná velikost dat během psaní cyklu.
-10050	Neznámá vlastnost v Get/SetOption.
-10051	Neznámá hodnota v Get/SetOption.
2	Při použití OP Select 4D Server uživatel klepnul na tlačítko Další.

Příkazy a odkazy pro Kódy chyb

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Chyby manažeru souborů OS

Příkazy a odkazy pro Kódy chyb

Verze 6.0

Následující tabulka zobrazuje chyby, které vrací manažer souborů operačního systému. Tyto chyby mohou být vráceny pokud, například, používáte příkazy Systémových dokumentů. V tomto seznamu, slovo "soubor" označuje dokumenty na disku ne soubor v databázi (od verze 6 tabulky).

Kód	Popis
-33	Plná složka souborů. Nemůžete na disku vytvořit nové soubory.
-34	Disk je plný. Není volné místo na disku.
-35	Zadaná jednotka neexistuje.
-36	I/O chyba. Pravděpodobně je špatný blok na disku.
-37	Špatný název souboru nebo disku.
-38	Pokus zapsat nebo číst ze souboru který není otevřen.
-39	Během operace čtení byl dosažen logický konec souboru.
-40	Vyžadovaná pozice je před začátkem souboru.
-41	Nedostatek paměti k otevření nového souboru na disku.
-42	Ve stejnou chvíli otevřeno příliš mnoho souborů.
-43	Soubor nenalezen.
-44	Jednotka je zamčena nastavením hardwaru.
-45	Soubor je zamčen.
-46	Jednotka je zamčena aplikací.
-47	Pokus o přístup k souboru který byl již vymazán.
-48	Pokus o přejmenování souboru s názvem vymazaného souboru.
-49	Pokus o otevření soubory již otevřeného s právy zapisovat.
-51	Pokus o zpřístupnění dokumentu pomocí neplatného čísla odkazu na dokument.
-52	Vnitřní chyba manažera souborů (pozice značky souboru je ztracena).
-53	Jednotka není on-line.
-54	Pokus o otevření zamčeného souboru pro psaní.
-57	Pokus o práci s diskem, který není Macintosh.
-58	Chyba v externím systému souborů.
-60	Chybný blok hlavní složky. Váš disk je poškozen.
-61	Číst/psát povolení neumožňuje psaní.
-64	Hardwarový problém disku (špatná instalace, nesprávné formátování...)
-84	Hardwarový problém disku (špatná instalace, nesprávné formátování...)
-120	Pokus o otevření souboru s použitím cesty definované neexistující složkou.
-121	Cesta přístupu nemůže být vytvořena.
-124	Pokus o zpřístupnění odpojené jednotky.

Dále si přečtěte

ON ERR CALL.

Příkazy a odkazy pro Kódy chyb

4th Dimension 6.5, Seznam témat příkazů

Chyby manažeru paměti OS

Verze 6.0

Následující tabulka ukazuje kódy chyb, které vrací manažer paměti Operačního systému.

Kód	Popis
-108	Nedostatek paměti k provedení operace. K vaší aplikaci 4D přiřaďte více paměti.
-109	Vnitřní problém paměti. Paměť je pravděpodobně nakažena. Skončete tak brzo jak je to možné. Restartujte počítač a znovu otevřete databázi.
-111	Vnitřní problém paměti. Paměť je pravděpodobně nakažena Skončete tak brzo jak je to možné. Restartujte počítač a znovu otevřete databázi. (*)
-117	Vnitřní problém paměti. Paměť je pravděpodobně nakažena Skončete tak brzo jak je to možné. Restartujte počítač a znovu otevřete databázi.

Tip: Při práci s velkými array, BLOBy, obrázky a stejně tak se sadami (objekty které mohou obsahovat velké množství dat), použijte metodu projektu ON ERR CALL k testování chyby -108.

(*) Chyba -111 může také nastat pokud se pokusíte číst BLOBu na posunu který je mimo rozsah. V tomto případě je to malá chyba a vy nemusíte přerušovat svojí práci. Pouze opravte posun který jste vložili do příkazu BLOBu.

Dále si přečtěte

ON ERR CALL.

Příkazy a odkazy pro Kódy chyb

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Chyby manažeru tisku OS

Verze 6.0

Následující tabulka zobrazuje seznam chyb které vrací manažer tisku Operačního systému. Tyto chyby mohou být vráceny během tisku:

Kód	Popis
-1	Problém uložení souboru k tisku
-27	Problém s otevřením nebo ukončením spojení s tiskárnou
-128	Tisk přerušen uživatelem
-193	Zdrojový soubor nebyl nalezen
-4100	Spojení s tiskárnou bylo přerušeno
-4101	Tiskárna je zaneprázdněná nebo není připojena
-8150	Není vybrána laserová tiskárna
-8151	Tiskárna byla nastavena jinou verzí ovladače
-8192	Vypršel čas Laserové tiskárny

Dále si přečtěte

ON ERR CALL.

Příkazy a odkazy pro Kódy chyb

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Chyby manažeru zdrojů OS

Příkazy a odkazy pro Kódy chyb

Verze 6.0

Následující tabulka zobrazje kódy chyb které vrací manažer zdrojů Operačního systému.

Kód	Popis
-1	Zdrojový soubor nemůže být otevřen
-192	Zdroj nebyl nalezen
-193	Mapa zdrojů je poškozena (soubor je potřeba opravit)
-194	Zdroj nemůže být přidán
-196	Zdroj nemůže být vymazán

Dále si přečtěte

ON ERR CALL.

Příkazy a odkazy pro Kódy chyb

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

SANE NaN chyby

Příkazy a odkazy pro Kódy chyb

Verze 6.0

Náesledující tabulka ukazuje NaN kódy vracené Operačním systémem. NaN je znázornění Standard Apple Numeric Environment (SANE) (standardní číselné prostředí Apple), které znamená "Not a Number" (není číslo). NaN se objeví když operace provede žádost která je za rozsahem SANE.

Kód	**Popis**
1	Neplatná mocnina odmocniny
2	Neplatné sčítání
4	Neplatné dělení
8	Neplatné násobení
9	Neplatná památka
17	Konverze neplatného ASCII řetězce
20	Převod čísla typu Comp na plovoucí oddíl
21	Vytváření NaN s nulovým kódem
33	Neplatný argument do trig funkce
34	Neplatný argument do inverze trig funkce
36	Neplatný argument do log funkce
37	Neplatný argument do funkce xi nebu xy
38	Neplatný argument do finanční funkce
255	Neznámá paměť

Příkazy a odkazy pro Kódy chyb

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Chyby manažeru zvuku OS

Verze 6.0

Následující tabulka zobrazuje kódy chyb které vrací manažes zvuku Operačního systému.

Kód	Popis
-203	Příliš mnoho zvukových příkazů
-204	Zvukový zdroj nemůže být načten
-205	Zvukový kanál je poškozen
-206	Formát zvukováho zdroje je špatný
-207	Nedostatek paměti k provedení zvuku
-209	Zvukový kanál je zaneprázdněn

Dále si přečtěte

ON ERR CALL.

Příkazy a odkazy pro Kódy chyb

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Chyby manažeru sériového portu

Verze 6.0

Následující tabulka zobrazuje kódy chyb které vrací manažera sériového portu Operačního systému.

Kód	Popis
-28	Není otevřený žádný sériový port

Dále si přečtěte

ON ERR CALL.

Příkazy a odkazy pro Kódy chyb

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Chyby systému Mac OS

Verze 6.0

Následující tabulka zobrazuje některé kódy chyb které vrací Operační systém Mac OS. Obvykle není možné opravit tyto chyby.

Kód	Popis
4	Dělení nulou
15	Chyba načítání části: 4th Dimension selhala při načítání jedné z částí jejího kódu. Musíte k 4th Dimension přiřadit více paměti
17 až 24	Chybí systémový balík. Vyzkoušejte jestli vaše systémová složka byla správně nainstalovaná
25	Nedostatek paměti Musíte k 4th Dimension přiřadit více paměti
28	Vyrovnávací paměť byla přesunuta do heap paměti aplikace. Musíte k 4th Dimension přiřadit více paměti

Příkazy a odkazy pro Kódy chyb

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Testování jestli je datový soubor zamčený

Příkazy a odkazy pro Kódy chyb

Verze 3.2.5

Testování jestli zamčený datový soubor nebo jednotka na které je umístěn vyžaduje volání do manažera souborů Operačního systému. Pokud to budete dělat pro každou operaci která upravuje data může významě ovlivnit výkon stroje databáze.

Máte možnost testovat tento stav na začátku práce při spuštění databáze. Obvykle to uděláte v metodě databáze Při spuštění. Do 4D v 3.2.5 a 4D Serveru v 1.2.5 testování stavu zamčení vyžaduje externí rutinu která vrací vlastnosti datového souboru a jednotku kde je umístěn. Nyní 4D a 4D Server zjednodušili tento test signalizováním zamčených dat pokaždé když se pokusíte vytvořit nový záznam. Pokud je datový soubor nebo jeho jednotka zamčena, bude generována chyba -9999 Není místo k uložení záznamu.

Následující kód je příklad testu na zamčený datový soubor:

```
    ` Metoda projektu Is data locked
    ` Is data locked → Logické
    ` Is data locked → True znamená zamčený nebo plný
gError:=0
ON ERR CALL("ERROR HANDLING")
CREATE RECORD([Tabulka])
SAVE RECORD([Tabulka])
ON ERR CALL("")
If (gError=0)
   DELETE RECORD([Tabulka])
End if
$0:=(gError=-9999)
```

Metoda ON ERR CALL, nazvaná ERROR HANDLING vypadá následovně:

```
    ` Metoda projektu ERROR HANDLING
gError:=Error
```

Pak do metody databáze Při spuštění můžete napsat následující:

```
    ` Metoda databáze Při spuštění
    ` ...
If (Is data locked)
        ` Zobrazí dialogové okno popisující, že:
        ` - Datový soubor je zamčený,
        ` - nebo jednotka na které je datový soubor je zamčená nebo planá,
        ` Ať je to kterýkoli případ, můžete chtít databázi použít pouze v módu číst:
        ` datový soubor může být umístěn na jednotce CD-ROM.
        ` Dialog může mít tlačítka "Otevřít pouze pro čtení" a "Konec".
   If (bQuit=1)
      QUIT 4D ` Opustit databázi a zkontrolovat co se děje na úrovni systému
   Else
         ` Nastavit meziprocesní příznak k označení, že data jsou zamčená
      <>gREADONLY:=True
         ` ... Pokračovat v metodě Při spuštění
   End if
End if
```

Pokud použijete databázi se zamčeným datovým souborem, ujistěte se, že jste omezili přístup k operacím které mění obsah databáze. K tomu můžete provést vaší testovací funkci znovu nebo založit meziprocesní vlajku jako v předchozím případě. Například pokud máte metodu projektu M_ADD_CUST která zapne proces ve kterém přidáte záznamy, tak víte že pokud proběhne, tak je to možné. Pokud ne, nelze přidávat záznamy. Například:

```
    ` Metoda projektu M_ADD_CUST
If (Can write data)
        ` Go ahead
   <>vlPID_CUST:=New process(...;...;...)
        ` ...
End if
```

Metoda Can write data je zde:

```
    ` Metoda projektu Can write data
    ` Can write data → Logické
    ` Can write data → True, jestli datový soubor není zamčený
$0:=True
If (<>gREADONLY) ` nebo If (Is data locked)
   ALERT("Tato akce nemůže být provedena"+Char(13)+"pokud je datový soubor v módu pouze číst.")
   $0:=False
End if
```

Důležitá poznámka: Aby jste si zachovali výkon databáze ve 4D 3.2.5 a 4D Server 1.2.5 (a dřívější verze), NETESTUJTE datový soubor při každé operaci která může měnit obsah databáze. Pokud například přidáváte záznamy v transakci, nezapomeňte že datový soubor zůstane nedotčený dokud transakci nepotvrdíte. Testování zamčení dat při provádění úprav v transakci může přidat zbytečné testování navíc. Proto je stav zamčení dat testován pouze při vytváření nových dat mimo transakci. To obsahuje přidání a import dat v prostředí uživatele. Týká se to také příkazů ADD RECORD, SAVE RECORD (použité na nový záznam) a ARRAY TO SELECTION (které vytvoří nové záznamy). Netýká se to přidávání nových záznamů v transakci nebo upravování a mazání záznamů již existujících.

Poznámka: Testování zamčenosti dat vás neochrání před potřebou testovat I/O chyby, které mohou nastat při zapisování do datového souboru.

Příkazy a odkazy pro Kódy chyb

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ASCII kódy

Příkazy a odkazy pro ASCII kódy

Verze 6.0

Tabulka ASCII kódů

• Standardní ASCII kódy, od 0 do 127, jsou stejné na Windows a Macintoshi. Tyto kódy jsou zobrazeny v částech ASCII kódy 0..63 a ASCII kódy 64..127.

• ASCII kody od 128 do 255 jsou na Windows a na Macintoshi odlišné. Aby byla zachována nezávislost na platformě, Windows verze 4th Dimension při vkládání znaků do prostředí 4D (vstup dat, Upravit/vložit, Import, atd) automaticky převádí ASCII kódy z Windows na Macintosh ASCII mapu a při výstupu znaků z prostředí 4D (Upravit/vyjmout nebo kopírovat, export, atd.) převádí z Macintosh do Windows ASCII mapy.

ASCII kódy 128 až 255 jsou zobrazeny v částech ASCII kódy 128..191 a ASCII kódy 192..255.

Porozumění ASCII kódům a 4th Dimension

Jak na Macintoshi tak na Windows pracuje vnitřní stroj databáze a jazyk 4D s Macintosh ASCII nastavením. Pokud vkládáte data přes klávesnici (přidávání záznamů, upravování procedur, atd.) 4th Dimension použije vnitřní Altura ASCII konverzi k převodu znaků z klávesnice na Macintosh nastavení. Například pro předání "é" napíšete ALT+0233 a 4th Dimension uloží do záznamu kód 142. Toto je zřejmé pro konečného uživatele, protože když vytvoříte hledání, napíšete hodnoty (do Editoru dotazů) které chcete najít. Proto hodnota kterou předáte (ALT+0233) je přeložena do ASCII kódu 142 a najdete tuto hodnotu.

Kódy pracují stejně i v Editoru metod. Nicméně pro nalezení ASCII kódu, používáte Macintosh ASCII kód znaků.

Například:

```
QUERY(...;[Tabulka]Pole="é") ` é je ALT+0233
```

je stejné jako:

```
QUERY (...;[Tabulka]Pole=Char(142)) ` é je ASCII 142
```

Dále si přečtěte

Ascii, ISO to Mac, Mac to ISO, Mac to Win, ON EVENT CALL, Win to Mac.

Příkazy a odkazy pro ASCII kódy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ASCII kódy 0..63

Příkazy a odkazy pro ASCII kódy

Verze 6.0

Standardní ASCII kódy (0 až 127) jsou stejné jak na Windows tak na Macintoshi.

ASCII kódy 0..63 ASCII kódy 64..127 ASCII kódy 128..191 ASCII kódy 192..255

ASCII		Macintosh nebo Windows	ASCII		Macintosh nebo Windows
Des	Hex	Výsledek	Des	Hex	Výsledek
0	0	NUL	16	10	DLE
1	1	SOH	17	11	DC1
2	2	STX	18	12	DC2
3	3	ETX	19	13	DC3
4	4	EOT	20	14	DC4
5	5	ENQ	21	15	NAK
6	6	ACK	22	16	SYN
7	7	BEL	23	17	ETB
8	8	BS	24	18	CAN
9	9	HT	25	19	EM
10	A	LF	26	1A	SUB
11	B	VT	27	1B	ESC
12	C	FF	28	1C	FS
13	D	CR	29	1D	GS
14	E	SO	30	1E	RS
15	F	SI	31	1F	US

ASCII		Macintosh nebo Windows	ASCII		Macintosh nebo Windows
Des	Hex	Výsledek	Des	Hex	Výsledek
32	20	mez	48	30	0
33	21	!	49	31	1
34	22	"	50	32	2
35	F	#	51	33	3
36	10	$	52	34	4
37	11	%	53	35	5
38	12	&	54	36	6
39	13	'	55	37	7
40	28	(	56	38	8
41	29	)	57	39	9
42	2A	*	58	3A	:
43	2B	+	59	3B	;
44	2C	,	60	3C	<
45	2D	-	61	3D	=
46	2E	.	62	3E	>
47	2F	/	63	3F	?

Příkazy a odkazy pro ASCII kódy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ASCII kódy 64..127

Příkazy a odkazy pro ASCII kódy

Verze 6.0

Standardní ASCII kódy (0 až 127) jsou stejné jak na Windows tak na Macintoshi.

ASCII		Macintosh nebo Windows
Des	Hex	Výsledek
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O

ASCII		Macintosh nebo Windows
Des	Hex	Výsledek
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

ASCII		Macintosh nebo Windows
Des	Hex	Výsledek
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o

ASCII		Macintosh nebo Windows
Des	Hex	Výsledek
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	DEL

Příkazy a odkazy pro ASCII kódy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ASCII kódy 128..191

Příkazy a odkazy pro ASCII kódy

Verze 6.0

Následující tabulka zobrazuje znaky zobrazené 4th Dimension pro ASCII kódy na Windows a na Macintosh. Dále tabulka obsahuje kombinace kláves doporučené pro vytvoření znaku na české klávesnici.

Poznámka pro Macintosh: Pokud je před zkratkou US, pak je tato zkratka pro US klávesnici.

ASCII		Macintosh		Windows	
Des	Hex	Výsledek	Kombinace kláves	Výsledek	Kombinace kláves
128	80	Ä	Option-¨, Shift-a	Ä	Alt+0196
129	81	Ą	Option+Shift+˙, Shift+a	€	Alt+0128
130	82	ą	Option+Shift+˙, a		
131	83	É	É	É	Alt+0201
132	84	Ą	Shift+Option+a	Ą	Alt+0165
133	85	Ö	Option-¨, Shift-o	Ö	Alt+0214
134	86	Ü	Option-¨, Shift-u	Ü	Alt+0220
135	87	á	á	á	Alt+0225
136	88	ą	Option-a	ą	Alt+0185
137	89	Č	Č	Č	Alt+0200
138	8A	ä	Option-¨, a	ä	Alt+0228
139	8B	č	č	č	Alt+0232
140	8C	Ć	´, Shift+c	Ć	Alt+0198
141	8D	ć	´, c	ć	Alt+0230
142	8E	é	é	é	Alt+0233
143	8F	Ź	´, Shift+z	Ź	Alt+0143

ASCII		Macintosh		Windows	
Des	Hex	Výsledek	Kombinace kláves	Výsledek	Kombinace kláves
144	90	ź	´, z	ź	Alt+0159
145	91	Ď	Ď	Ď	Alt+0207
146	92	í	í	í	Alt+0237
147	93	ď	ď	ď	Alt+0239
148	94	Ē	Option+Shift+˙, Shift+e		
149	95	ē	Option+Shift+˙, e	‡	Alt+0135
150	96	Ê	Option+Shift+w	^	Alt+0136
151	97	ó	ó	ó	Alt+0243
152	98	ė	Option+w	‰	Alt+0137
153	99	ô	Option+^, o	ô	Alt+0244
154	9A	ö	Option+¨, o	ö	Alt+0246
155	9B	õ	Option+Shift+§, o		
156	9C	ú	ú	ú	Alt+0250
157	9D	Ě	Ě	Ě	Alt+0204
158	9E	ě	ě	ě	Alt+0236
159	9F	ü	Option+¨, u	ü	Alt+0252

ASCII		Macintosh		Windows	
Des	Hex	Výsledek	Kombinace kláves	Výsledek	Kombinace kláves
160	90	†	Option+Shift+ž	†	Alt+0134
161	91	°	Option+=	°	Alt+0176
162	92	Ę	Option+Shift+e	Ę	Alt+0202
163	93	£	Option+Shift+č	ˇ	Alt+0152
164	94	§	§	§	Alt+0167
165	95	•	Option+Shift+ě	•	Alt+0149
166	96	¶	Option+Shift+ý	¶	Alt+0182
167	97	ß	Option+s	ß	Alt+0223
168	98	®	Option+r	®	Alt+0174
169	99	©	Option+Shift+c	©	Alt+0169
170	9A	™	Option+Shift+t	™	Alt+0153
171	9B	ę	Option+e	ę	Alt+0234
172	9C	¨	Option+¨, Mez	¨	Alt+0168
173	9D	≠	Option+Shift+š	ˇ	Alt+0161
174	9E	ģ	´, g	Ī	Alt+0238
175	9F	Į	Option+Shift+=, Shift+i	¤	Alt+0164

ASCII		Macintosh		Windows	
Des	Hex	Výsledek	Kombinace kláves	Výsledek	Kombinace kláves
176	90	į	Option+Shift+=, i	¦	Alt+0166
177	91	Ī	Option+Shift+´, Shift+i	Ş	Alt+0170
178	92	≤	Option+<	-	Alt+173
179	93	≥	Option+Shift+>	±	Alt+0177
180	94	ī	Option+Shift+´, i	¸	Alt+0178
181	95	ĸ	Option+Shift+=, Shift+k	´	Alt+0180
182	96	∂	Option+d	µ	Alt+0181
183	97	Σ	Option+Shift+s	·	Alt+0183
184	98	ł	Option+l	ł	Alt+0179
185	99	Ļ	Option+Shift+=, Shift+l	¸	Alt+0184
186	9A	ļ	Option+Shift+=, l	ş	Alt+0186
187	9B	Ľ	Shift+´, Shift+l	Ľ	Alt+0188
188	9C	ľ	Shift+´, l	ľ	Alt+0190
189	9D	Ĺ	´, Shift+l	Ĺ	Alt+0197
190	9E	ĺ	´, l	ĺ	Alt+0229
191	9F	Ņ	Option+Shift+=, Shift+n	Â	Alt+0194

Poznámka: Buňky v části pro Windows které jsou šedé, označují znaky které nejsou na Windows dostupné nebo jsou odlišné od Macintosh znaků.

Dále si přečtěte

Char, ISO to Mac, Mac to ISO, Mac to Win, ON EVENT CALL, Win to Mac.

Příkazy a odkazy pro ASCII kódy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

ASCII kódy 192..255

Příkazy a odkazy pro ASCII kódy

Verze 6.0

Následující tabulka zobrazuje znaky zobrazené 4th Dimension pro ASCII kódy na Windows a na Macintosh. Dále tabulka obsahuje kombinace kláves doporučené pro vytvoření znaku na US klávesnici.

Poznámka pro Macintosh: Pokud je před zkratkou US, pak je tato zkratka pro US klávesnici.

ASCII		Macintosh		Windows	
Des	Hex	Výsledek	Kombinace kláves	Výsledek	Kombinace kláves
192	90	ą	Option+Shift+=, n	Ą	Alt+0195
193	91	Ń	´, Shift+n	Ń	Alt+0209
194	92	¬	Option+Shift++	¬	Alt+0172
195	93	√	Option+Shift+v	¯	Alt+0189
196	94	ń	´, n	ń	Alt+0241
197	95	Ñ	Shift+´, Shift+n	Ñ	Alt+0210
198	96	∆	Option+Shift+d	Ç	Alt+0199
199	97	«	Option+Shift+í	«	Alt+0171
200	98	»	Option+Shift+é	»	Alt+0187
201	99	…	Option+Shift+ů	…	Alt+0133
202	9A	Mez	Mezerník	Mez	Alt+0160
203	9B	ň	ň	ń	Alt+0242
204	9C	Ő	Option+Shift+¨, Shift+o	Õ	Alt+0213
205	9D	Ő	Option+Shift+ů, Shift+o	Ě	Alt+0203
206	9E	ő	Option+Shift+¨, o	ő	Alt+0245
207	9F	Õ	Option+Shift+´, Shift+o	Ì	Alt+0206

ASCII		Macintosh		Windows	
Des	Hex	Výsledek	Kombinace kláves	Výsledek	Kombinace kláves
208	90	–	Option+-	–	Alt+0150
209	91	—	Option+Shift+-	—	Alt+0151
210	92	"	Option+Shift+h	"	Alt+0147
211	93	"	Option+Shift+j	"	Alt+0148
212	94	'	Option+h	'	Alt+0145
213	95	'	Option+j	'	Alt+0146
214	96	÷	Option+Shift+á	÷	Alt+0247
215	97	◊	Option+Shift+ř	Đ	Alt+0208
216	98	ő	Option+Shift+´, o	×	Alt+0215
217	99	Ŕ	´, Shift+r	Ŕ	Alt+0192
218	9A	ŕ	´, r	ŕ	Alt+0224
219	9B	Ř	Ř	Ŕ	Alt+0216
220	9C	‹	US Option+Shift+3	‹	Alt+0139
221	9D	›	US Option+Shift+4	›	Alt+0155
222	9E	ř	ř	ŗ	Alt+0248
223	9F	Ŗ	Option+Shift+=, Shift+r	ŧ	Alt+0254

ASCII		Macintosh		Windows	
Des	Hex	Výsledek	Kombinace kláves	Výsledek	Kombinace kláves
224	90	ŗ	Option+Shift+=, r	Ţ	Alt+0222
225	91	Š	Š	Š	Alt+0138
226	92	‚	Option+n		
227	93	„	Option+Shift+n		
228	94	š	š	š	Alt+0154
229	95	Ś	´, Shift+s	Ś	Alt+0140
230	96	ś	´, s	ś	Alt+0156
231	97	Á	Á	Á	Alt+0193
232	98	Ť	Ť	Ť	Alt+0141
233	99	ť	ť	ť	Alt+0157
234	9A	Í	Í	Í	Alt+0205
235	9B	Ž	Ž	Ž	Alt+0142
236	9C	ž	ž	ž	Alt+0158
237	9D	Ű	Option+Shift+´, Shift+u	ă	Alt+0226
238	9E	Ó	Ó	Ó	Alt+0211
239	9F	Ô	Option+´, Shift+o	Ô	Alt+0212

ASCII		Macintosh		Windows	
Des	Hex	Výsledek	Kombinace kláves	Výsledek	Kombinace kláves
240	90	ű	Option+Shift+´, u	ă	Alt+0227
241	91	Ů	Ů	Ů	Alt+0217
242	92	Ú	Ú	Ú	Alt+0218
243	93	ů	ů	ů	Alt+0249
244	94	Ü	Option+Shift+¨, Shift+u	Ü	Alt+0219
245	95	ü	Option+Shift+¨, u	ü	Alt+0251
246	96	Ų	Option+Shift+=, Shift+u	ç	Alt+0231
247	97	ų	Option+Shift+=, u	ë	Alt+0235
248	98	Ý	Ý	Ý	Alt+0221
249	99	ý	ý	ý	Alt+0253
250	9A	ķ	Option+Shift+=, k	đ	Alt+0240
251	9B	Ż	Option+Shift+z	Ż	Alt+0175
252	9C	Ł	Option+Shift+l	Ł	Alt+0163
253	9D	ż	Option+z	ż	Alt+0191
254	9E	Ģ	Option+Shift+=, Shift+g	˙	Alt+0255
255	9F	ˇ	Shift+´, Mezerník	˙	Alt+0162

Poznámka: Buňky v části pro Windows které jsou šedé, označují znaky které nejsou na Windows dostupné nebo jsou odlišné od Macintosh znaků.

Dále si přečtěte

Ascii, ISO to Mac, Mac to ISO, Mac to Win, ON EVENT CALL, Win to Mac.

Příkazy a odkazy pro ASCII kódy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

Kódy kláves funkcí

Příkazy a odkazy pro ASCII kódy

Verze 6.0

Hodnoty kláves funkcí vrací 4th Dimension do systémové proměnné KeyCode, která je použita v metodě projektu instalované příkazem ON EVENT CALL. Tyto metody projektu jsou použity k zachytávání událostí. Hodnoty kláves funkcí nejsou založeny na ASCII kódech. Jsou to:

Klávesa funkce	Kód
F1	-122
F2	-120
F3	-99
F4	-118
F5	-96
F6	-97
F7	-98
F8	-100
F9	-101
F10	-109
F11	-103
F12	-111
F13	-105
F14	-107
F15	-113

Připomínka: Systémová proměnná KeyCode je určena pro použití v metodě instalované příkazem ON EVENT CALL.

Následující tabulka ukazuje hodnoty, které bude obsahovat KeyCode po stisknutí jedné z důležitých kláves jako Return nebo Enter.

Kód	Klávesa
3	Enter
13	Return
8	Backspace nebo Delete
9	Tabelátor
27	Escape nebo Clear
127	Del
5	Hepl
1	Home
4	End
11	Page up
12	Page down
28	Šipka vlevo
29	Šipka vpravo
30	Šipka nahoru
31	Šipka dolu

Příkazy a odkazy pro ASCII kódy

4th Dimension 6.5, Seznam témat příkazů

4th Dimension 6.5, Abecední seznam příkazů

4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů

Konstanty pro 4D prostředí

Konstanta	*Typ*	*Hodnota*
4D Client	*Long Integer*	*4*
4D Engine	*Long Integer*	*1*
4D First	*Long Integer*	*6*
4D Runtime	*Long Integer*	*2*
4D Runtime Classic	*Long Integer*	*3*
4D Server	*Long Integer*	*5*
4th Dimension	*Long Integer*	*0*
Demo Version	*Long Integer*	*1*
Full Version	*Long Integer*	*0*

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro ASCII kódy

Konstanta	*Typ*	*Hodnota*
ACK ASCII code	*Long Integer*	*6*
At sign	*Long Integer*	*64*
Backspace	*Long Integer*	*8*
BEL ASCII code	*Long Integer*	*7*
BS ASCII code	*Long Integer*	*8*
CAN ASCII code	*Long Integer*	*24*
Carriage return	*Long Integer*	*13*
CR ASCII code	*Long Integer*	*13*
DC1 ASCII code	*Long Integer*	*17*
DC2 ASCII code	*Long Integer*	*18*
DC3 ASCII code	*Long Integer*	*19*
DC4 ASCII code	*Long Integer*	*20*
DEL ASCII code	*Long Integer*	*127*
DLE ASCII code	*Long Integer*	*16*
Double quote	*Long Integer*	*34*
EM ASCII code	*Long Integer*	*25*
ENQ ASCII code	*Long Integer*	*5*
Enter	*Long Integer*	*3*
EOT ASCII code	*Long Integer*	*4*
ESC ASCII code	*Long Integer*	*27*
Escape	*Long Integer*	*27*
ETB ASCII code	*Long Integer*	*23*
ETX ASCII code	*Long Integer*	*3*
FF ASCII code	*Long Integer*	*12*
FS ASCII code	*Long Integer*	*28*
GS ASCII code	*Long Integer*	*29*
HT ASCII code	*Long Integer*	*9*
LF ASCII code	*Long Integer*	*10*
Line feed	*Long Integer*	*10*
NAK ASCII code	*Long Integer*	*21*
NBSP	*Long Integer*	*202*
NUL ASCII code	*Long Integer*	*0*
Period	*Long Integer*	*46*
Quote	*Long Integer*	*39*
RS ASCII code	*Long Integer*	*30*
SI ASCII code	*Long Integer*	*15*
SO ASCII code	*Long Integer*	*14*
SOH ASCII code	*Long Integer*	*1*
SP ASCII code	*Long Integer*	*32*
Space	*Long Integer*	*32*
STX ASCII code	*Long Integer*	*2*
SUB ASCII code	*Long Integer*	*26*
SYN ASCII code	*Long Integer*	*22*
Tab	*Long Integer*	*9*
US ASCII code	*Long Integer*	*31*
VT ASCII code	*Long Integer*	*11*

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro BLOB

Konstanta	*Typ*	*Hodnota*
C string	*Long Integer*	*0*
Compact compression mode	*Long Integer*	*1*
Extended real format	*Long Integer*	*1*
Fast compression mode	*Long Integer*	*2*
Is not compressed	*Long Integer*	*0*
Macintosh byte ordering	*Long Integer*	*1*
Macintosh double real format	*Long Integer*	*2*
Native byte ordering	*Long Integer*	*0*
Native real format	*Long Integer*	*0*
Pascal string	*Long Integer*	*1*
PC byte ordering	*Long Integer*	*2*
PC double real format	*Long Integer*	*3*
Text with length	*Long Integer*	*2*
Text without length	*Long Integer*	*3*

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro schránku

Konstanta	*Typ*	*Hodnota*
No such data in clipboard	*Long Integer*	*-102*
Picture data	*String*	*PICT*
Text data	*String*	*TEXT*

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro barvy

Konstanta	*Typ*	*Hodnota*
Black	Long Integer	15
Blue	Long Integer	6
Brown	Long Integer	13
Dark Blue	Long Integer	5
Dark Brown	Long Integer	10
Dark Green	Long Integer	9
Dark Grey	Long Integer	11
Green	Long Integer	8
Grey	Long Integer	14
Light Blue	Long Integer	7
Light Grey	Long Integer	12
Orange	Long Integer	2
Purple	Long Integer	4
Red	Long Integer	3
White	Long Integer	0
Yellow	Long Integer	1

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro komunikaci

Konstanta	*Typ*	*Hodnota*
Data bits 5	Long Integer	0
Data bits 6	Long Integer	2048
Data bits 7	Long Integer	1024
Data bits 8	Long Integer	3072
MacOS Printer Port	Long Integer	0
MacOS Serial Port	Long Integer	1
Parity Even	Long Integer	12288
Parity None	Long Integer	0
Parity Odd	Long Integer	4096
Protocol DTR	Long Integer	30
Protocol None	Long Integer	0
Protocol XONXOFF	Long Integer	20
Speed 115200	Long Integer	1022
Speed 1200	Long Integer	94
Speed 1800	Long Integer	62
Speed 19200	Long Integer	4
Speed 230400	Long Integer	1021
Speed 2400	Long Integer	46
Speed 300	Long Integer	380
Speed 3600	Long Integer	30
Speed 4800	Long Integer	22
Speed 57600	Long Integer	0
Speed 600	Long Integer	189
Speed 7200	Long Integer	14
Speed 9600	Long Integer	10
Stop bits One	Long Integer	16384
Stop bits One and a half	Long Integer	-32768
Stop bits Two	Long Integer	-16384

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro databázový motor

Konstanta	*Typ*	*Hodnota*
New record	Long Integer	-3
No current record	Long Integer	-1

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro události databáze

Konstanta	*Typ*	*Hodnota*
On Deleting Record Event	Long Integer	3
On Loading Record Event	Long Integer	4
On Saving Existing Record Event	Long Integer	2
On Saving New Record Event	Long Integer	1

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro parametry databáze

Konstanta	*Typ*	*Hodnota*
Database Cache Size	Long Integer	9
Index Compacting	Long Integer	4
Maximum Web Process	Long Integer	7
Minimum Web Process	Long Integer	6
Seq Access Optimization	Long Integer	2
Seq Distinct Values Ratio	Long Integer	3
Seq Order Ratio	Long Integer	1
Seq Query Select Ratio	Long Integer	5
Web Conversion Mode	Long Integer	8

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro formáty zobrazení datumů

Konstanta	*Typ*	*Hodnota*
Abbr Month Day	Long Integer	6
Abbreviated	Long Integer	2
Long	Long Integer	3
MM DD YYYY	Long Integer	4
MM DD YYYY Forced	Long Integer	7
Month Day Year	Long Integer	5
Short	Long Integer	1

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro dny a měsíce

Konstanta	*Typ*	*Hodnota*
April	Long Integer	4
August	Long Integer	8
December	Long Integer	12
February	Long Integer	2
Friday	Long Integer	6
January	Long Integer	1
July	Long Integer	7
June	Long Integer	6
March	Long Integer	3
May	Long Integer	5
Monday	Long Integer	2
November	Long Integer	11
October	Long Integer	10
Saturday	Long Integer	7
September	Long Integer	9
Sunday	Long Integer	1
Thursday	Long Integer	5
Tuesday	Long Integer	3
Wednesday	Long Integer	4

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro Euro měny

Konstanta	*Typ*	*Hodnota*
Austrian Schilling	String	ATS
Belgian Franc	String	BEF
Deutschemark	String	DEM
Euro	String	EUR
Finnish Markka	String	FIM
French Franc	String	FRF
Irish Pound	String	IEP
Italian Lira	String	ITL
Luxembourg Franc	String	LUF
Netherlands Guilder	String	NLG
Portuguese Escudo	String	PTE
Spanish Peseta	String	ESP

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro události (modifikátory)

Konstanta	*Typ*	*Hodnota*
Activate window bit	Long Integer	0
Activate window mask	Long Integer	1
Caps Lock key bit	Long Integer	10
Caps Lock key mask	Long Integer	1024
Command key bit	Long Integer	8
Command key mask	Long Integer	256
Control key bit	Long Integer	12
Control key mask	Long Integer	4096
Mouse button bit	Long Integer	7
Mouse button mask	Long Integer	128
Option key bit	Long Integer	11
Option key mask	Long Integer	2048
Right control key bit	Long Integer	15
Right control key mask	Long Integer	32768
Right option key bit	Long Integer	14
Right option key mask	Long Integer	16384
Right shift key bit	Long Integer	13
Right shift key mask	Long Integer	8192
Shift key bit	Long Integer	9
Shift key mask	Long Integer	512

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro události (co)

Konstanta	*Typ*	*Hodnota*
Activate event	Long Integer	8
Auto key event	Long Integer	5
Disk event	Long Integer	7
Key down event	Long Integer	3
Key up event	Long Integer	4
Mouse down event	Long Integer	1
Mouse up event	Long Integer	2
Null event	Long Integer	0
Operating system event	Long Integer	15
Update event	Long Integer	6

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro výrazy

Konstanta	*Typ*	*Hodnota*
MAXINT	Long Integer	32767
MAXLONG	Long Integer	2147483647
MAXTEXTLEN	Long Integer	32000

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro typy polí a proměnných

Konstanta	*Typ*	*Hodnota*
Array 2D	Long Integer	13
Boolean array	Long Integer	22
Date array	Long Integer	17
Integer array	Long Integer	15
Is Alpha Field	Long Integer	0
Is BLOB	Long Integer	30
Is Boolean	Long Integer	6
Is Date	Long Integer	4
Is Integer	Long Integer	8
Is LongInt	Long Integer	9
Is Picture	Long Integer	3
Is Pointer	Long Integer	23
Is Real	Long Integer	1
Is String Var	Long Integer	24
Is Subtable	Long Integer	7
Is Text	Long Integer	2
Is Time	Long Integer	11
Is Undefined	Long Integer	5
LongInt array	Long Integer	16
Picture array	Long Integer	19
Pointer array	Long Integer	20
Real array	Long Integer	14
String array	Long Integer	21
Text array	Long Integer	18

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro stav v okně

Konstanta	*Typ*	*Hodnota*
In contents	Long Integer	3
In drag	Long Integer	4
In go away	Long Integer	6
In grow	Long Integer	5
In menu bar	Long Integer	1
In system window	Long Integer	2
In zoom box	Long Integer	8

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro styly písem

Konstanta	*Typ*	*Hodnota*
Bold	Long Integer	1
Condensed	Long Integer	32
Extended	Long Integer	64
Italic	Long Integer	2
Outline	Long Integer	8
Plain	Long Integer	0
Shadow	Long Integer	16
Underline	Long Integer	4

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro události formuláře

Konstanta	*Typ*	*Hodnota*
On Activate	Long Integer	11
On After Keystroke	Long Integer	28
On Before Keystroke	Long Integer	17
On Clicked	Long Integer	4
On Close Box	Long Integer	22
On Close Detail	Long Integer	26
On Data Change	Long Integer	20
On Deactivate	Long Integer	12
On Display Detail	Long Integer	8
On Double Clicked	Long Integer	13
On Drag Over	Long Integer	21
On Drop	Long Integer	16
On Getting Focus	Long Integer	15
On Header	Long Integer	5
On Load	Long Integer	1
On Losing Focus	Long Integer	14
On Menu Selected	Long Integer	18
On Open Detail	Long Integer	25
On Outside Call	Long Integer	10
On Plug in Area	Long Integer	19
On Printing Break	Long Integer	6
On Printing Detail	Long Integer	23
On Printing Footer	Long Integer	7
On Resize	Long Integer	29
On Timer	Long Integer	27
On Unload	Long Integer	24
On Validate	Long Integer	3

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro klávesy funkcí

Konstanta	*Typ*	*Hodnota*
Backspace Key	Long Integer	8
Down Arrow Key	Long Integer	31
End Key	Long Integer	4
Enter Key	Long Integer	3
Escape Key	Long Integer	27
F1 Key	Long Integer	-122
F10 Key	Long Integer	-109
F11 Key	Long Integer	-103
F12 Key	Long Integer	-111
F13 Key	Long Integer	-105
F14 Key	Long Integer	-107
F15 Key	Long Integer	-113
F2 Key	Long Integer	-120
F3 Key	Long Integer	-99
F4 Key	Long Integer	-118
F5 Key	Long Integer	-96
F6 Key	Long Integer	-97
F7 Key	Long Integer	-98
F8 Key	Long Integer	-100
F9 Key	Long Integer	-101
Help Key	Long Integer	5
Home Key	Long Integer	1
Left Arrow Key	Long Integer	28
Page Down Key	Long Integer	12
Page Up Key	Long Integer	11
Return Key	Long Integer	13
Right Arrow Key	Long Integer	29
Tab Key	Long Integer	9
Up Arrow Key	Long Integer	30

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro hierarchické seznamy

Konstanta	*Typ*	*Hodnota*
Ala Macintosh	Long Integer	1
Ala Windows	Long Integer	2
Macintosh node	Long Integer	860
Use PicRef	Long Integer	131072
Use PICT resource	Long Integer	65536
Windows node	Long Integer	138

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro ISO Latin1

Konstanta	*Typ*	*Hodnota*
ISO L1 a acute	String	á
ISO L1 a circumflex	String	â
ISO L1 a grave	String	à
ISO L1 a ring	String	å
ISO L1 a tilde	String	ã
ISO L1 a umlaut	String	ä
ISO L1 ae ligature	String	æ
ISO L1 Ampersand	String	&
ISO L1 c cedilla	String	ç
ISO L1 Cap A acute	String	Á
ISO L1 Cap A circumflex	String	Â
ISO L1 Cap A grave	String	À
ISO L1 Cap A ring	String	Å
ISO L1 Cap A tilde	String	Ã
ISO L1 Cap A umlaut	String	Ä
ISO L1 Cap AE ligature	String	&AELig;
ISO L1 Cap C cedilla	String	Ç
ISO L1 Cap E acute	String	É
ISO L1 Cap E circumflex	String	Ê
ISO L1 Cap E grave	String	È
ISO L1 Cap E umlaut	String	Ë
ISO L1 Cap Eth Icelandic	String	Ð
ISO L1 Cap I acute	String	Í
ISO L1 Cap I circumflex	String	Î
ISO L1 Cap I grave	String	Ì
ISO L1 Cap I umlaut	String	Ï
ISO L1 Cap N tilde	String	Ñ
ISO L1 Cap O acute	String	Ó
ISO L1 Cap O circumflex	String	Ô
ISO L1 Cap O grave	String	Ò
ISO L1 Cap O slash	String	Ø
ISO L1 Cap O tilde	String	Õ
ISO L1 Cap O umlaut	String	Ö
ISO L1 Cap THORN Icelandic	String	Þ
ISO L1 Cap U acute	String	Ú
ISO L1 Cap U circumflex	String	Û
ISO L1 Cap U grave	String	Ù
ISO L1 Cap U umlaut	String	Ü
ISO L1 Cap Y acute	String	Ý
ISO L1 Copyright	String	©
ISO L1 e acute	String	é
ISO L1 e circumflex	String	ê
ISO L1 e grave	String	è
ISO L1 e umlaut	String	ë
ISO L1 eth Icelandic	String	ð
ISO L1 Greater than	String	>
ISO L1 i acute	String	í
ISO L1 i circumflex	String	î
ISO L1 i grave	String	ì
ISO L1 i umlaut	String	ï
ISO L1 Less than	String	<
ISO L1 n tilde	String	ñ

ISO L1 o acute	String	&oacute;
ISO L1 o circumflex	String	&ocirc;
ISO L1 o grave	String	&ograve;
ISO L1 o slash	String	&oslash;
ISO L1 o tilde	String	&otilde;
ISO L1 o umlaut	String	&ouml;
ISO L1 Quotation mark	String	&quot;
ISO L1 Registered	String	&reg;
ISO L1 sharp s German	String	&szlig;
ISO L1 thorn Icelandic	String	&thorn;
ISO L1 u acute	String	&uacute;
ISO L1 u circumflex	String	&ucirc;
ISO L1 u grave	String	&ugrave;
ISO L1 u umlaut	String	&uuml;
ISO L1 y acute	String	&yacute;
ISO L1 y umlaut	String	&yuml;

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro matematiku

Konstanta	*Typ*	*Hodnota*
Degree	Real	0.0174532925199432958
e number	Real	2.71828182845904524
Pi	Real	3.14159265358979323
Radian	Real	57.29577951308232088

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro otevření okna

Konstanta	*Typ*	*Hodnota*
Alternate dialog box	Long Integer	3
Has grow box	Long Integer	4
Has highlight	Long Integer	1
Has window title	Long Integer	2
Has zoom box	Long Integer	8
Modal dialog box	Long Integer	1
Movable dialog box	Long Integer	5
Palette window	Long Integer	720
Plain dialog box	Long Integer	2
Plain fixed size window	Long Integer	4
Plain no zoom box window	Long Integer	0
Plain window	Long Integer	8
Round corner window	Long Integer	16

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro formát zobrazení obrázku

Konstanta	*Typ*	*Hodnota*
On Background	Long Integer	3
Scaled to Fit	Long Integer	2
Scaled to fit prop centered	Long Integer	6
Scaled to fit proportional	Long Integer	5
Truncated Centered	Long Integer	1
Truncated non Centered	Long Integer	4

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro rozhraní platformy

Konstanta	*Typ*	*Hodnota*
Automatic interface	Long Integer	-1
Macintosh interface	Long Integer	0
Platinum interface	Long Integer	3
Windows 95 interface	Long Integer	2
Windows NT 3.51 interface	Long Integer	1

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro vlastnosti platformy

Konstanta	*Typ*	*Hodnota*
INTEL 386	Long Integer	386
INTEL 486	Long Integer	486
Macintosh 68K	Long Integer	1
Pentium	Long Integer	586
Power Macintosh	Long Integer	2
PowerPC 601	Long Integer	601
PowerPC 603	Long Integer	603
PowerPC 604	Long Integer	604
PowerPC G3	Long Integer	510
Windows	Long Integer	3

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro stav procesu

Konstanta	*Typ*	*Hodnota*
Aborted	Long Integer	-1
Delayed	Long Integer	1
Does not exist	Long Integer	-100
Executing	Long Integer	0
Hidden modal dialog	Long Integer	6
Paused	Long Integer	5
Waiting for input output	Long Integer	3
Waiting for internal flag	Long Integer	4
Waiting for user event	Long Integer	2

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro typ procesu

Konstanta	*Typ*	*Hodnota*
Apple Event Manager	Long Integer	-7
Cache Manager	Long Integer	-4
Created from Menu Command	Long Integer	2
Created from Programming	Long Integer	1
Created from User Mode	Long Integer	3
Design Process	Long Integer	-2
Event Manager	Long Integer	-8
External Task	Long Integer	-9
Indexing Process	Long Integer	-5
None	Long Integer	0
Other 4D Process	Long Integer	-10
Other User Process	Long Integer	4
Serial Port Manager	Long Integer	-6
User or Custom Menus Process	Long Integer	-1
Web Process with Context	Long Integer	-11
Web Process with no Context	Long Integer	-3

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro cílení dotazu

Konstanta	*Typ*	*Hodnota*
Into current selection	Long Integer	0
Into named selection	Long Integer	2
Into set	Long Integer	1
Into variable	Long Integer	3

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro vlastnosti zdroje

Konstanta	*Typ*	*Hodnota*
Changed resource bit	Long Integer	1
Changed resource mask	Long Integer	2
Locked resource bit	Long Integer	4
Locked resource mask	Long Integer	16
Preloaded resource bit	Long Integer	2
Preloaded resource mask	Long Integer	4
Protected resource bit	Long Integer	3
Protected resource mask	Long Integer	8
Purgeable resource bit	Long Integer	5
Purgeable resource mask	Long Integer	32
System heap resource bit	Long Integer	6
System heap resource mask	Long Integer	64

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro hloubku obrazovky

Konstanta	*Typ*	*Hodnota*
Black and white	Long Integer	0
Four colors	Long Integer	2
Is color	Long Integer	1
Is gray scale	Long Integer	0
Millions of colors 24 bit	Long Integer	24
Millions of colors 32 bit	Long Integer	32
Sixteen colors	Long Integer	4
Thousands of colors	Long Integer	16
Two fifty six colors	Long Integer	8

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro nastavení RGB barvy

Konstanta	*Typ*	*Hodnota*
Default background color	Long Integer	-2
Default dark shadow color	Long Integer	-3
Default foreground color	Long Integer	-1
Default light shadow color	Long Integer	-4

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro standardní systémové podpisy

Konstanta	*Typ*	*Hodnota*
Picture Document	String	PICT
QT Animation compressor	String	rle
QT Compact video compressor	String	cdvc
QT Graphics compressor	String	smc
QT Photo compressor	String	jpeg
QT Raw compressor	String	raw
QT Video compressor	String	rpza
Text Document	String	TEXT
Windows MIDI Document	String	MID
Windows Sound Document	String	WAV
Windows Video Document	String	AVI

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro systémové dokumenty

Konstanta	*Typ*	*Hodnota*
Get Pathname	Long Integer	3
Is a directory	Long Integer	0
Is a document	Long Integer	1
Read and Write	Long Integer	0
Read Mode	Long Integer	2
Write Mode	Long Integer	1

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro čísla portů TCP

Konstanta	*Typ*	*Hodnota*
TCP Authentication	Long Integer	113
TCP DNS	Long Integer	53
TCP Finger	Long Integer	79
TCP FTP Control	Long Integer	21
TCP FTP Data	Long Integer	20
TCP Gopher	Long Integer	70
TCP HTTP WWW	Long Integer	80
TCP IMAP3	Long Integer	220
TCP Kerberos	Long Integer	88
TCP KLogin	Long Integer	543
TCP Nickname	Long Integer	43
TCP NNTP	Long Integer	119
TCP NTalk	Long Integer	518
TCP NTP	Long Integer	123
TCP PMCP	Long Integer	1643
TCP PMD	Long Integer	1642
TCP POP3	Long Integer	110
TCP Printer	Long Integer	515
TCP RADACCT	Long Integer	1646
TCP RADIUS	Long Integer	1645
TCP Remote Cmd	Long Integer	514
TCP Remote Exec	Long Integer	512
TCP Remote Login	Long Integer	513
TCP Router	Long Integer	520
TCP SMTP	Long Integer	25
TCP SNMP	Long Integer	161
TCP SNMPTRAP	Long Integer	162
TCP SUN RPC	Long Integer	111
TCP Talk	Long Integer	517
TCP Telnet	Long Integer	23
TCP TFTP	Long Integer	69
TCP UUCP	Long Integer	540
TCP UUCP RLOGIN	Long Integer	541

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro formát zobrazení času

Konstanta	*Typ*	*Hodnota*
HH MM	Long Integer	2
HH MM AM PM	Long Integer	5
HH MM SS	Long Integer	1
Hour Min	Long Integer	4
Hour Min Sec	Long Integer	3

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro druh okna

Konstanta	*Typ*	*Hodnota*
External window	Long Integer	5
Floating window	Long Integer	14
Modal dialog	Long Integer	9
Regular window	Long Integer	8

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro pozici okna

Konstanta	*Typ*	*Hodnota*
At the Bottom	Long Integer	393216
At the Top	Long Integer	327680
Horizontally Centered	Long Integer	65536
On the Left	Long Integer	131072
On the Right	Long Integer	196608
Vertically Centered	Long Integer	262144

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Konstanty pro Log události Window NT

Konstanta	*Typ*	*Hodnota*
Error Message	Long Integer	2
Information Message	Long Integer	0
Warning Message	Long Integer	1

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , A

ABORT
Abs (číslo) → Číslo
ACCEPT
ACCUMULATE (data{; data2; ...;dataN})
ACI folder → Řetězec
Activated
ADD DATA SEGMENT
ADD RECORD ({tabulka}{; }{*})
ADD SUBRECORD (podtabulka; formulář{; *})
Add to date (datum; roků; měsíců; dní) → Datum
ADD TO SET ({tabulka ;}set)
After
ALERT (zpráva{; text tlačítka OK})
ALL RECORDS ({tabulka})
ALL SUBRECORDS (podtabulka)
Append document (dokument{; typSouboru}) → DorRef
APPEND MENU ITEM (nabídka; položkaText{; proces})
APPEND TO CLIPBOARD (DruhDat; data)
APPEND TO LIST (list; PoložkaText; PoložkaRef{; podlist{; rozšiř}})
Application file → Řetězec
Application type → Long Integer
Application version ({*}) → Řetězec
APPLY TO SELECTION ({tabulka; }tvrzení)
APPLY TO SUBSELECTION (podtabulka; tvrzení)
Arctan (číslo) → Číslo
Array
Array a jazyk 4D
Array a objekty formuláře
Array a paměť
Array a Ukazatele
ARRAY BOOLEAN (NázevArray; velikost {;velikost2})
ARRAY DATE (NázevArray; velikost {;velikost2})
ARRAY INTEGER (NázevArray; Velikost {; velikost2})
ARRAY LONGINT (NázevArray; velikost {;velikost2})
ARRAY PICTURE (NázevArray; velikost {;velikost2})
ARRAY POINTER (NázevArray; velikost {;velikost2})
ARRAY REAL (NázevArray; velikost {;velikost2})
ARRAY STRING(délka řetězce; NázevArray; velikost {;velikost2})
ARRAY TEXT (NázevArray; velikost {;velikost2})
ARRAY TO LIST (array; Seznam {; OdkPol})
ARRAY TO SELECTION (array; pole {;array2; pole2; .....; arrayN;poleN})
ARRAY TO STRING LIST (řatězec; resID{; resSoubor})
Ascii (znak) → Číslo
AUTOMATIC RELATIONS (jedinci{; skupiny})
Average (čís.řada) → Číslo

4th Dimension 6.5, Příkazy v abecedním pořadí , B

BEEP
Before
Before selection ({tabulka}) → Logické
Before subselection (podtabulka) > Logické
Bitwise operátory
BLOB PROPERTIES (blob; zabaleno{; rozbalenáVel{; AktVelikost}})
BLOB size (blob) → Číslo
BLOB TO DOCUMENT (dokument; blob{; *})
BLOB to integer (blob; pořadíBytů{; offset}) → Číslo
BLOB to list (blob{; offset}) → OdkSeznam
BLOB to longint (blob; pařadíBytů{; offset}) → Číslo
BLOB to real (blob; FormátReal{; offset})
BLOB to text (blob; FormátText{; offset{; DélkaTextu}})
BLOB TO VARIABLE (blob; proměnná{; offset})
BOOLEAN ARRAY FROM SET (logickéArray{; sada})
BREAK LEVEL (úroveň {; zlomStr})
BRING TO FRONT (proces)
BUTTON TEXT ({*; }objekt; TextTlačítka)

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , C

C_BLOB ({metoda; }proměnná{; proměnná2;...;proměnnáN})
C_BOOLEAN ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_DATE ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_GRAPH ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_INTEGER ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_LONGINT ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_PICTURE ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_POINTER ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_REAL ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_STRING ({metoda}velikost; proměnná{; proměnná2;...;proměnnáN})
C_TEXT ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_TIME ({metoda}proměnná{; proměnná2;...;proměnnáN})
CALL PROCESS (proces)
CANCEL
CANCEL TRANSACTION
Caps lock down → Logické
Case of...Else...End case
CLEAR CLIPBOARD
CLEAR LIST (listRef{; *})
CLEAR NAMED SELECTION (název)
CLEAR SEMAPHORE (semafor)
CLEAR SET (set)
CLEAR VARIABLE (proměnná)
CLOSE DOCUMENT (docRef)
CLOSE RESOURCE FILE (resSoubor)
CLOSE WINDOW {(RefExtOkno)}
Command name (příkaz) → Řetězec
Compiled application → Logické
COMPRESS BLOB (blob{; komprese})
COMPRESS PICTURE (obrázek; metoda; kvalita)
COMPRESS PICTURE FILE (dokument; metoda; kvalita)
CONFIRM (zpráva{; titul tlačítka OK{; titul tlačítka Storno}})
COPY ARRAY (zdroj; cíl)
COPY BLOB (zdrBLOB; cílBLOB; zdrOffset; cílOffset; délka)
COPY DOCUMENT (zdrojNázev; cílNázev{; *})
Copy list (list) → ListRef
COPY NAMED SELECTION ({tabulka; }název)
COPY SET (zdrSet; cílSet)
Cos (číslo) → Číslo
Count fields (tabulkaČíslo | ukazTabulky) → Číslo
Count list items (list) → Long Integer
Count menu items (nabídka{; proces}) → Číslo
Count menus ({proces}) → Číslo
Count parameters → Číslo
Count screens → Číslo
Count tables → Číslo

Count tasks → Integer
Count user processes → Integer
Count users → Integer
Create document (dokument{; typSouboru}) → DocRef
CREATE EMPTY SET ({tabulka ;}set)
CREATE FOLDER (cestasložky)
CREATE RECORD ({tabulka})
CREATE RELATED ONE (pole)
Create resource file (resSoubor{; SouborTyp}) → DocRef
CREATE SELECTION FROM ARRAY ({tabulka; } záznArray{; název})
CREATE SET ({tabulka ;}set)
CREATE SET FROM ARRAY ({tabulka; }záznArray{; setNázev})
CREATE SUBRECORD (podtabulka)
Current date ({*}) → Datum
Current default table → Ukazatel
Current form page → Číslo
Current form table → Ukazatel
Current machine → Řetězec
Current machine owner → Řetězec
Current process → Číslo
Current time ({*})→ Čas
Current user → Řetězec
CUT NAMED SELECTION ({tabulka; }název)

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , Č

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , D

Data file {(část)} → Řetězec
DATA SEGMENT LIST (Části)
Database event → Číslo
Date (ŘetězecDatum) → Datum
Datumové operátory
Day number (datum) → Číslo
Day of (datum) → Číslo
Deactivated
Dec (číslo) → Číslo
DEFAULT TABLE (tabulka)
DELAY PROCESS (proces; trvání)
DELETE DOCUMENT (dokument)
DELETE ELEMENT (array; kde{; kolik})
DELETE FROM BLOB (blob; offset; délka)
DELETE LIST ITEM (List; položkaRef | *{; *})
DELETE MENU ITEM (nabídka; položkaNab{; proces})
DELETE RECORD ({tabulka})
DELETE RESOURCE (resTyp; resID{; resSoubor})
DELETE SELECTION ({tabulka})
Delete string (zdroj; kde; početZnaků) → Řetězec
DELETE SUBRECORD (podtabulka)
DELETE USER (uživID)
DIALOG ({tabulka; }formulář)
DIFFERENCE (set; odečítSet; výslSet)
DISABLE BUTTON({*; }objekt)
DISABLE MENU ITEM (nabídka; položkaNab{; proces})
DISPLAY RECORD ({tabulka})
DISPLAY SELECTION ({tabulka}{; *}{; *})
DISTINCT VALUES (pole; array)
Document creator (dokument) → Řetězec
DOCUMENT LIST (cesta; dokumenty)
DOCUMENT TO BLOB (dokument; blob{; *})
Document type (dokument) → Řetězec
DRAG AND DROP PROPERTIES (zdrObjekt; zdrPrvek; zdrProces)
DRAG WINDOW
Drop position → Číslo
Typy dat
DUPLICATE RECORD ({tabulka})
During
Dvojrozměrné array

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , E

EDIT ACCESS
ENABLE BUTTON({*; }objekt)
ENABLE MENU ITEM (nabídka; položkaNab{; proces})
End selection ({tabulka}) → Logické
End subselection (podtabulka) → Logické
ERASE WINDOW {(okno)}
Euro Converter (hodnota; zMěny; doMěny) → Real
Poznámka: Tento příkaz budete používat jen vyjímečně.
EXECUTE ON CLIENT (NázevKlient; metoda{; param}{; param2; ...; paramN})
Execute on server (metoda; stack{; název{; param{; param2; ...; paramN}{; *}}}) → Číslo
Exp (číslo) → Číslo
EXPAND BLOB(blob)
EXPORT DATA (SouborNázev{; projekt{; *}})
EXPORT DIF ({tabulka; }dokument)
EXPORT TEXT ({tabulka; }dokument)
EXPORT TEXT ({tabulka; }dokument)

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , F

False → Logické
Field (tabulkačíslo | UkazatelPole{; polečíslo}) -→> Číslo | Ukazatel
Field name (tabulkaČíslo; poleČíslo) | (ukazPole) → Řetězec
FILTER EVENT
FILTER KEYSTROKE (ZnakFiltru)
Find in array (array; hodnota {; start}) → číslo
Find index key (indexovanéPole; hodnota) → LongInt
Find window (levá; vrch{; oknočást}) -→> WinRef
FIRST PAGE
FIRST RECORD ({tabulka})
FIRST SUBRECORD (podtabulka)
FLUSH BUFFERS
FOLDER LIST (cesta; složky)
FONT ({*; }objekt; písmo)
FONT LIST (písma)
Font name (ČísloPísma) → Řetězec
Font number (NázevPísma) → Číslo
FONT SIZE ({*; }objekt; velikost)
FONT STYLE ({*; }objekt; styl)
For...End for
Form event → Číslo
Frontmost process {(*)} → Integer
Frontmost window {(*)} → WinRef

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , G

Gestalt (dotaz; hodnota) → Číslo
GET CLIPBOARD (drihDat; data)
Get database parameter ({tabulka; }volba) → Longint
Get document position (docRef) → Číslo
GET DOCUMENT PROPERTIES (dokument; zamčen; neviděn; vytvořenD; vytvořenČ; upravenD; upravenČ)
Get document size (dokument{; *}) → Číslo
Get edited text → Text
GET FIELD PROPERTIES (tabulkačíslo; polečíslo | ukazpole; poleTyp{; poleDel{; indexované}})
GET FORM PROPERTIES ({tabulka; }formulář; šířka; výška{; počStran{; PromŠíř{; PromVýš{; titul}}}})
GET GROUP LIST (NázvySkupin; číslaSkupin)
GET GROUP PROPERTIES (skupinaID; název; vlastník{; členi})
GET HIGHLIGHT (oblast; počvyb; konecvyb)
GET ICON RESOURCE (resID; resData{; resSoubor})
Get indexed string (resID; strID{; resSoubor) → Řetězec
GET LIST ITEM (list; PoložkaPoz; PoložkaRef; PoložkaText{; podlist{; rozšiř}})
GET LIST ITEM PROPERTIES (list; PoložkaRef; dostup{; styl{; ikona}})
GET LIST PROPERTIES (list; vzhled{; ikona{; výškařád}})
Get menu item (nabídka; položkaNab{; proces}) → Řetězec
Get menu item key (nabídka; položkaNab{; proces}) → Číslo
Get menu item mark (nabídka; položkaNab{; proces}) → Řetězec
Get menu item style (nabídka; položkaNab{; proces}) → Číslo
Get menu title (nabídka{; proces}) → Řetězec
GET MOUSE (myšX; myšY; myšTlačítko{; *})
GET OBJECT RECT ({*; }objekt; levá; vrch; pravá; spodní)
GET PICTURE FROM CLIPBOARD (obrázek)
GET PICTURE FROM LIBRARY (picRef; obrázek)
GET PICTURE RESOURCE (resID; resData{; resSoubor})
Get platform interface → Číslo
Get pointer (NázevProm) → Ukazatel
GET PROCESS VARIABLE (proces; zdrojProm; cílProm{; zdrojProm2; cílProm2; ...; zdrojPromN; cílPromN})
GET REGISTERED CLIENTS (seznamKlientů; metody)
GET RESOURCE (resTyp; resID; resData{; resSoubor})
Get resource name (resTyp; resID{; resSoubor}) → Řetězec
Get resource properties (resTyp; resID{; resSoubor}) → Číslo
Get string resource (resID{; resSoubor}) → Řetězec
Get text from clipboard → Řetězec
Get text resource (resID{; resSoubor}) → Text
GET USER LIST (JménaUživatelů; ČíslaUživatelů)
GET USER PROPERTIES (uživID; jméno; startup; heslo; početPřihl; poslPřihl{; členem})
GET WINDOW RECT (levá; vrch; pravá; spodní{; okno})
Get window title {(okno)} → Řetězec
GOTO AREA ({*; }objekt)
GOTO PAGE (číslo stránky)
GOTO RECORD ({tabulka; }záznam)
GOTO SELECTED RECORD ({tabulka; }záznam)
GOTO XY (x;y)

GRAPH (oblastgraf; grafčíslo; xpopis; yprvky{; yprvky2; ...; yprvkyN})
GRAPH SETTINGS (graf; xmin; xmax; ymin; ymax; xprop; xsíť; ysíť; titul{; titul2; ...; titulN})
GRAPH TABLE ({tabulka; }grafTyp; x pole; y pole{; y pole2; ...; y poleN})

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , H

HIDE MENU BAR
HIDE PROCESS (proces)
HIDE TOOL BAR
HIDE WINDOW ({okno})
HIGHLIGHT RECORDS ({NázevSady})
HIGHLIGHT TEXT (oblast; počvyb; konecvyb)

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , CH

CHANGE ACCESS
CHANGE PASSWORD (heslo)
Change string (zdroj; novéznaky; kde) → Řetězec
CHANGE WEB LICENCE
Char (asciikód) → Řetězec

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , I

Identifikátory
IDLE
If...Else...End if
IMPORT DATA (SouborNázev{; projekt{; *}})
IMPORT DIF ({tabulka; }dokument)
IMPORT SYLK ({tabulka; }dokument)
IMPORT TEXT ({tabulka; }dokument)
In break
In footer
In header
In transaction → Logické
INPUT FORM ({tabulka; }formulář{; *})
INSERT ELEMENT (array; kam; {; kolik})
INSERT IN BLOB (blob; offset; délka{; vyplnit})
INSERT LIST ITEM (list; PředPolRef | *; PoložkaText; PoložkaRef{; podlist{; rozšiř}})
INSERT MENU ITEM (nabídka; zaPoložku; PoložkaText{; process})
Insert string (zdroj; co; kam) → Řetězec
Int (číslo) → Číslo
INTEGER TO BLOB (integer; blob; PořadíBytů{; offset | *})
INTERSECTION (set1; set2; VýsleSet)
INVERT BACKGROUND ({*; }textProm | textPole)
Is a list (list) → Logické
Is a variable (Ukazatel) → Logické
Is in set (set) → Logické
Is new record ({tabulka}) → Logické
Is record loaded ({tabulka}) → Logické
Is user deleted (uživatelČíslo) → Logické
ISO to Mac (text) → Řetězec

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , K

Keystroke → řetězec
Konstanty
Krokování procesu neviditelného nebo neprovádějícího kód

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , L

Ladění
Last object → Ukazatel
LAST PAGE
LAST RECORD ({tabulka})
LAST SUBRECORD (podtabulka)
Length (řetězec) → Číslo
Level → Číslo
List item parent (list; PoložkaRef) → Číslo
List item position (list; PoložkaRef) → Číslo
LIST TO ARRAY (seznam; array {; OdkPol})
LIST TO BLOB (seznam; blob{; *})
LOAD COMPRESSED PICTURE FROM FILE (dokument; metoda; kvalita; obrázek)
Load list (NázevSeznamu) → ListRef
LOAD RECORD ({tabulka})
LOAD SET ({tabulka; }set; dokument)
LOAD VARIABLES (dokument; prom{; prom2; ...; promN})
Locked ({tabulka}) → Logické
LOCKED ATTRIBUTES ({tabulka; }proces; uživatel; stroj; NázevProcesu)
Log(číslo) → Číslo
LOG EVENT (zpráva{; důležitost})
Logické operátory
Logické příkazy
LONGINT ARRAY FROM SELECTION ({tabulka; }záznamArray{; výběr})
LONGINT TO BLOB (longInt; blob; PořadíBytů{; offset | *})
Lowercase (Řetězec) → Řetězec

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , M

Mac to ISO (text) → Řetězec
Mac to Win (text) → Řetězec
Macintosh command down → Logické
Macintosh control down → Logické
Macintosh option down → Logické
MAP FILE TYPES (MacOS; Windows; obsah)
Matematické řady
Max (čís.řada) → Číslo
MAXIMIZE WINDOW {(okno)}
MENU BAR (záhlnabídky{; proces}{; *})
Menu bar height → Číslo
Menu bar screen → Číslo
Menu selected → Číslo
MESSAGE (zpáva)
MESSAGES OFF
MESSAGES ON
Metoda databáze Při ověření WEB
Metoda databáze Při spuštění
Metoda databáze Při ukončení
Metoda databáze Při Web spojení
Metody
Metody databáze
Metody projektu
Milliseconds Ć Číslo
Min (čís.řada) → Číslo
MINIMIZE WINDOW {(okno)}
Mod (číslo1; číslo2) -→ Číslo
Modified (pole) → Logické
Modified record ({tabulka}) → Logické
MODIFY RECORD ({tabulka}{; }{*})
MODIFY SELECTION ({tabulka}{; *}{; *})
MODIFY SUBRECORD (podtabulka; formulář{; *})
Month of (datum) → Číslo
MOVE DOCUMENT (zdrCesta; cílCesta)
MOVE OBJECT ({*; }objekt; posunH; posunV{; změnaH{; změnaV{; *}}})

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , N

New list → ListRef
New process (metoda; stack{; název{; param{; param2; ...; paramN}{; *}}}) → Číslo
NEXT PAGE
NEXT RECORD ({tabulka})
NEXT SUBRECORD (podtabulka)
Next window (okno) → Číslo
Nil (Ukazatel) → Logické
NO TRACE
Not (Logické) → Logické
Num (výraz) → Číslo

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , O

O číslech záznamů
Obrázkové operátory
Obrázky
Okno chyby syntaxe
Old (pole) → Výraz
OLD RELATED MANY (pole)
OLD RELATED ONE (pole)
ON ERR CALL (MetodaChyby)
ON EVENT CALL (MetodaUdálostí{; NázevProcesu})
ON SERIAL PORT CALL (serialMetoda{; proces})
ONE RECORD SELECT ({tabulka})
Open document (dokument{; TypSouboru{; mod}}) → DocRef
Open external window (levá; vrch; pravá; spodní; typ; titlul; OblastPlugIn) → WinRef
Open form window ({tabulka; }formulářNázev{; typ{; hPoz{; vPoz{; *}}}}) → WinRef
Open resource file (resSoubor{; SouborTyp}) → DocRef
OPEN WEB URL (url{; *})
Open window (levá; vrch; pravá; spodní{; typ{; titul{; řídícíokénko}}}){ → WinRef }
Operátory
Operátory porovnání
Operátory řetězce
ORDER BY ({tabulka}{; pole}{; >nebo<}{; pole2; >nebo<2; ...; poleN; >nebo<N})
ORDER BY FORMULA (tabulka{; Výraz}{; >nebo<}{; Výraz2; >nebo<2; ...; VýrazN; >nebo<N})
ORDER SUBRECORDS BY (podtabulka; podpole{; >nebo<}{; podpole2; >nebo<2; ...; podpoleN; >nebo<N})
OUTPUT FORM ({tabulka ;}formulář)
Outside call

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , P

PAGE BREAK (* | >)
PAGE SETUP (tabulka; formulář)
PAUSE PROCESS (proces)
PICT TO GIF (obr; blobGIF)
PICTURE LIBRARY LIST (picRefs; picNázvy)
PICTURE PROPERTIES (obrázek; šířka; výška; hOffset; vOffset; mod)
Picture size (obrázek) → Číslo
PLATFORM PROPERTIES (platforma; Systém; Stroj)
PLAY (objektnázev; kanál)
Pojmenovaný výběr
POP RECORD (tabulka)
Pop up menu (obsah; výchozí) -→> Číslo
Position(nalézt; řetězec) → Číslo
POST CLICK (myšX; myšY; proces; *)
POST EVENT (co; zpráva; kdy; myšX; myšY; modifikátor; proces)
POST KEY (kód; modifikátor; proces)
Použití prvku nula v array
Použití Transakcí
Používání paměti Stack pro záznam
PREVIOUS PAGE
PREVIOUS RECORD (tabulka)
PREVIOUS SUBRECORD (podtabulka)
PRINT FORM (tabulka; formulář)
PRINT LABEL (tabulka; dokument; *)
PRINT RECORD (tabulka; *)
PRINT SELECTION (tabulka; *)
PRINT SETTINGS
Printing page → Číslo
Process state (proces) → Číslo
Process aborted → Logické
Process number (název; *) → Číslo
PROCESS PROPERTIES (proces; procNázev; procStav; procČas; procVidět; jedinečnéID; původ)
Procesy
Proč ladění?
Proměnné
Přerušení
Příkazy BLOB
Příkazy kompilátoru
PUSH RECORD (tabulka)

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , Q

QUERY (tabulka; argumentdotazu ; *)
QUERY BY EXAMPLE (tabulka;*)
QUERY BY FORMULA (tabulka; VýrazDotazu)
QUERY SELECTION (tabulka; argumentdotazu ; *)
QUERY SELECTION BY FORMULA (tabulka; VýrazDotazu)
QUERY SUBRECORDS (podtabulka; VýrazDotazu)
QUIT 4D

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , R

Random → Číslo
READ ONLY (tabulka | *)
Read only state (tabulka) → Logické
READ WRITE (tabulka | *)
REAL TO BLOB (real; blob; FormátReal; offset | *)
RECEIVE BUFFER (doplProm)
RECEIVE PACKET (odkDok;dplProm; stopZnak | počZnak)
RECEIVE RECORD (tabulka)
RECEIVE VARIABLE (proměnná)
Record number (tabulka) → Číslo
Records in selection (tabulka) → Číslo
Records in set (set) → Číslo
Records in subselection (podtabulka) → Číslo
Records in table (tabulka) → Číslo
REDRAW (objekt)
REDRAW LIST (list)
REDRAW WINDOW (okno)
REDUCE SELECTION (tabulka ;počet)
REGISTER CLIENT (NázevKlienta; perioda; *)
REJECT (pole)
RELATE MANY (TabulkaJed | Pole)
RELATE MANY SELECTION (pole)
RELATE ONE (TabulkaSkupin | Pole; VýbPole)
RELATE ONE SELECTION (TabSkupin; TabJedinců)
REMOVE FROM SET (tabulka ;set)
REMOVE PICTURE FROM LIBRARY (picRef)
Repeat...Until
Replace string (zdroj; původet; novýřet; kolik) → Řetězec
REPORT (tabulka; dokument; *)
Request (zpráva; výchozí odpověď; titul OK; titul Storno) → Řetězec
RESOLVE POINTER (ukazatel; PromNazev; tabČíslo; poleČíslo)
RESOURCE LIST (resTyp; resID; resNázvy; resSoubor)
RESOURCE TYPE LIST (resTypy; resSoubor)
RESUME PROCESS (proces)
Round (zaokr; míst) → Číslo

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , Ř

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , S

Sady
SAVE LIST (list; listNázev)
SAVE OLD RELATED ONE (pole)
SAVE PICTURE TO FILE (dokument; obrázek)
SAVE RECORD (tabulka)
SAVE RELATED ONE (pole)
SAVE SET (set; dokument)
SAVE VARIABLES (dokument; prom; prom2; ...; promN)
SCAN INDEX (pole; počet ; > nebo <)
SCREEN COORDINATES (levá; vrch; pravá; spodní; obrazovka)
SCREEN DEPTH (hloubka; barva; obrazovka)
Screen height (*) → Číslo
Screen width (*) → Číslo
SEARCH BY INDEX
Select folder (zpráva) → Řetězec
SELECT LIST ITEM (list; PoložkaPoz)
SELECT LIST ITEM BY REFERENCE (list; PoložkaRef)
SELECT LOG FILE (logSoubor | *)
Selected list item (list) → Long Integer
Selected record number (tabulka) → Číslo
SELECTION RANGE TO ARRAY(start; konec; pole | tabulka; array ;pole2 | tabulka2; array2;....; poleN | tabulkaN; arrayN)
SELECTION TO ARRAY (pole | tabulka; array ;pole2 | tabulka2;....;poleN | tabulkaN; arrayN)
Self → Ukazatel
Semaphore (semafor; počet tiků) → Logické
SEND HTML BLOB (blob; typ; bezKontextu)
SEND HTML FILE (htmlSoubor)
SEND HTTP REDIRECT (url; *)
SEND PACKET (odkDok; balík)
SEND RECORD (tabulka)
SEND VARIABLE (proměnná)
Sequence number (tabulka) → Číslo
SET ABOUT (TextPoložky; metoda)
SET BLOB SIZE (blob; velikost ; vyplnit)
SET COLOR VISIBLE (*; objekt; barva)
SET CURSOR (kurzor)
SET DATABASE PARAMETER (tabulka; volba; hodnota)
SET DEFAULT CENTURY (století; dělícíRok)
SET DOCUMENT CREATOR (dokument; SouborCreator)
SET DOCUMENT POSITION (docRef; offset; ukotvení)
SET DOCUMENT PROPERTIES (dokument; zamčen; neviděn; vytvořenD; vytvořenČ; upravenD; upravenČ)
SET DOCUMENT SIZE (dokument; velikost)
SET DOCUMENT TYPE (document; SouborTyp)
SET ENTERABLE (*; vstupníOblast; dostup)
SET FIELD TITLES (tabulka | podtabulka; poleTituly; poleČísla)
SET FILTER (*; objekt; vstupFiltr)
SET FORMAT (*; objekt; formátzobr)

Set group properties (skupinaID; název; vlastník; členi) → Číslo
SET HOME PAGE (DomovStránka)
SET HTML ROOT (cestaHTML)
SET HTTP HEADER (poleHTTP)
SET CHANNEL (port | operace; nastavení | dokument)
SET CHOICE LIST (*; objekt; seznam)
SET INDEX (pole; index; mod; *)
SET LIST ITEM (list; PoložkaRef; NovýPolText; NováPolRef; podlist; rozšiř)
SET LIST ITEM PROPERTIES (list; PoložkaRef; dostup; styl; ikona)
SET LIST PROPERTIES (list; vzhled; ikona; výškařád)
SET MENU ITEM (Nabídka; PoložkaNab; TextPoložky; proces)
SET MENU ITEM KEY (nabídka; položkaNab; položkaKlíč; proces)
SET MENU ITEM MARK (nabídka; položka; značka; proces)
SET MENU ITEM STYLE (nabídka; položkaNab; PoložkaStyl ; proces)
SET PICTURE RESOURCE (resID; resData; resSoubor)
SET PICTURE TO CLIPBOARD (obrázek)
SET PICTURE TO LIBRARY (obrázek; picRef; picNázev)
SET PLATFORM INTERFACE (interface)
SET PRINT PREVIEW (náhled)
SET PROCESS VARIABLE (proces; cílProm; výraz; cílProm2; výraz2; ...; cílPromN; výrazN)
SET QUERY DESTINATION (cílovýTyp; cílovýObjekt)
SET QUERY LIMIT (limit)
SET REAL COMPARISON LEVEL (epsilon)
SET RESOURCE (resTyp; resID; resData; resSoubor)
SET RESOURCE NAME (resTyp; resID; resNázev; resSoubor)
SET RESOURCE PROPERTIES (resTyp; resID; resVlastn; resSoubor)
SET RGB COLORS (*; objekt; popředíBarva; pozadíBarva)
SET SCREEB DEPTH (hloubka; barva; obrazovka)
SET STRING RESOURCE (resID; resData; resSoubor)
SET TABLE TITLES (tabTituly; tabČísla)
SET TEXT RESOURCE (resID; resData; resSoubor)
SET TEXT TO CLIPBOARD (text)
SET TIMEOUT (sekundy)
SET TIMER (početTiků)
Set user properties (uživID; jméno; startup; heslo; početPřihl; poslPřihl; členem) → Číslo
SET VISIBLE (*; objekt; viditelné)
SET WEB DISPLAY LIMITS (početZáznamů; početStran; picRef)
SET WEB TIMEOUT (vyprší)
SET WINDOW RECT (levá; vrch; pravá; spodní; okno)
SET WINDOW TITLE (titul; okno)
Seznam přerušení
Shift down → Logické
SHOW MENU BAR
SHOW PROCESS (proces)
SHOW TOOL BAR
SHOW WINDOW (okno)
Sin (číslo) → Číslo
Size of array (array) číslo
SORT ARRAY (array;array2; ...; arrayN; > nebo <)
SORT BY INDEX
SORT LIST (list; > nebo <)
Square root (číslo) → Číslo

START TRANSACTION
START WEB SERVER
Std deviation (čís.řada) → Číslo
STOP WEB SERVER
Stránky formuláře
String (výraz; formát) → Řetězec
STING LIST TO ARRAY (resID; strArray; resSoubor)
Structure file → Řetězec
Struktura
Substring (zdroj; prvnízn; početzn) → Řetězec
Subtotal (data; zlomstr) → Číslo
Sum (čís.řada) → Číslo
Sum squares (čís.řada) → Číslo
Symboly odkazů na znaky
System folder → Řetězec
Systémové dokumenty
Systémové proměnné

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , T

Table (tabulkaČíslo | Ukazatel) → Ukazatel | Číslo
Table name (tabulkaČíslo | ukazTabulky) → Řetězec
Táhnout a vsadit
Tan (číslo) → Číslo
Temporary folder → Řetězec
Tet clipboard (druhDat) → Číslo
Test path name (cesta) → Číslo
Test semaphore (semafor) → Logické
TEXT TO BLOB (text; blob; FormátText; offset | *)
Textové parametry předávané do 4D Metod volaných přes URL
Tickcount → Číslo
Time (časovýŘetězec) → Čas
Time string (sekund) → Čas
TRACE
Trigger level → Číslo
TRIGGER PROPERTIES (úroveňTriggeru; dbUdálost; tabČíslo; záznamČíslo)
Triggery
True → Logické
Trunc (kořízn; míst) → Číslo
Type (PoleProm) → Číslo
Typy oken

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , U

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , V

Validate password (uživID; heslo) → Logické
VALIDATE TRANSACTION
VARIABLE TO BLOB (proměnná; blob; *)
VARIABLE TO VARIABLE (proces; cílProm; zdrProm; cílProm2; zdrProm2; ...; cílPromN; zdrPromN)
Variance (čís.řada) → Číslo
Version type → Long Integer
Vlastnosti objektů
VOLUME ATTRIBUTES (jenotka; velikost; užito; volno)
VOLUME LIST (jednotky)
Vytváření array
Vytvoření skupiny posuvných oblastí
Vztahy

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , W

WEB CACHE STATISTICS (stránky; hitů; užití)
Web context → Logické
Web služby, Bezpečnost připojení
Web služby, Informace o Web site
Web služby, Nastavení
Web služby, Nekontextní mód
Web služby, Podpora HTML
Web služby, Popis
Web služby, Poprvé (Část I)
Web služby, Poprvé (Část II)
Web služby, Procesy Web spojení
Web služby, Web server nastavení
Web služby, Zahrnutí HTML a JavaScriptu
While...End while
Win to Mac (text) → Řetězec
Window kind (okno)
WINDOW LIST (okna; *)
Window process (okno) → Číslo
Windows Alt down → Logické
Windows Ctrl down → Logické

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , Y

Year of (datum) → Číslo

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

4th Dimension 6.5, Příkazy v abecedním pořadí , Z

4th Dimension 6.5, Seznam témat příkazů
4th Dimension 6.5, Abecední seznam příkazů
4th Dimension 6.5, Seznam témat konstant

Syntaxe příkazů abecedně

Příkazy a odkazy pro Syntaxi

A

ABORT
Abs (číslo) → Číslo
ACCEPT
ACCUMULATE (data{; data2; ...;dataN})
ACI folder → Řetězec
Activated
ADD DATA SEGMENT
ADD RECORD ({tabulka}{; }{})*
*ADD SUBRECORD (podtabulka; formulář{; *})*
Add to date (datum; roků; měsíců; dní) → Datum
ADD TO SET ({tabulka ;}set)
After
ALERT (zpráva{; text tlačítka OK})
ALL RECORDS ({tabulka})
ALL SUBRECORDS (podtabulka)
Append document (dokument{; typSouboru}) → DorRef
APPEND MENU ITEM (nabídka; položkaText{; proces})
APPEND TO CLIPBOARD (DruhDat; data)
APPEND TO LIST (list; PoložkaText; PoložkaRef{; podlist{; rozšiř}})
Application file → Řetězec
Application type → Long Integer
Application version ({}) → Řetězec*
APPLY TO SELECTION ({tabulka; }tvrzení)
APPLY TO SUBSELECTION (podtabulka; tvrzení)
Arctan (číslo) → Číslo
ARRAY BOOLEAN (NázevArray; velikost {;velikost2})
ARRAY DATE (NázevArray; velikost {;velikost2})
ARRAY INTEGER (NázevArray; Velikost {; velikost2})
ARRAY LONGINT (NázevArray; velikost {;velikost2})
ARRAY PICTURE (NázevArray; velikost {;velikost2})
ARRAY POINTER (NázevArray; velikost {;velikost2})
ARRAY REAL (NázevArray; velikost {;velikost2})
ARRAY STRING (délka řetězce; NázevArray; velikost {;velikost2})
ARRAY TEXT (NázevArray; velikost {;velikost2})
ARRAY TO LIST (array; Seznam {; OdkPol})
ARRAY TO SELECTION (array; pole {;array2; pole2; .....; arrayN;poleN})
ARRAY TO STRING LIST (řatězec; resID{; resSoubor})
Ascii (znak) → Číslo
AUTOMATIC RELATIONS (jedinci{; skupiny})
Average (čís.řada) → Číslo

B

BEEP
Before
Before selection ({tabulka}) → Logické
Before subselection (podtabulka) > Logické
BLOB PROPERTIES (blob; zabaleno{; rozbalenáVel{; AktVelikost}})
BLOB size (blob) → Číslo
*BLOB TO DOCUMENT (dokument; blob{; *})*
BLOB to integer (blob; pořadíBytů{; offset}) → Číslo
BLOB to list (blob{; offset}) → OdkSeznam
BLOB to longint (blob; pařadíBytů{; offset}) → Číslo
BLOB to real (blob; FormátReal{; offset})
BLOB to text (blob; FormátText{; offset{; DélkaTextu}})
BLOB TO VARIABLE (blob; proměnná{; offset})
BOOLEAN ARRAY FROM SET (logickéArray{; sada})
BREAK LEVEL (úroveň {; zlomStr})
BRING TO FRONT (proces)
BUTTON TEXT ({; }objekt; TextTlačítka)*

C

C_BLOB ({metoda; }proměnná{; proměnná2;...;proměnnáN})
C_BOOLEAN ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_DATE ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_GRAPH ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_INTEGER ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_LONGINT ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_PICTURE ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_POINTER ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_REAL ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_STRING ({metoda}velikost; proměnná{; proměnná2;...;proměnnáN})
C_TEXT ({metoda}proměnná{; proměnná2;...;proměnnáN})
C_TIME ({metoda}proměnná{; proměnná2;...;proměnnáN})
CALL PROCESS (proces)
CANCEL
CANCEL TRANSACTION
Caps lock down → Logické
Case of...Else...End case
CLEAR CLIPBOARD
*CLEAR LIST (listRef{; *})*
CLEAR NAMED SELECTION (název)
CLEAR SEMAPHORE (semafor)
CLEAR SET (set)
CLEAR VARIABLE (proměnná)
CLOSE DOCUMENT (docRef)
CLOSE RESOURCE FILE (resSoubor)
CLOSE WINDOW {(RefExtOkno)}
Command name (příkaz) → Řetězec

Compiled application → Logické
COMPRESS BLOB (blob{; komprese})
COMPRESS PICTURE (obrázek; metoda; kvalita)
COMPRESS PICTURE FILE (dokument; metoda; kvalita)
CONFIRM (zpráva{; titul tlačítka OK{; titul tlačítka Storno}})
COPY ARRAY (zdroj; cíl)
COPY BLOB (zdrBLOB; cílBLOB; zdrOffset; cílOffset; délka)
*COPY DOCUMENT (zdrojNázev; cílNázev{; *})*
Copy list (list) → ListRef
COPY NAMED SELECTION ({tabulka; }název)
COPY SET (zdrSet; cílSet)
Cos (číslo) → Číslo
Count fields (tabulkaČíslo | ukazTabulky) → Číslo
Count list items (list) → Long Integer
Count menu items (nabídka{; proces}) → Číslo
Count menus ({proces}) → Číslo
Count parameters → Číslo
Count screens → Číslo
Count tables → Číslo
Count tasks → Integer
Count user processes → Integer
Count users → Integer
Create document (dokument{; typSouboru}) → DocRef
CREATE EMPTY SET ({tabulka ;}set)
CREATE FOLDER (cestasložky)
CREATE RECORD ({tabulka})
CREATE RELATED ONE (pole)
Create resource file (resSoubor{; SouborTyp}) → DocRef
CREATE SELECTION FROM ARRAY ({tabulka; } záznArray{; název})
CREATE SET ({tabulka ;}set)
CREATE SET FROM ARRAY ({tabulka; }záznArray{; setNázev})
CREATE SUBRECORD (podtabulka)
Current date ({}) → Datum*
Current default table → Ukazatel
Current form page → Číslo
Current form table → Ukazatel
Current machine → Řetězec
Current machine owner → Řetězec
Current process → Číslo
Current time ({})→ Čas*
Current user → Řetězec
CUT NAMED SELECTION ({tabulka; }název)

D

Data file {(část)} → Řetězec
DATA SEGMENT LIST (Části)
Database event → Číslo
Date (ŘetězecDatum) → Datum
Day number (datum) → Číslo

Day of (datum) → Číslo
Deactivated
Dec (číslo) → Číslo
DEFAULT TABLE (tabulka)
DELAY PROCESS (proces; trvání)
DELETE DOCUMENT (dokument)
DELETE ELEMENT (array; kde{; kolik})
DELETE FROM BLOB (blob; offset; délka)
*DELETE LIST ITEM (List; položkaRef | *{; *})*
DELETE MENU ITEM (nabídka; položkaNab{; proces})
DELETE RECORD ({tabulka})
DELETE RESOURCE (resTyp; resID{; resSoubor})
DELETE SELECTION ({tabulka})
Delete string (zdroj; kde; početZnaků) → Řetězec
DELETE SUBRECORD (podtabulka)
DELETE USER (uživID)
DIALOG ({tabulka; }formulář)
DIFFERENCE (set; odečítSet; výslSet)
DISABLE BUTTON ({; }objekt)*
DISABLE MENU ITEM (nabídka; položkaNab{; proces})
DISPLAY RECORD ({tabulka})
*DISPLAY SELECTION ({tabulka}{; *}{; *})*
DISTINCT VALUES (pole; array)
Document creator (dokument) → Řetězec
DOCUMENT LIST (cesta; dokumenty)
*DOCUMENT TO BLOB (dokument; blob{; *})*
Document type (dokument) → Řetězec
DRAG AND DROP PROPERTIES (zdrObjekt; zdrPrvek; zdrProces)
DRAG WINDOW
Drop position → Číslo
Druhy dat
DUPLICATE RECORD ({tabulka})
During

E

EDIT ACCESS
ENABLE BUTTON ({; }objekt)*
ENABLE MENU ITEM (nabídka; položkaNab{; proces})
End selection ({tabulka}) → Logické
End subselection (podtabulka) → Logické
ERASE WINDOW {(okno)}
Euro Converter (hodnota; zMěny; doMěny) → Real
Poznámk a: Tento příkaz budete používat jen vyjímečně.
EXECUTE ON CLIENT (NázevKlient; metoda{; param}{; param2; ...; paramN})
*Execute on server (metoda; stack{; název{; param{; param2; ...; paramN}{; *}}}) → Číslo*
Exp (číslo) → Číslo
EXPAND BLOB (blob)
*EXPORT DATA (SouborNázev{; projekt{; *}})*
EXPORT DIF ({tabulka; }dokument)

EXPORT TEXT (\{tabulka; }dokument)
EXPORT TEXT (\{tabulka; }dokument)

F

False → Logické
Field (tabulkačíslo | UkazatelPole{; polečíslo}) -→> Číslo | Ukazatel
Field name (tabulkaČíslo; poleČíslo) | (ukazPole) → Řetězec
FILTER EVENT
FILTER KEYSTROKE (ZnakFiltru)
Find in array (array; hodnota {; start}) → číslo
Find index key (indexovanéPole; hodnota) → LongInt
Find window (levá; vrch{; okničást}) -→> WinRef
FIRST PAGE
FIRST RECORD (\{tabulka})
FIRST SUBRECORD (podtabulka)
FLUSH BUFFERS
FOLDER LIST (cesta; složky)
FONT (\{; }objekt; písmo)*
FONT LIST (písma)
Font name (ČísloPísma) → Řetězec
Font number (NázevPísma) → Číslo
FONT SIZE (\{; }objekt; velikost)*
FONT STYLE (\{; }objekt; styl)*
For...End for
Form event → Číslo
Frontmost process {(*)} → Integer
Frontmost window {(*)} → WinRef

G

Gestalt (dotaz; hodnota) → Číslo
GET CLIPBOARD (drihDat; data)
Get database parameter (\{tabulka; }volba) → Longint
Get document position (docRef) → Číslo
GET DOCUMENT PROPERTIES (dokument; zamčen; neviděn; vytvořenD; vytvořenČ; upravenD; upravenČ)
*Get document size (dokument{; *}) → Číslo*
Get edited text → Text
GET FIELD PROPERTIES (tabulkačíslo; polečíslo | ukazpole; poleTyp{; poleDel{; indexované}})
GET FORM PROPERTIES (\{tabulka; }formulář; šířka; výška{; počStran{; PromŠíř{; PromVýš{; titul}}}})
GET GROUP LIST (NázvySkupin; číslaSkupin)
GET GROUP PROPERTIES (skupinaID; název; vlastník{; členi})
GET HIGHLIGHT (oblast; počvyb; konecvyb)
GET ICON RESOURCE (resID; resData{; resSoubor})
Get indexed string (resID; strID{; resSoubor) → Řetězec
GET LIST ITEM (list; PoložkaPoz; PoložkaRef; PoložkaText{; podlist{; rozšiř}})
GET LIST ITEM PROPERTIES (list; PoložkaRef; dostup{; styl{; ikona}})

GET LIST PROPERTIES (list; vzhled{; ikona{; výškařád}})
Get menu item (nabídka; položkaNab{; proces}) → Řetězec
Get menu item key (nabídka; položkaNab{; proces}) → Číslo
Get menu item mark (nabídka; položkaNab{; proces}) → Řetězec
Get menu item style (nabídka; položkaNab{; proces}) → Číslo
Get menu title (nabídka{; proces}) → Řetězec
*GET MOUSE (myšX; myšY; myšTlačítko{; *})*
GET OBJECT RECT ({; }objekt; levá; vrch; pravá; spodní)*
GET PICTURE FROM CLIPBOARD (obrázek)
GET PICTURE FROM LIBRARY (picRef; obrázek)
GET PICTURE RESOURCE (resID; resData{; resSoubor})
Get platform interface → Číslo
Get pointer (NázevProm) → Ukazatel
GET PROCESS VARIABLE (proces; zdrojProm; cílProm{; zdrojProm2; cílProm2; ...; zdrojPromN; cílPromN})
GET REGISTERED CLIENTS (seznamKlientů; metody)
GET RESOURCE (resTyp; resID; resData{; resSoubor})
Get resource name (resTyp; resID{; resSoubor}) → Řetězec
Get resource properties (resTyp; resID{; resSoubor}) → Číslo
Get string resource (resID{; resSoubor}) → Řetězec
Get text from clipboard → Řetězec
Get text resource (resID{; resSoubor}) → Text
GET USER LIST (JménaUživatelů; ČíslaUživatelů)
GET USER PROPERTIES (uživID; jméno; startup; heslo; početPřihl; poslPřihl{; členem})
GET WINDOW RECT (levá; vrch; pravá; spodní{; okno})
Get window title {(okno)} → Řetězec
GOTO AREA ({; }objekt)*
GOTO PAGE (číslo stránky)
GOTO RECORD ({tabulka; }záznam)
GOTO SELECTED RECORD ({tabulka; }záznam)
GOTO XY (x;y)
GRAPH (oblastgraf; grafčíslo; xpopis; yprvky{; yprvky2; ...; yprvkyN})
GRAPH SETTINGS (graf; xmin; xmax; ymin; ymax; xprop; xsíť; ysíť; titul{; titul2; ...; titulN})
GRAPH TABLE ({tabulka; }grafTyp; x pole; y pole{; y pole2; ...; y poleN})

H

HIDE MENU BAR
HIDE PROCESS (proces)
HIDE TOOL BAR
HIDE WINDOW ({okno})
HIGHLIGHT RECORDS ({NázevSady})
HIGHLIGHT TEXT (oblast; počvyb; konecvyb)

CH

CHANGE ACCESS
CHANGE PASSWORD (heslo)

Change string (zdroj; novéznaky; kde) → Řetězec
CHANGE WEB LICENCE
Char (asciikód) → Řetězec

I

IDLE
If...Else...End if
*IMPORT DATA (SouborNázev{; projekt{; *}})*
IMPORT DIF ({tabulka; }dokument)
IMPORT SYLK ({tabulka; }dokument)
IMPORT TEXT ({tabulka; }dokument)
In break → Logické
In footer → Logické
In header → Logické
In transaction → Logické
*INPUT FORM ({tabulka; }formulář{; *})*
INSERT ELEMENT (array; kam; {; kolik})
INSERT IN BLOB (blob; offset; délka{; vyplnit})
*INSERT LIST ITEM (list; PředPolRef | *; PoložkaText; PoložkaRef{; podlist{; rozšiř}})*
INSERT MENU ITEM (nabídka; zaPoložku; PoložkaText{; process})
Insert string (zdroj; co; kam) → Řetězec
Int (číslo) → Číslo
*INTEGER TO BLOB (integer; blob; PořadíBytů{; offset | *})*
INTERSECTION (set1; set2; VýsleSet)
INVERT BACKGROUND ({; }textProm | textPole)*
Is a list (list) → Logické
Is a variable (Ukazatel) → Logické
Is in set (set) → Logické
Is new record ({tabulka}) → Logické
Is record loaded ({tabulka}) → Logické
Is user deleted (uživatelČíslo) → Logické
ISO to Mac (text) → Řetězec

K

Keystroke → řetězec

L

Last object → Ukazatel
LAST PAGE
LAST RECORD ({tabulka})
LAST SUBRECORD (podtabulka)
Length (řetězec) → Číslo
Level → Číslo

List item parent (list; PoložkaRef) → Číslo
List item position (list; PoložkaRef) → Číslo
LIST TO ARRAY (seznam; array {; OdkPol})
*LIST TO BLOB (seznam; blob{; *})*
LOAD COMPRESSED PICTURE FROM FILE (dokument; metoda; kvalita; obrázek)
Load list (NázevSeznamu) → ListRef
LOAD RECORD ({tabulka})
LOAD SET ({tabulka; }set; dokument)
LOAD VARIABLES (dokument; prom{; prom2; ...; promN})
Locked ({tabulka}) → Logické
LOCKED ATTRIBUTES ({tabulka; }proces; uživatel; stroj; NázevProcesu)
Log (číslo) → Číslo
LOG EVENT (zpráva{; důležitost})
LONGINT ARRAY FROM SELECTION ({tabulka; }záznamArray{; výběr})
*LONGINT TO BLOB (longInt; blob; PořadíBytů{; offset | *})*
Lowercase (Řetězec) → Řetězec

M

Mac to ISO (text) → Řetězec
Mac to Win (text) → Řetězec
Macintosh command down → Logické
Macintosh control down → Logické
Macintosh option down → Logické
MAP FILE TYPES (MacOS; Windows; obsah)
Max (čís.řada) → Číslo
MAXIMIZE WINDOW {(okno)}
*MENU BAR (záhlnabídky{; proces}{; *})*
Menu bar height → Číslo
Menu bar screen → Číslo
Menu selected → Číslo
MESSAGE (zpáva)
MESSAGES OFF
MESSAGES ON
Milliseconds Ć Číslo
Min (čís.řada) → Číslo
MINIMIZE WINDOW {(okno)}
Mod (číslo1; číslo2) -→ Číslo
Modified (pole) → Logické
Modified record ({tabulka}) → Logické
MODIFY RECORD ({tabulka}{; }{})*
*MODIFY SELECTION ({tabulka}{; *}{; *})*
*MODIFY SUBRECORD (podtabulka; formulář{; *})*
Month of (datum) → Číslo
MOVE DOCUMENT (zdrCesta; cílCesta)
MOVE OBJECT ({; }objekt; posunH; posunV{; změnaH{; změnaV{; *}}})*

N

New list → ListRef
*New process (metoda; stack{; název{; param{; param2; ...; paramN}{; *}}}) → Číslo*
NEXT PAGE
NEXT RECORD ({tabulka})
NEXT SUBRECORD (podtabulka)
Next window (okno) → Číslo
Nil (Ukazatel) → Logické
NO TRACE
Not (Logické) → Logické
Num (výraz) → Číslo

O

Old (pole) → Výraz
OLD RELATED MANY (pole)
OLD RELATED ONE (pole)
ON ERR CALL (MetodaChyby)
ON EVENT CALL (MetodaUdálostí{; NázevProcesu})
ON SERIAL PORT CALL (serialMetoda{; proces})
ONE RECORD SELECT ({tabulka})
Open document (dokument{; TypSouboru{; mod}}) → DocRef
Open external window (levá; vrch; pravá; spodní; typ; titlul; OblastPlugIn) → WinRef
*Open form window ({tabulka; }formulářNázev{; typ{; hPoz{; vPoz{; *}}}}) → WinRef*
Open resource file (resSoubor{; SouborTyp}) → DocRef
*OPEN WEB URL (url{; *})*
Open window (levá; vrch; pravá; spodní{; typ{; titul{; řídícíokénko}}}){ → WinRef }
ORDER BY ({tabulka}{; pole}{; >nebo<}{; pole2; >nebo<2; ...; poleN; >nebo<N})
ORDER BY FORMULA (tabulka{; Výraz}{; >nebo<}{; Výraz2; >nebo<2; ...; VýrazN; >nebo<N})
ORDER SUBRECORDS BY (podtabulka; podpole{; >nebo<}{; podpole2; >nebo<2; ...; podpoleN; >nebo<N})
OUTPUT FORM ({tabulka ;}formulář)
Outside call

P

PAGE BREAK (| >)*
PAGE SETUP (tabulka; formulář)
PAUSE PROCESS (proces)
PICT TO GIF (obr; blobGIF)
PICTURE LIBRARY LIST (picRefs; picNázvy)
PICTURE PROPERTIES (obrázek; šířka; výška; hOffset; vOffset; mod)
Picture size (obrázek) → Číslo
PLATFORM PROPERTIES (platforma; Systém; Stroj)
PLAY (objektnázev; kanál)
POP RECORD (tabulka)
Pop up menu (obsah; výchozí) -→> Číslo

Position (nalézt; řetězec) → Číslo
*POST CLICK (myšX; myšY; proces; *)*
POST EVENT (co; zpráva; kdy; myšX; myšY; modifikátor; proces)
POST KEY (kód; modifikátor; proces)
PREVIOUS PAGE
PREVIOUS RECORD (tabulka)
PREVIOUS SUBRECORD (podtabulka)
PRINT FORM (tabulka; formulář)
*PRINT LABEL (tabulka; dokument; *)*
*PRINT RECORD (tabulka; *)*
*PRINT SELECTION (tabulka; *)*
PRINT SETTINGS
Printing page → Číslo
Process state (proces) → Číslo
Process aborted → Logické
*Process number (název; *) → Číslo*
PROCESS PROPERTIES (proces; procNázev; procStav; procČas; procVidět; jedinečnéID; původ)
PUSH RECORD (tabulka)

Q

*QUERY (tabulka; argumentdotazu ; *)*
QUERY BY EXAMPLE (tabulka;)*
QUERY BY FORMULA (tabulka; VýrazDotazu)
*QUERY SELECTION (tabulka; argumentdotazu ; *)*
QUERY SELECTION BY FORMULA (tabulka; VýrazDotazu)
QUERY SUBRECORDS (podtabulka; VýrazDotazu)
QUIT 4D

R

Random → Číslo
*READ ONLY (tabulka | *)*
Read only state (tabulka) → Logické
*READ WRITE (tabulka | *)*
*REAL TO BLOB (real; blob; FormátReal; offset | *)*
RECEIVE BUFFER (doplProm)
RECEIVE PACKET (odkDok;dplProm; stopZnak | počZnak)
RECEIVE RECORD (tabulka)
RECEIVE VARIABLE (proměnná)
Record number (tabulka) → Číslo
Records in selection (tabulka) → Číslo
Records in set (set) → Číslo
Records in subselection (podtabulka) → Číslo
Records in table (tabulka) → Číslo

REDRAW (objekt)
REDRAW LIST (list)
REDRAW WINDOW (okno)
REDUCE SELECTION (tabulka ;počet)
*REGISTER CLIENT (NázevKlienta; perioda; *)*
REJECT (pole)
RELATE MANY (TabulkaJed | Pole)
RELATE MANY SELECTION (pole)
RELATE ONE (TabulkaSkupin | Pole; VýbPole)
RELATE ONE SELECTION (TabSkupin; TabJedinců)
REMOVE FROM SET (tabulka ;set)
REMOVE PICTURE FROM LIBRARY (picRef)
Repeat...Until
Replace string (zdroj; původřet; novýřet; kolik) → Řetězec
*REPORT (tabulka; dokument; *)*
Request (zpráva; výchozí odpověď; titul OK; titul Storno) → Řetězec
RESOLVE POINTER (ukazatel; PromNazev; tabČíslo; poleČíslo)
RESOURCE LIST (resTyp; resID; resNázvy; resSoubor)
RESOURCE TYPE LIST (resTypy; resSoubor)
RESUME PROCESS (proces)
Round (zaokr; míst) → Číslo

S

SAVE LIST (list; listNázev)
SAVE OLD RELATED ONE (pole)
SAVE PICTURE TO FILE (dokument; obrázek)
SAVE RECORD (tabulka)
SAVE RELATED ONE (pole)
SAVE SET (set; dokument)
SAVE VARIABLES (dokument; prom; prom2; ...; promN)
SCAN INDEX (pole; počet ; > nebo <)
SCREEN COORDINATES (levá; vrch; pravá; spodní; obrazovka)
SCREEN DEPTH (hloubka; barva; obrazovka)
Screen height (*) → Číslo
Screen width (*) → Číslo
SEARCH BY INDEX
Select folder (zpráva) → Řetězec
SELECT LIST ITEM (list; PoložkaPoz)
SELECT LIST ITEM BY REFERENCE (list; PoložkaRef)
*SELECT LOG FILE (logSoubor | *)*
Selected list item (list) → Long Integer
Selected record number (tabulka) → Číslo
SELECTION RANGE TO ARRAY (start; konec; pole | tabulka; array ;pole2 | tabulka2; array2;....; poleN | tabulkaN; arrayN)
SELECTION TO ARRAY (pole | tabulka; array ;pole2 | tabulka2;....;poleN | tabulkaN; arrayN)
Self → Ukazatel
Semaphore (semafor; počet tiků) → Logické
SEND HTML BLOB (blob; typ; bezKontextu)
SEND HTML FILE (htmlSoubor)

*SEND HTTP REDIRECT (url; *)*
SEND PACKET (odkDok; balík)
SEND RECORD (tabulka)
SEND VARIABLE (proměnná)
Sequence number (tabulka) → Číslo
SET ABOUT (TextPoložky; metoda)
SET BLOB SIZE (blob; velikost ; vyplnit)
SET COLOR VISIBLE (; objekt; barva)*
SET CURSOR (kurzor)
SET DATABASE PARAMETER (tabulka; volba; hodnota)
SET DEFAULT CENTURY (století; dělícíRok)
SET DOCUMENT CREATOR (dokument; SouborCreator)
SET DOCUMENT POSITION (docRef; offset; ukotvení)
SET DOCUMENT PROPERTIES (dokument; zamčen; neviděn; vytvořenD; vytvořenČ; upravenD; upravenČ)
SET DOCUMENT SIZE (dokument; velikost)
SET DOCUMENT TYPE (document; SouborTyp)
SET ENTERABLE (; vstupníOblast; dostup)*
SET FIELD TITLES (tabulka | podtabulka; poleTituly; poleČísla)
SET FILTER (; objekt; vstupFiltr)*
SET FORMAT (; objekt; formátzobr)*
Set group properties (skupinaID; název; vlastník; členi) → Číslo
SET HOME PAGE (DomovStránka)
SET HTML ROOT (cestaHTML)
SET HTTP HEADER (poleHTTP)
SET CHANNEL (port | operace; nastavení | dokument)
SET CHOICE LIST (; objekt; seznam)*
*SET INDEX (pole; index; mod; *)*
SET LIST ITEM (list; PoložkaRef; NovýPolText; NováPolRef; podlist; rozšiř)
SET LIST ITEM PROPERTIES (list; PoložkaRef; dostup; styl; ikona)
SET LIST PROPERTIES (list; vzhled; ikona; výškařád)
SET MENU ITEM (Nabídka; PoložkaNab; TextPoložky; proces)
SET MENU ITEM KEY (nabídka; položkaNab; položkaKlíč; proces)
SET MENU ITEM MARK (nabídka; položka; značka; proces)
SET MENU ITEM STYLE (nabídka; položkaNab; PoložkaStyl ; proces)
SET PICTURE RESOURCE (resID; resData; resSoubor)
SET PICTURE TO CLIPBOARD (obrázek)
SET PICTURE TO LIBRARY (obrázek; picRef; picNázev)
SET PLATFORM INTERFACE (interface)
SET PRINT PREVIEW (náhled)
SET PROCESS VARIABLE (proces; cílProm; výraz; cílProm2; výraz2; ...; cílPromN; výrazN)
SET QUERY DESTINATION (cílovýTyp; cílovýObjekt)
SET QUERY LIMIT (limit)
SET REAL COMPARISON LEVEL (epsilon)
SET RESOURCE (resTyp; resID; resData; resSoubor)
SET RESOURCE NAME (resTyp; resID; resNázev; resSoubor)
SET RESOURCE PROPERTIES (resTyp; resID; resVlastn; resSoubor)
SET RGB COLORS (; objekt; popředíBarva; pozadíBarva)*
SET SCREEB DEPTH (hloubka; barva; obrazovka)
SET STRING RESOURCE (resID; resData; resSoubor)
SET TABLE TITLES (tabTituly; tabČísla)
SET TEXT RESOURCE (resID; resData; resSoubor)
SET TEXT TO CLIPBOARD (text)

SET TIMEOUT (sekundy)
SET TIMER (početTiků)
Set user properties (užívID; jméno; startup; heslo; početPřihl; poslPřihl; členem) → Číslo
SET VISIBLE (; objekt; viditelné)*
SET WEB DISPLAY LIMITS (početZáznamů; početStran; picRef)
SET WEB TIMEOUT (vyprší)
SET WINDOW RECT (levá; vrch; pravá; spodní; okno)
SET WINDOW TITLE (titul; okno)
Shift down → Logické
SHOW MENU BAR
SHOW PROCESS (proces)
SHOW TOOL BAR
SHOW WINDOW (okno)
Sin (číslo) → Číslo
Size of array (array) číslo
SORT ARRAY (array;array2; ...; arrayN; > nebo <)
SORT BY INDEX
SORT LIST (list; > nebo <)
Square root (číslo) → Číslo
START TRANSACTION
START WEB SERVER
Std deviation (čís.řada) → Číslo
STOP WEB SERVER
String (výraz; formát) → Řetězec
STING LIST TO ARRAY (resID; strArray; resSoubor)
Structure file → Řetězec
Substring (zdroj; prvnízn; početzn) → Řetězec
Subtotal (data; zlomstr) → Číslo
Sum (čís.řada) → Číslo
Sum squares (čís.řada) → Číslo
Symboly odkazů na znaky
System folder → Řetězec

T

Table (tabulkaČíslo | Ukazatel) → Ukazatel | Číslo
Table name (tabulkaČíslo | ukazTabulky) → Řetězec
Tan (číslo) → Číslo
Temporary folder → Řetězec
Tet clipboard (druhDat) → Číslo
Test path nam e (cesta) → Číslo
Test semaphore (semafor) → Logické
*TEXT TO BLOB (text; blob; FormátText; offset | *)*
Tickcount → Číslo
Time (časovýŘetězec) → Čas
Time string (sekund) → Čas
TRACE
Trigger level → Číslo
TRIGGER PROPERTIES (úroveňTriggeru; dbUdálost; tabČíslo; záznamČíslo)

Triggery
True → Logické
Trunc (kořízn; míst) → Číslo
Type (PoleProm) → Číslo
Typy oken

U

Undefined (proměnná) → Logické
UNION (set1; set2; výsleSet)
UNLOAD RECORD (tabulka)
UNREGISTER CLIENT
Uppercase (řetězec) → Řetězec
*USE ASCII MAP (mapa | *; mapInOut)*
USE NAMED SELECTION (název)
USE SET (set)
User in Group (uživatel; skupina) → Logické

V

Validate password (uživID; heslo) → Logické
VALIDATE TRANSACTION
*VARIABLE TO BLOB (proměnná; blob; *)*
VARIABLE TO VARIABLE (proces; cílProm; zdrProm; cílProm2; zdrProm2; ...; cílPromN; zdrPromN)
Variance (čís.řada) → Číslo
Version type → Long Integer
Vlastnosti objektů
VOLUME ATTRIBUTES (jenotka; velikost; užito; volno)
VOLUME LIST (jednotky)

W

WEB CACHE STATISTICS (stránky; hitů; užití)
Web context → Logické
While...End while
Win to Mac (text) → Řetězec
Window kind (okno)
*WINDOW LIST (okna; *)*
Window process (okno) → Číslo
Windows Alt down → Logické
Windows Ctrl down → Logické

Y

Year of (datum) → Číslo